ROBOTIC WORKSHOP

MB230 INSTRUCTION MANUAL

Multibotics

MB230

ROBOTIC WORKSHOP

INSTRUCTION MANUAL

VERSION 2.1

INTRODUCTION

Welcome to the world of MULTIBOTICS.

The MULTIBOTICS concept was developed for those of you who want to learn more about how computers and the real world interact.

Our entire product line is designed around a creative environment which we call the Robotic Workshop. The term "workshop" is used to describe the interactive nature of MULTIBOTICS. We've included several example projects to get you started, but our goal is to teach you how to design your own. That's when you'll discover the real fun and challenge of MULTIBOTICS.

Please read this manual carefully. The material begins at a very basic level and becomes more technical as you progress.

The example projects are designed to be both fun and educational. If you're a beginner, they'll help you learn about computers and about basic principles of engineering and science. It is not necessary that you understand programming to build or perform these projects. If you're a "BASIC" programmer, you'll want to explore on your own. The examples demonstrate how to use the Robotic commands in your own programs. We suggest that you do the projects in chronological order, as some are designed to build on what was learned previously. If you're quite advanced and serious about robotics, you may only need the examples as a reference. We recommend, however that you read all the projects and perform those that seem challenging. You'll find valuable information on the interface unit, the operating software, and the mechanical components.

Regardless of your knowledge of computers, we hope that you'll apply your creativity and explore the world of MULTIBOTICS.

If you have comments or suggestions on how to improve our products, we would like to here from you.

Write to: Steven Witzel, President
MULTIBOTICS, INC.
2561 South 1560 West
Woods Cross, Utah 84087

TO INSURE THAT YOU RECEIVE UPDATES ON THE LATEST PROJECTS, ACCESSORIES AND SOFTWARE, PLEASE TAKE A MINUTE TO SEND IN YOUR WARRANTEE REGISTRATION CARD NOW.

ROBOTIC WORKSHOP MB230

TABLE OF CONTENTS

A LISTING OF 50 PROJECTS BY CATEGORY

Motors and Gears

Generators and Feedback

Digital Electronics and Programming

Light and Color

Counting, Measuring and I/R Detection

Multivolt, Multiscope and Electronics

Moving Vehicles

Electronic Speech and Audio

I

GETTING STARTED

CARE OF THE INTERFACE

The B100 interface is a highly sophisticated, state of the art electronic device. As such, please treat it with respect and care.

CAUTION!
NEVER INSERT OR REMOVE THE B100 WITH THE COMPUTER TURNED ON!! DAMAGE TO THE COMPUTER, B100 OR BOTH MAY OCCUR.

INSTALLATION OF THE BATTERIES

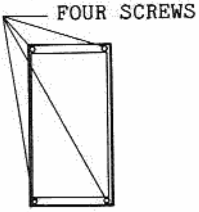
It is recommended that you use only **ALKALINE** AA batteries. These batteries should provide many hours of normal use. The batteries are used to power the motors. They have no other function.

Six "AA" batteries are required.

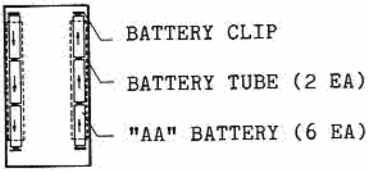
To install the batteries follow these steps:

CAUTION!
MAKE SURE THE BATTERY POLARITIES ARE CORRECT. FOLLOW THE LEGEND ON THE PRINTED CIRCUIT BOARD.

1. Remove the interface unit from the computer.
2. Turn the unit upside down.

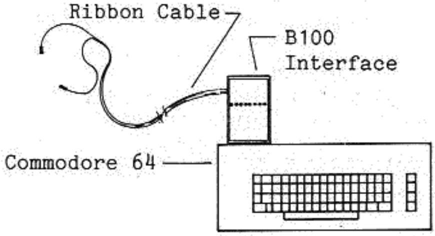


3. Remove the 4 screws from the base. (A phillips screwdriver is required.)
4. Turn it right side up and remove the top.
5. Insert 3 AA batteries in each battery tube.
6. Install the battery tubes (with batteries) between the battery clips on both sides of the interface.
7. Replace the cover.
8. Replace the 4 screws. (Do not overtighten.)



CONNECTING TO YOUR COMPUTER

The B100 interface plugs into the USER PORT of your Commodore 64. **MAKE SURE THE COMPUTER IS TURNED OFF BEFORE CONNECTING OR DISCONNECTING THE INTERFACE.**



II

ABOUT THE ELECTRONICS

SWITCHES

ON OFF MOTORS This switch provides power to the motors. It does not control anything else. In the **ON** position, the motors will respond to commands from the computer. In the **OFF** position all computer commands are disregarded, and the motors WILL NOT OPERATE. This switch is especially useful in an EMERGENCY, when you've started the motors, but things aren't going quite right. Don't forget to **TURN THE SWITCH ON WHEN YOU'RE READY TO TRY AGAIN.** This is the most common mistake people make, when learning to use the ROBOTIC WORKSHOP.

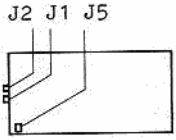
SELECTOR This switch has 3 positions for selecting the various functions of the B100:

1. **METER/SCOPE** Allows the interface to function as a voltmeter or oscilloscope depending on the software loaded into the computer.

2. **SENSORS** Selects the infrared (I/R) sensors.

3. **AUDIO** Engages the speech/audio digitizer.

JACKS

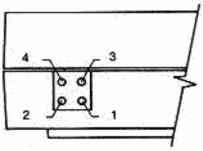


J1 OUT AUDIO ONLY This jack is used for audio output when using speech/audio function. It should be connected either to your monitor (audio in) or stereo system (aux or tape in).

J2 IN METER SENSORS AUDIO This jack is used as the input for the voltmeter and multiscope. It is also used as the audio input jack when "recording" audio signals.

J5 The pins of this jack are used as follows:

Pin 1. Advanced feature for multivolt, multiscope shunt.
Pin 2. Aux input for J2 above. (May be used as a replacement connection for J2.)
Pin 3. Ground connection.
Pin 4. Output to be used for connecting temperature probes.



MOTOR LEADS

The motor leads are connected to the B100 with a 10 foot cable. The motor leads are color coded:

Motor 1 = RED (2 leads)
Motor 2 = WHITE (2 leads)
Motor 3 = BLUE (2 leads)

If the motor spins in the reverse direction from that desired, you may simply interchange or "flip" the two motor leads.

SENSORS

The infrared (I/R) sensors are designed to work in pairs. They are set up as follows:

Sensor 7 = Output sensor (transmitter)
Senser 8 = Input sensor (receiver)

NOTE:
It is important that the correct device (transmitter or receiver) be plugged into the correct socket at the end of the 10 foot cable. If the sensors must be removed from their sockets, be sure to refer to the TECHNICAL NOTES section of this manual for correct replacement procedure.

SEPARATING THE CABLE LEADS

Both the motor and sensor leads are combined into a 10 conductor ribbon cable. As you use the cable for various projects, it will be necessary to separate the individual conductors in order to make connections to motors and to position the I/R sensors. The conductors can be easily separated by holding one in each hand and by pulling them apart as if "breaking a wishbone". The plastic which joins the conductors will "tear" quite cleanly. It is best not to separate more length than is needed.

III

ABOUT THE MECHANICS

IMPORTANT : PLEASE READ

1. The capsules snap together with octagonal connectors. They do not need to be forced or glued! No tools are necessary.

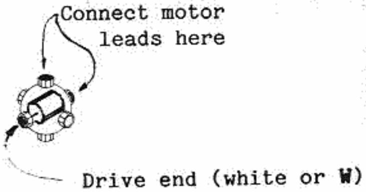
2. Motors are usually connected to a gear reduction or worm gear capsule. **ALWAYS CONNECT THE GRAY END TO THE MOTOR** or it will not work!

THE STANDARD MECHANICAL COMPONENTS

MOTOR CAPSULE
Part #1

The motor is completely powered by the B100 interface. The motor leads from the interface are color coded in pairs

RED	= Motor #1
WHITE	= Motor #2
BLUE	= Motor #3

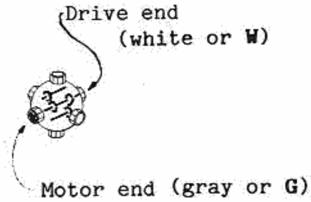


SPEED REDUCTION CAPSULE
Part #2

The speed reduction capsule slows down the speed of the motor, but increases its torque. For most projects, the motor will be connected either to the speed reduction capsule or to the worm gear capsule.

SPEED REDUCTION RATIO = 23:1

> **CAUTION!**
> ONLY CONNECT THE MOTOR TO THE **GRAY** (EASY TO TURN) END OR THE MOTOR WILL NOT SPIN.

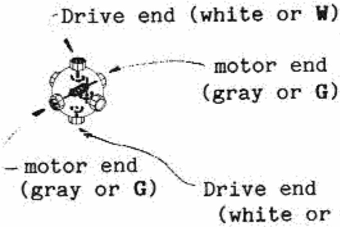


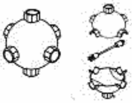
WORM GEAR CAPSULE
Part #3

The worm gear is very versatile. It may be used in place of a gear reduction capsule. The worm gear changes the drive direction 90°.

SPEED REDUCTION RATIO = 50:1

> **CAUTION!**
> ONLY CONNECT THE MOTOR TO THE **GRAY** (EASY TO TURN) END OR THE MOTOR WILL NOT SPIN.

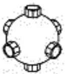
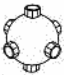
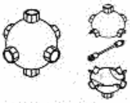




TRANSMISSION CAPSULE
Part #8

The transmission can extend the length of the motor shaft. It does not act as a gear reduction. The capsule comes apart and may be used as two support bases.

COMPLETE PARTS LIST

PART #	QTY	DESCRIPTION	PART #	QTY	DESCRIPTION
1	2	 MOTOR	17	1	AXLE SUPPORT
2	2	 SPEED REDUCTION	18	1	AXLE
3	1	WORM GEAR	25	4	WHEEL
8	1	 TRANSMISSION	28	4	TIRE
10	10	OCTAGONAL CONNECTOR	33	1	PROPELLAR
14	2	COUPLER	54	1	TURN COUPLER
15	4	COUPLER CAP			

IV

EXAMPLE PROJECTS

PREFACE

We've designed 50 projects to help you understand the MULTIBOTICS system. They'll also demonstrate several principles of robotics, engineering, and science and serve as a guide for your own projects.

One of the goals of MULTIBOTICS is to help the average person understand how computers are used in our everday world. Most people know that a computer processes their bank statement. Smaller, less frightening computers are all around us, doing all kinds of useful things.

Using your computer and the Robotic Workshop, we'll explain what they do, and how they do it.

Remember, the projects are designed to be performed in sequence. Read and conduct each one carefully and re-read each explanation if necessary. You'll find it will be an enjoyable learning experience.

PROJECT #1

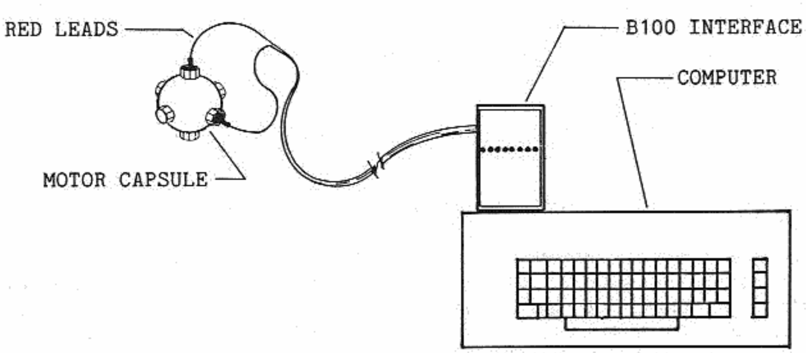
MOTORS & ELECTRICITY

DESCRIPTION:
This simple project will show you how to control the electric motors.

An electric motor is a device which converts electrical energy into rotating mechanical energy. The motors in the workshop are D.C. (Direct Current) motors and are powered by the Interface Unit. The words "current" and "voltage" are common electrical terms that we use occasionally in this manual. It will be helpful to understand the basic meaning of these words.

A simple way to discuss electricity is to compare it to the water system in your home. When you turn on the kitchen faucet, pressure in the main water line causes water to travel through the pipes and out into the sink. Similarly, voltage in the battery causes current to flow through the wires (in the 10 foot cable) and into the motor. Actually there is an important difference between the water system and the electrical system. The electrical system requires a return path or "complete circuit". We must therefore have two wires for each motor. One wire from the battery to the motor, and another wire from the motor back to the battery. The flow of water in your home is controlled by a faucet or valve. In electricity, the flow of current is controlled by a switch. In the ROBOTIC WORKSHOP, the computer acts as the switch. It can make a motor start or stop by turning the current on or off. Remember the analogy.

Electrical System		**Water System**
Voltage	=	Pressure
Current	=	Water
Wire	=	Pipe
Switch	=	Faucet



INSTRUCTIONS:

PART 1: **MOTOR ON or OFF**

(Note: If the unit fails to operate properly, turn to the TROUBLE SHOOTING section in the appendix.)

STEP 1> Move the **"MOTORS ON/OFF"** switch on the Interface unit to **OFF.**

STEP 2> Turn on the Computer and Disk Drive.

STEP 3> LOAD R.O.S. (The Robotic Operating System) as follows:

a. Put the MB230 Software Disk into the disk drive and close the door.

b. Type: **LOAD"ROS",8** and press the **<return>** key. When the file has finished loading, the computer will display **READY.**

c. Type: **RUN** and press **<return>.**

STEP 4> When the computer screen displays READY, move the **"MOTORS ON/OFF"** switch to the **ON** position.

NOTE: The voltage on the motor leads is about the same as a flashlight (3-4 volts) and will not harm you.

STEP 5> Connect the two motor leads for Motor #1 (red leads) to the motor capsule. (It does not matter which lead is connected to which plug on the motor.)

The computer (with R.O.S. Loaded) has control of the motor and will recognize certain **MOTOR COMMANDS.** You may type these commands directly or include them in your programs. You can also switch all the motors off at any time (regardless of what the computer is doing) by moving the **MOTORS ON/OFF** switch to the **OFF** position. Don't forget to return the switch to the **ON** position. The motors cannot run when this switch is **OFF.**

NOTE: When you finish typing the next step, the motor will start. You may want to have someone hold it or place it on a table. We also suggest that you read ahead one step so you'll know how to stop the motor.

STEP 6> Type: **MOTR1,30** and press **<return>.** The motor will run at full speed.

STEP 7> Type: **MOTR1,0 <return>.** The motor will stop.

"MOTR" is a **MOTOR COMMAND** which signals to the computer that you wish to do something with a motor. But the computer needs to know **WHICH MOTOR?** and **WHAT YOU WANT DONE?** So the command has two numbers following it, separated by a comma. The 1st number ("1" in the example) is the **MOTOR NUMBER** (1, 2, or 3). Remember that we connected the red leads to the motor capsule? These are the leads for MOTOR 1. MOTOR 2 has white leads and MOTOR 3 has blue leads. The 2nd number after the command is the **SPEED NUMBER.** Full speed is 30. Full stop is 0. Remember that a comma must separate the two numbers.

PART 2: **MOTOR SPEED**

It is important when performing projects to control the speed of a motor. The computer does this by systematically turning the motor on and off in a series of pulses. By varying the number of on and off pulses, it can change motor speed. For example, suppose that during a 1/10 second time interval, the computer sends 30 pulses to the motor. If all 30 pulses are **"OFF PULSES"**, the motor will not turn. If all 30 pulses are **"ON"**, then the motor will run at full speed. Any number of **"ON"** pulses between 0 and 30 will produce something between stop and full speed. If the computer sends 15 **"ON"** and 15 **"OFF"** pulses, the motor would run at about half speed. This is basically how the **"SPEED"** portion of the **"MOTR"** command works.

STEP 8> Type: **MOTR1,5 <return>** |
Type: **MOTR1,15 <return>** | The motor speed has
Type: **MOTR1,25 <return>** | gradually increased.

Type: **MOTR1,30 <return>** We're again at full speed.

Type: **MOTR1,0 <return>** To stop the motor (speed=0).

PART 3 **MOTOR DIRECTION**

It is also important when performing Robotic projects and experiments, to control the direction of a motor.

STEP 9> Type: **MOTR1,2 <return>** The motor will run at slow speed. **Note the direction.** Now, with the motor still running, reverse the red motor leads (Pull both leads out of the motor and switch their positions.) Notice that the motor has changed direction. Sometimes a motor will turn the opposite direction to that desired. You can correct the problem by reversing the leads as you just did, but there is a much easier way.

STEP 10> Type: **FLIP1 <return>** The motor will reverse.
Type: **MOTR1,0 <return>** The motor will stop.

The **FLIP** command has the same effect as switching the motor leads. After the word **FLIP**, you must specify the motor number (1, 2, or 3). **FLIP** will stay in effect until the motor is turned off or until another **FLIP** or **MOTR** command is issued to that motor.

STEP 11> Type: **MOTR1,-10 <return>** The motor will change direction.

STEP 12> Type: **MOTR1,10 <return>** The motor will again change direction.

Notice that the **sign** (+ or -) of the **MOTOR SPEED** also determines the **DIRECTION** of the motor.

STEP 13> Experiment with various combinations of speeds and directions with all three motors. Connect the white leads to the motor and use **MOTR2,SPEED#** and **FLIP2.** Then connect the blue leads to the motor and use **MOTR3,SPEED#** and **FLIP3.**

NOTE: The **MOTR** and **FLIP** commands (as with most commands) can be used in either the direct mode (as we did in the example) or within a program.

PART 4: **STOP ALL MOTORS**

There is another form of the MOTOR COMMAND that will come in handy when all three motors are running and you want to stop them quickly.

Simply type: **MOTR0 <return>**
This will send **SPEED=0** to all three motors.

IMPORTANT: PLEASE READ

We were very deliberate in our discussion of motors and electricity because it is essential to the understanding of many other projects. If you did not understand this project, please re-read it. It will save you time and frustration on future projects.

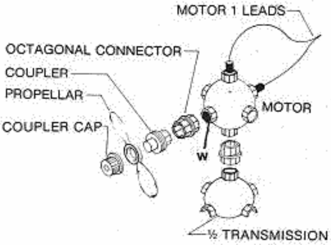
PROJECT #2

VARIABLE SPEED FAN

DESCRIPTION:
This project demonstrates the use of the **"MOTR"** and **"FLIP"** commands and discusses how information flows in and out of the computer. You will load and run a program called **"FAN"**, which will check the keyboard for any key being pressed. If a key is pressed, the program will determine which key it was and issue the appropriate **MOTR** or **FLIP** commands. You will be able to control the direction and speed of the fan from the keyboard.

INSTRUCTIONS:

STEP 1> Assemble the project as shown below.



Parts Required:

1	Motor
2	Octagonal Connectors
1	Coupler
1	Propeller
1	Coupler Cap
1	1/2 Transmission Model

NOTE: When assembling this project, be careful not to over tighten the coupler cap.

STEP 2> Make sure R.O.S. is loaded. If you are continuing on from Project #1, R.O.S. should still be in place. Do the following, just to make sure.

Type: **HELP <return>**

If R.O.S. is still there, a list of all valid R.O.S. commands will be displayed on the screen.

STEP 3> Type: **LOAD"FAN",8 <return>**
This will load the program call **FAN.** Then type **RUN <return>** to RUN the program.

The program is now checking the keyboard and waiting for a key to be pressed.

STEP 4> Experiment by pressing the following keys and observing what happens.

KEY	**RESULT**
Function Key "F1"	Increase Fan Speed
Function Key "F7"	Decrease Fan Speed
Space Bar	Stop the Program
Any Other Key	Reverse Direction

NOTE: If you stop the program and wish to re-start it, simply type **RUN <return>.**

SUMMARY: This project is designed to show how a computer program can do the work of issuing various **MOTR** commands. This time, instead of typing many different versions of **MOTR**, you press single keys. The computer (under the direction of the program call **FAN**) waits for you to press a key and then acts accordingly. The keyboard in this case is acting as the **INPUT DEVICE.** Your keystroke is **INPUT** to the computer, giving it information on what to do. The **INPUT DEVICE** could be a joystick, light pen, or any of several other connections to the outside world. These connections, through which information can enter or leave the computer, are called **INTERFACE PORTS.** Information which enters is called **INPUT** and information which leaves is called **OUTPUT.** A **PORT** can be an **INPUT PORT**, an **OUTPUT PORT**, or both. Those that are both are called **INPUT/OUTPUT** or **I/O PORTS.**

Hardware Components connected to these ports are called **DEVICES.** As with the ports, some are **INPUT DEVICES**, some are **OUTPUT DEVICES** and some are both. In this project, the keyboard is an **INPUT DEVICE.** There are two **OUTPUT DEVICES**, the screen and the B100 INTERFACE UNIT. (The B100 is actually an I/O device, but only the output portion is being used.)

Later you'll learn how to do fascinating things by manipulating various **INPUT** and **OUTPUT** devices.

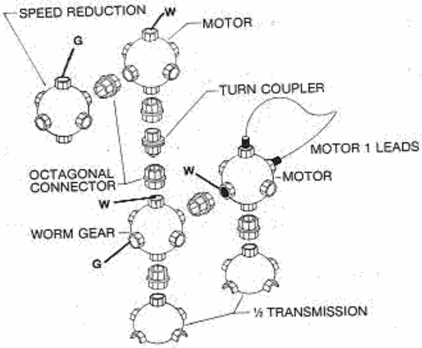
PROJECT #3

ROTATING ARM

DESCRIPTION:
In this demonstration, you'll run a program which monitors the joystick and changes the motor speed or direction when movement of the stick is detected. The program uses a new command, called **"JOY"**, to read the joystick.

INSTRUCTIONS:

STEP 1> Assemble the project as shown below.



Parts required:

2	Motors
1	Speed Reduction
1	Worm Gear
1	Transmission Capsule
6	Octagonal Connectors
1	Turn Coupler

STEP 2> Make sure R.O.S. is loaded.

STEP 3> TYPE: **LOAD"ROTATING ARM", 8 <return> then RUN <return>.**

This program requires that a joystick be plugged into joystick port #2. Direction is controlled by moving the stick to the right or left. Speed is limited by the program to a maximum of 15 and a minimum of 2. Speed is changed by moving the joystick up or down. To stop the rotating arm, press the 'FIRE' button. To start again, move the joystick to the right or left.

After experimenting with the project to observe the speed and direction changes, do the following:

STEP 3>   Stop the program by pressing the **RUN/STOP** key.

STEP 4>   List the program by typing: **LIST <return>**

The listing should appear as follows:

```
10 REM              ROTATING ARM
20 REM              USES JOYSTICK #2
30 REM              LIMITS SPEED BETWEEN 2 AND 15
40 REM
50  S=5                       :REM  INITIAL SPEED = 5
60  D=1                       :REM  1 = FORWARD, -1 = REVERSE
70  MOTR0                     :REM  ALL MOTORS OFF
80  JOY2,A                    :REM  READ JOYSTICK #2
90  IF A=1 THEN S=S+1         :REM  UP     = FASTER
100  IF A=2 THEN S=S-1        :REM  DOWN   = SLOWER
110  IF A=4 THEN D=-1         :REM  LEFT   = REVERSE
120  IF A=8 THEN D=1          :REM  RIGHT  = FORWARD
130  IF A=16 THEN D=0         :REM  BUTTON = STOP
140  SP=D*S                   :REM  COMBINE SPEED AND DIRECTION
150  MOTR1,SP                 :REM  ISSUE COMMAND TO MOTOR #1
160  PRINT SP                 :REM  PRINT SPEED ON SCREEN
170  GOTO 80                  :REM  GO READ THE STICK AGAIN
```

Notice the use of the **MOTR** command in lines 70 and 150. You'll also notice that we've used a new command **JOY** in line 80. The **JOY** command tells the computer to read a joystick and put the value in a variable. The computer will need to know which joystick and which variable. Therefore, the stick number (1 or 2) and a variable name (such as "A" in our program) must follow the command and must be separated by a comma.

For example:

```
JOY1,X         (Place stick #1 reading in "X")
JOY2,B         (Place stick #2 reading in "B")
```

You'll learn more about reading and using the joysticks in future projects.

There were two major purposes in performing this project. First, we wanted to introduce you to the **JOY** command and second we wanted to show you how to **LIST** a program. If you do not understand the program, don't be alarmed. It is not neccessary that you know the "BASIC" language in order to perform the example projects. Nevertheless, you will enjoy the ROBOTIC WORKSHOP much more if you can learn enough to write simple programs on your own.

Your Commodore 64 or 128 user manual contains good "beginners level" instructions in BASIC programming. We suggest that you LIST the programs that accompany the projects, and try to determine what is happening, while referring to the user manual. Then try to modify them or write your own. Don't be afraid to experiment. You can't hurt the system by pressing the keys. GOOD LUCK!

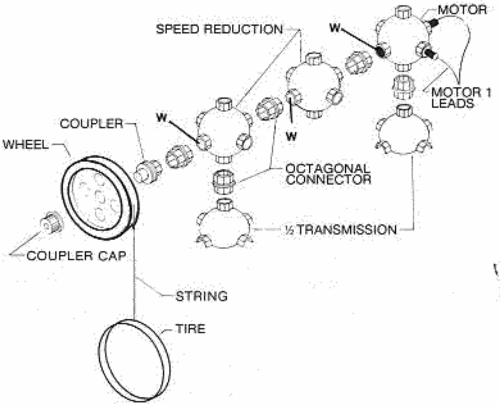
PROJECT #4

HOIST

DESCRIPTION:
This project will provide more practice in using the motor commands in the 'direct mode'.

INSTRUCTIONS:

STEP 1> Assemble the project as shown below.



Parts required:

5	Octagonal Connectors
1	Transmission Module
2	Speed Reduction
1	Coupler
1	Coupler Cap
1	Large Wheel w/o Tire
1	Length String (user provided)
1	Tire

STEP 2> Place the hoist near the edge of a table and allow the tire and string to hang over the side.

STEP 3> Make sure R.O.S. is loaded, then turn ON the motor switch.

NOTE: To **STOP** the hoist at any time, type: **MOTR1,0 <return>** or **MOTRO <return>** or turn the **ON/OFF** switch to **OFF.**

STEP 4> Type the following commands in the direct mode.

MOTR1,2 <return>
The hoist should raise the tire. If the hoist is lowering the tire, just reverse the 2 motor wires.

Now type:
MOTR1,0 <return>
The hoist will stop.

Now try:
MOTR1,-2 <return>
The hoist should lower the tire.

Type:
FLIP1 <return>
The motor should reverse and raise the tire.

Now type in:
FLIP1 <return>
The motor should reverse again.

STEP 5> Now that everything is working, try using different motor speeds such as: **MOTR1,6 <return>** and **MOTOR1,-6 <return>.** Notice that a motor speed of **6** is slower than **-6.** This is because the weight of the tire is working against the motor as you try to raise it and working with the motor as you lower it. This is called **LOAD** and will be present in most motorized systems. As you design your own projects, you must always consider the **LOAD** on the motor and adjust your speeds accordingly. Normally, the best way to find the optimum speed is by trial and error. Pick a speed that you think might work, and try it. Then change it, if it's not what you want.

In this project we've used two (2) **speed reduction** capsules, and in the previous project we used a **worm gear** capsule. In both cases we increased the **TORQUE,** or turning force, by reducing the speed of the motor. You will learn more about how gears affect torque and speed in future projects.

PROJECT #5

HOIST WITH PROGRAM

DESCRIPTION:
In project #4 the hoist was controlled by sending MOTR commands in 'direct mode'. In this project, you'll control the same HOIST by sending commands in 'program mode'.

INSTRUCTIONS:

STEP 1> Use the same model and motor connections from project #4, and make sure R.O.S. is loaded.

STEP 2> Load in the program HOIST by typing: **LOAD"HOIST",8 <return> then RUN <return>.**

STEP 3> Experiment with the Hoist using objects of various weights. The CURSOR **UP/DOWN** key controls the direction of the hoist. If the direction is reversed, then press the **'R'** key to correct it, or simply reverse the motor leads. To change speed use the **'F'** key for faster, the **'S'** key for slower, and **SPACE** to stop.

STEP 4> Use the **RUN/STOP** key to stop the program and then type LIST as you did in Project #3. See if you can follow the program logic.

It should have been much easier to control the hoist with this project than the previous one. The computer program waits for us to press a specific key and then responds with lightning speed. Computers are capable of performing thousands of operations per second and that's what makes them so useful. It's not that they're smart. In fact, a computer can't do anything, unless it's told what to do and how to do it by a programmer. But, because they can do some jobs so fast and so precise, they have become indispensible 'tools' in today's world.

PROJECT #6

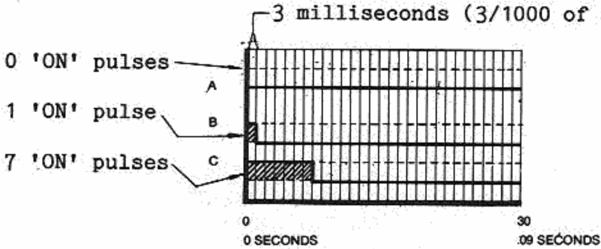
MORE ABOUT MOTOR SPEED
(PULSE WIDTH MODULATION)

DESCRIPTION: Some robotic kits simply allow control of a motor by turning it either ON or OFF, but in most robotic systems, it is also important to control the speed. We use a technique called PULSE WIDTH MODULATION to adjust motor speed. We discussed it briefly in Project #1.

With this technique, the motor is turned ON and OFF very rapidly to vary the speed.

Unless you change the settings, the speed will range from <u>full off</u> to <u>maximum</u> in 30 steps. This can be changed, however by using RANGE, or a sophisticated version of the MOTR command for very precise and exacting work.

For more information on these advanced commands, see the section on ADVANCED PROGRAMMING.

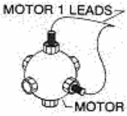


MOTOR SPEED CHART
In 3 millisecond intervals (3 m.s.)

The Motor Speed Chart shows 30 equally spaced intervals. Each interval is 3 milliseconds long (0.003 sec). This is the default value and may be changed. During each interval the computer can send either an **ON** or an **OFF** pulse to the motor. The first waveform **(A)** shows the motor turned off, since all pulses are **OFF**. The **B** waveform shows the motor turned **ON** for 1 pulse and **OFF** for 29 pulses. This is the speed = **1** given by **MOTR#,1.** Waveform **C** shows a speed of **7,** given by **MOTR#,7.** This is 7 pulses **ON** and 23 pulses **OFF.** The other speeds between 0 and 30 work exactly the same way.

INSTRUCTIONS:

STEP 1> To demonstrate this technique, connect a motor to motor leads #1.



STEP 2> Turn the **ON/OFF** switch to **ON** and make sure R.O.S is loaded.

STEP 3> Now load the program PWM by typing: **LOAD"PWM",8 <return> then RUN.**

Notice the L.E.D's (Light Emitting Diodes) on the Interface Unit.

These are the 8 red lights numbered 0 thru 7. LED 0 shows what type of pulse is being sent to the motor. When LED 0 is on, an ON pulse is being sent. When LED 0 is off, an OFF pulse is being sent. The program has started the motor at speed = 1 and you can see that LED 0 is only ON for about 1 pulse in 30.

The computer is now waiting for you to select a new speed. Enter any speed between 0 and 30 on the keyboard and hit <return>. Notice how LED 0 reflects the change in ON versus OFF pulses as the motor speed up or slows down. This is PULSE WIDTH MODULATION in action.

NOTE: When using the ADVANCED commands, the ON/OFF times can be changed to virtually any ratio and any time period that the user wishes!

PROJECT #7

INCREASING TORQUE WITH GEARS

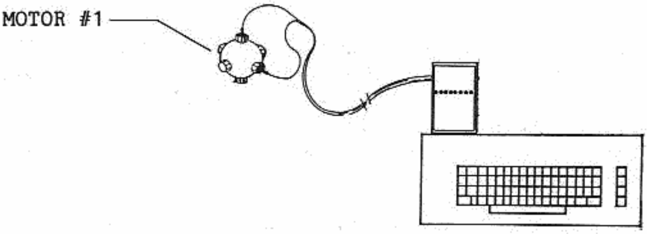
DESCRIPTION:
When building some of your own models, you will probably notice that the motor does not have enough "power" to do the job. Many times the power is adequate, but the **TORQUE** is too low. Torque is actually a turning force.

If we can gear down the motor, the turning force or torque will be increased.

INSTRUCTIONS:

Try a simple experiment:

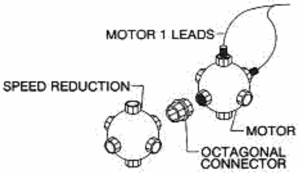
STEP 1> Assemble the project shown below.



STEP 2> Make sure R.O.S. is loaded and give it the simple command: **MOTR 1,10 <return>.**

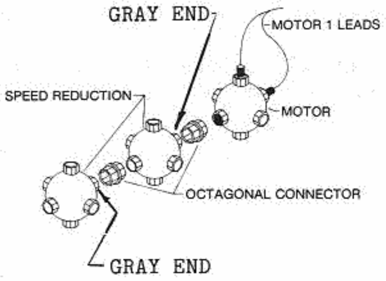
Now that the motor is running, place your finger on the motor shaft and notice how little pressure it takes to stop the motor.

STEP 3> Connect a speed reduction capsule to the motor:



With the motor still going at a speed of 10, try to stop the gear from spinning. Notice that it is more difficult to stop. Decreasing the speed has increased the torque.

STEP 4> Add one more speed reduction capsule.



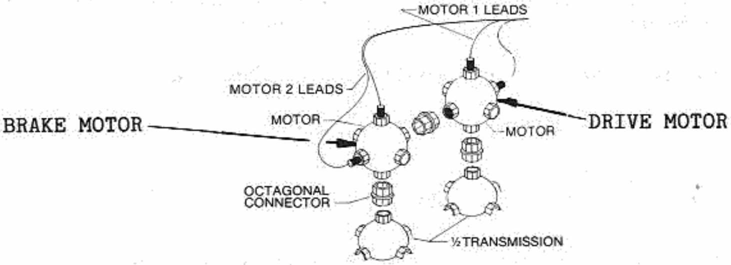
It is now very difficult to stop the motor! The torque is very high.

Use this process when you need more torque in your models.

There is a more scientific way to demonstrate how gears affect torque.

By connecting two motors together and by using the second motor as a **BRAKE**, it will be possible to estimate increases in torque.

STEP 5> Assemble the project shown below:



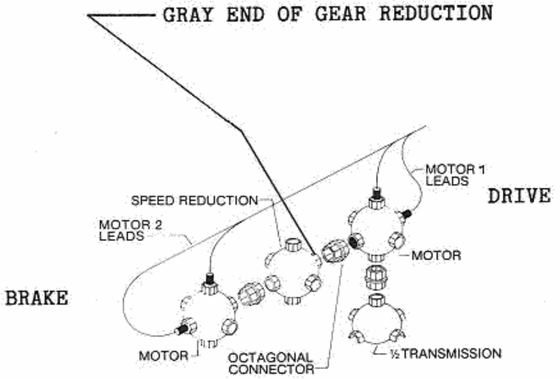
STEP 6> Start Motor #1 (drive motor) by typing: **MOTR1,1 <return>.**

Now turn on Motor #2 (brake motor) by typing: **MOTR2,-1 <return>.**

The motors should stop or almost stop. If they don't, reverse the leads to motor #2.

With the motors connected directly together and with neither having the advantage of a speed reduction capsule, the brake motor is able to stop the drive motor by applying equal torque in the opposite direction.

STEP 7> TYPE: **MOTRO** (all motors off) and insert a speed reduction capsule as shown below:



STEP 8> Type: **MOTR1,1 <return>.** The drive motor should spin very slowly now.

Turn on the brake motor by typing: **MOTR2,-1 <return>.** The motors don't stop as they did before because the drive motor now has a mechanical advantage over the brake. Type: **MOTR2,-10 <return>.** They still don't stop. Give the brake motor full power by typing: **MOTR2,-30 <return>.** It still can't stop the drive motor.

A simplified mathematical explanation for changing torque is as follows:

P = Power
T = Torque
S = Rotational Speed (RPM)

$T = \frac{P}{S}$ TORQUE equals Power divided by Speed.

If the speed is cut in half, then the torque is doubled as long as we give the motor the same amount of power. The more the motor is 'geared down', the more torque we get, but we sacrifice speed. Both the **SPEED REDUCTION** capsule and the **WORM GEAR** capsule use an arrangement of gears to reduce the speed of the drive shaft. Mechanical devices such as these are commonly called **GEAR BOXES** or **TRANSMISSIONS.** The term **GEAR RATIO** is used to define the ratio of input speed to output speed of a **GEAR BOX.**

For example, a 10:1 (10 to 1) **GEAR RATIO** means that the input shaft must make 10 revolutions to produce 1 revolution of the output shaft. This also means that the output speed is 10 times slower, but the output torque is 10 times greater.

The two gear capsules in the ROBOTIC WORKSHOP have the following GEAR RATIOS:

SPEED REDUCTION CAPSULE	23:1
WORM GEAR CAPSUL	50:1

Gear reduction systems are in use all around us to change the speed and torque of engines and motors. Your car's transmission is a system that can be 'shifted' into different gears. Low gear is actually a high ratio, that gives high torque to get the car moving. High gear is a lower ratio, that produces more speed but less torque. Once the car is moving, less torque is required to keep it moving, unless we try to climb a hill.

PROJECT #8

BASIC MOTORS

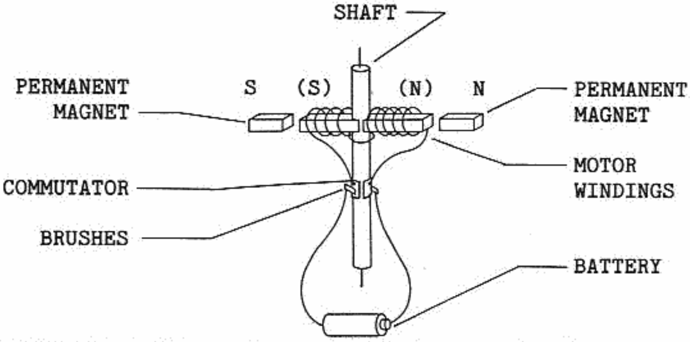
DESCRIPTION:
The purpose of this project is to demonstrate, in simple terms, how the small D.C. motors work.

All motors work on the principles of magnetism. There are basically two types of magnets. The first type is called a **PERMANENT MAGNET** and can be found in many shapes and sizes. The small objects that stick to your refrigerator door are examples of permanent magnets. The second type is called an **ELECTROMAGNET** and is created when electrical current passes through a coil of wire. When the current is turned off, the **magnetic field** disappears and the device ceases to be a magnet.

Both types of magnets have an interesting property called **POLARITY.** One end of the magnet is the **North** pole and the other end is the **South** pole. When two magnets are placed near each other, their **magnetic fields** interact to create certain forces. If you try to touch the **North** pole of one magnet to the **North** pole of the other, the magnetic forces will keep them apart. Whereas, if you touch the **North** pole on one to the **South** pole on the other, the forces will keep them together. The rule to remember is, "Opposite Poles Attract".

The D.C. (direct current) motors in the WORKSHOP use both permanent and electro-magnets to create the forces that make them spin.

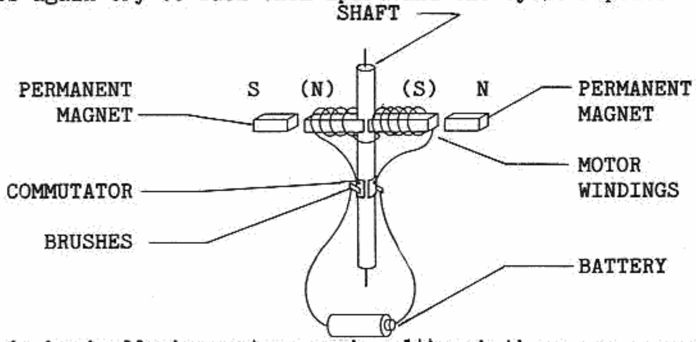
Refer to this simplified diagram of a motor.



The **MOTOR WINDINGS** produce an electromagnet when current passes through them. The polarity (North or South Poles) is determined by the direction that current is flowing in the wires. The **BRUSHES** and the **COMMUTATOR** pass the electrical current from the battery into the motor windings. Their purpose is to constantly switch the direction of current flow in the motor windings and by so doing, change the polarity of the electromagnet.

Follow the sequence as you refer to the previous diagram.

As current begins to flow in the motor windings, an electromagnet is created. Assume that the right side of the winding is the north pole and the left side is the south pole. In the diagram you will notice that the north poles of both the electromagnet and permanent magnet are adjacent to each other as are the south poles of both magnets. The forces will try to push them apart because the poles are the same. Since the permanent magnets are mounted in the motor case and cannot move, the electromagnet mounted to the shaft will rotate. The forces are turning the shaft (with the electromagnet attached) into a position where the poles are opposite. Just as the opposite poles get close to alignment (see figure below), the commutator changes the direction of current flow. Suddenly the poles of the electromagnet reverse and we're back where we were in the 1st figure, north pole to north pole and south pole to south pole. The forces again try to push them apart and the cycle repeats.



This is basically how motors work, although there are many variations for both D.C. and A.C. (alternating current) motors. The actual motors used in the WORKSHOP have three sets of windings. More windings provide better starting torque, smoother rotation, and insure that the motor runs in a specific direction for a given battery polarity.

INSTRUCTIONS: Now try this experiment with a motor.

STEP 1> Make sure R.O.S. is loaded.

STEP 2> Turn the ON/OFF switch to OFF. Type: **MOTR1,30 <return>**

With the motor switch OFF, the motor will not spin. Notice that LED 0 is on.

Now type: **MOTR1,-30 <return>**

Now LED 1 is on.

The first 6 LED's from right to left indicate what voltage is available on the 6 motor leads. The LED's will light up even if the motor switch is OFF and no motor is connected. LED's 0 and 1 are for Motor #1, LED's 2 and 3 are for Motor #2, and LED's 4 and 5 are for Motor #3. When the LED is ON, it means that a positive voltage exists on the corresponding motor lead. Experiment by sending different speed and direction commands to all three motors. Use both the MOTR and FLIP commands.

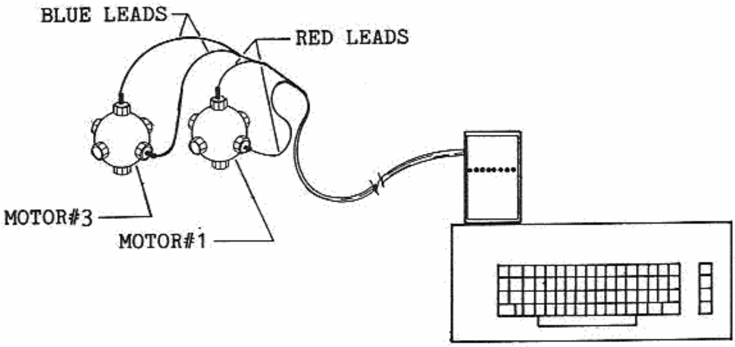
PROJECT #9

PROGRAMMING MORE THAN ONE MOTOR

DESCRIPTION:
The B100 interface and R.O.S. are capable of controlling up to 3 motors at the same time. Each motor may be going a different speed and/or direction. In other words, R.O.S. provides independent motor control.

The ROBOTIC WORKSHOP contains 2 motors, and the WORKSHOP PLUS contains 3 motors. You can also order extra motors and other parts if you desire.

INSTRUCTIONS:
Try this experiment with 2 motors.



STEP 1> Make sure R.O.S. is loaded.

STEP 2> Turn the ON/OFF switch to ON.

STEP 3> Now turn on motor #1 at a speed of 10 by typing: **MOTR1,10 <return>**.

Motor #1 should be spinning while motor #3 is off. Notice that LED 1 is blinking indicating the speed of motor #1.

STEP 4> Turn on motor #3 by typing: **MOTR3,1 <return>**.

Motor #3 is spinning slowly while motor #1 is unchanged. (Again look at the LED's.)

STEP 5> Type: **MOTR3,-30 <return>**

Motor #3 is going full speed in <u>reverse</u> while motor #1 is still unchanged.

Experiment with different combinations of motor speeds and directions for both motors. Watch the LED's. Learning to read the LED's will help you 'DEBUG' your own projects.

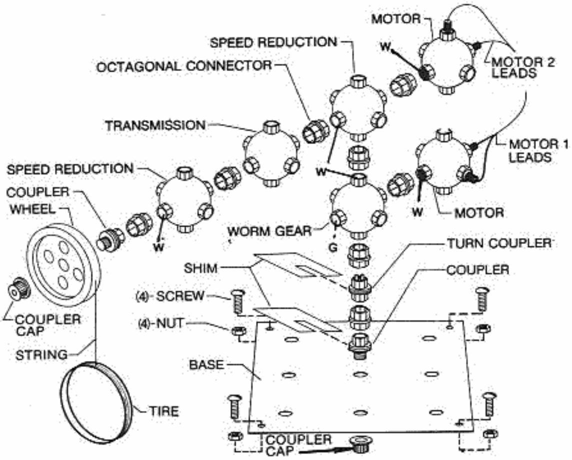
PROJECT #10

CRANE

DESCRIPTION:
This project is designed to demonstrate the use of two motors in the same project. Each will operate independently of the other.

INSTRUCTIONS:

STEP 1> Assemble the project shown below.



Parts required:

1	Base with Legs Attached
2	Coupler Caps
2	Shims
2	Coupler
8	Octagonal Connectors
1	Turn Coupler
1	Worm Gear
2	Motors
2	Speed Reduction
1	Transmission
1	Wheel
1	Tire
1	Length of String (user provided)

STEP 2> Insert a joystick into port #2 (rear port) of your computer.

STEP 3> Make sure R.O.S. is loaded and then type: **LOAD"CRANE",8 <return> then RUN <return>**

STEP 4> Switch the **ON/OFF** switch to **ON.**

The joystick should control the crane as follows:

STICK	CRANE
Left	Left
Right	Right
Up	Raise Tire
Down	Lower Tire

If the crane does not move left or right correctly you may reverse the direction by hitting the "L" key.

If the up/down direction is reversed, press the "U" key.

Experiment with the controls for a while, then stop the program using the **RUN/STOP** key.

Now type: **List -1000** (this will list the program up thru line 1000).

Notice line 400.

```
400 RANGE1,50
```

This is called the **RANGE** command and sets the 'range' of the motor speed. The normal value is 30. That's why, up till now, we've used MOTR1,30 to get full speed. In this project we've extended the range to 50 and so MOTR1,50 would be needed to get full speed. Note the number '1' after the **RANGE** command. That's the motor number (1, 2 or 3). Each motor can have its own range. You'll learn more about the **RANGE** command later.

PROJECT #11

BELT DRIVEN "GIZMO"

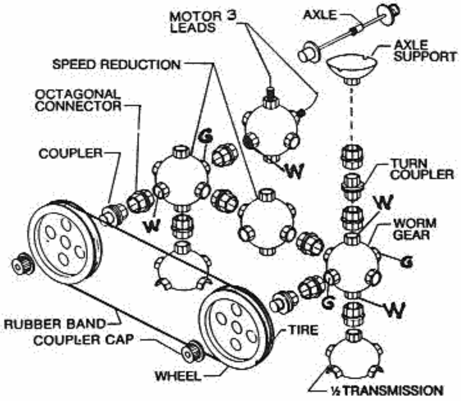
DESCRIPTION:
Belts are very useful devices for transferring power. This project will use a belt to drive a worm gear. You need a rubber band to be used as the belt.

INSTRUCTIONS:

STEP 1> Assemble the project as shown below:

Parts required:

2	Coupler Caps
2	Couplers
2	Wheels With Tires
1	Worm Gear
2	Speed Reduction
1	Motor
1	Transmission
9	Octagonal Connectors
1	Rubber Band (user provided)
1	Axle Support
1	Axles
1	Turn Coupler



STEP 2> With R.O.S. in place, type **LOAD"BELT",8 <return> then RUN <return>.**

Notice that all the drive power to the axle comes from the belt.

PROJECT #12

GYROSCOPES

DESCRIPTION:
A gyroscope is a device designed to maintain stability of ships, spacecraft, missles and other objects. Once set in motion, a gyroscope will try to maintain its axis of rotation. A child's spinning 'TOP' is an example of a simple gyroscope.

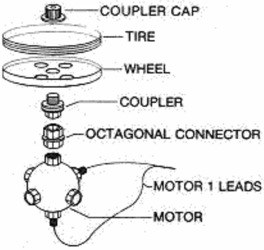
The gyroscopic effect is created whenever a mass is spinning. The degree of stability is determined by three things, rotational speed, mass, and distance of the mass from the axis of rotation. As in the example of the childs 'TOP', a gyroscope can be moved easily in 3 directions: forward or back, sideways, or up and down. But a gyroscope will resist being rotated. That is, it will take considerable effort to move the axis of rotation out of a position which is parallel to its present position.

INSTRUCTIONS:

Step 1> Assemble the project shown below.

Parts required:

1	Wheel with Tire
1	Octagonal Connector
1	Coupler
1	Coupler Cap
1	Motor



STEP 2> Hold the motor with the wheel up (as in the diagram), being careful not to touch the wheel, Now quickly rotate your wrist so that the wheel is down.

There was nothing hard about that. Now try it with the wheel spinning.

STEP 3> Type **MOTR1,30 <return>**

Hold the motor upright, as in Step 2, being careful not to touch the wheel. After the motor is going full speed, rotate your wrist as you did before. Can you feel how the spinning wheel resists being rotated? Move it side ways and up and down without rotating it. Notice how it doesn't resist motion in these directions. This is called translation. A gyroscope doesn't resist being translated, only rotated.

STEP 4> Stop the motor by typing: **MOTR1, 0 <return>.** Now remove the tire and repeat Step 3 without it.

This time the gyroscope is easier to rotate because it has less mass.

A gyroscope is based on a scientific principle called MOMENTUM. Simply stated, it says that an object in motion tends to remain in motion. It also says that the faster the object is traveling and the more mass it has (the heavier it is), the harder it will be to make it stop or change directions.

When the wheel spins, all the mass moves in a circular path or 'obit' around the motor shaft and develops a certain amount of ANGULAR (ROTATIONAL) MOMENTUM.

As you try to change the axis and thus change the direction of the orbiting mass, you must apply force to overcome the ANGULAR MOMENTUM.

The equation for momentum is:

$$\text{MOMENTUM} = \text{MASS} * \text{VELOCITY}$$

Notice that when the velocity equals zero, so does the momentum.

PROJECT #13

MOTOR EFFICIENCY TEST

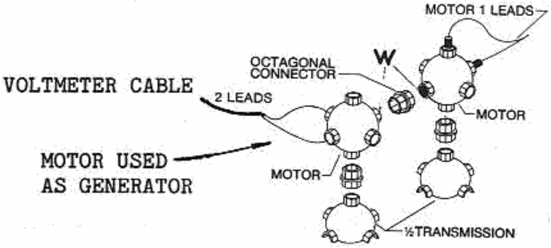
DESCRIPTION:
Electric motors are a very important part of our daily lives. They are in use all around us for many different purposes.

An electric motor converts electrical energy into mechanical energy. The process isn't 100% efficient though, which means that some energy is lost in the process. Most losses, especially with our small D.C. motors, are in the form of heat caused by friction and by heating in the wires. That is why all motors get hot, or at least warm, when they are running. Motors are rated as to their MOTOR EFFICIENCY and ratings vary from about 40% to 95%.

In this project you'll see how motor efficiency can be measured.

INSTRUCTIONS:

STEP 1> Assemble the project shown below: (Do not connect the generator (second motor) yet.



Parts required:

1 Transmission Module
2 Motors
3 Octagonal Connectors

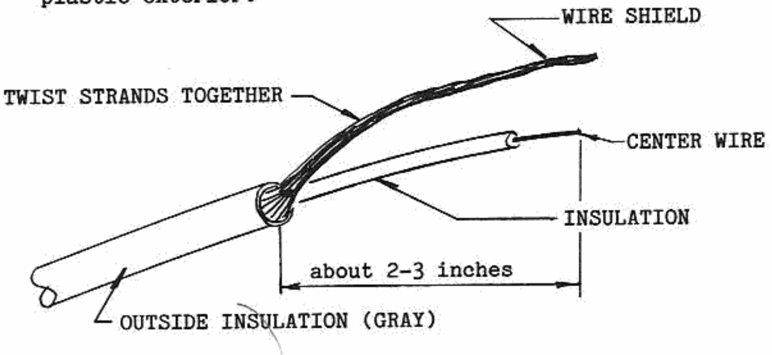
Efficiency is calculated as follows:

$$\% \text{ EFF} = \frac{\text{ENERGY OUT}}{\text{ENERGY IN}} * 100$$

It is difficult to measure the mechanical energy produced by a motor, and because of this, it is also difficult to get an accurate measure of efficiency. In this demonstration, we will sacrifice accuracy for ease of measurement. We will convert the mechanical energy back into electrical energy so we can measure it from our B100 interface. This is done using a second motor as a GENERATOR. A GENERATOR is a device that does the opposite of what a motor does. It converts mechanical energy into electrical energy. Your car has a generator (usually called an alternator), which produces electricity to keep the battery charged. Not all motors can act as generators, but the WORKSHOP motors can.

You've assembled tne project which has two motors coupled directly together, and connected the 1st motor to the red motor leads.

STEP 2> Find the 3 foot long gray cable in the workshop with a phono connector on one end. If the other end is not stripped, use a knife to carefully remove about 2 inches of the outside (gray) plastic insulation. This is called COAXIAL cable. It consists of a center wire surrounded by insulation, then a wire shield, and finally by a gray plastic exterior.



STEP 3> Twist the strands of shield wires together and then remove about 1/2 inch of the insulation from the center wire. Be careful not to cut the center wire. This is the cable you will use for the VOLTMETER and MULTISCOPE projects later in the manual. In this project we will need to use part of the VOLTMETER.

STEP 4> Plug the phono end of the cable into the IN (J2) jack on the back of the interface. Plug the leads that you've prepared in Steps 2-3 into the generator (2nd motor). Move the Selector switch into the METER/SCOPE position.

STEP 5> Make sure R.O.S. is load and that the ON/OFF switch is on.

Now type: **LOAD"EFFICIENCY",8 <return> then RUN <return>.**

The program first brings the motor up to speed and then determines if the voltage from the generator is of the correct polarity. If not, it reverses the drive motor. It then takes 10 voltage readings and averages them. Finally, it approximates the combined efficiency of the motor and generator by the formula:

$$\text{(Combined) \% EFF} = \frac{\text{OUTPUT VOLTAGE}}{\text{INPUT VOLTAGE}} * 100$$

(In this case, voltage is a good approximation of energy)

We were forced to make several assumptions and approximations in this demonstration in order to obtain a answer for motor efficiency. This should leave you quite skeptical about the accuracy of the results, and rightly so. However, now we have a good method of measuring the relative efficiencies of a group of motors (if we had them).

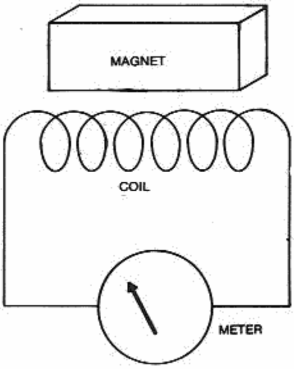
NOTE: This project is also a good test for the batteries in the B100 interface. Run the program and note the efficiency. Then reverse the voltmeter leads and run the program again. If the efficiency is more than 25% different, the batteries probably need to be replaced.

PROJECT #14

GENERATORS

DESCRIPTION:
In the last project, we learned that a generator is just the opposite of a motor. It converts mechanical energy into electrical energy.

Remember that an electromagnet is created as current flows in a coil of wire, and it is the magnetic forces that cause the motor to spin. When the process is reversed, an electrical force is created. By spinning the shaft, the coils of wire pass through the magnetic field of the permanent magnets and induce an electro-motive force (EMF) in the wires. This EMF is the VOLTAGE that we discussed in project #1



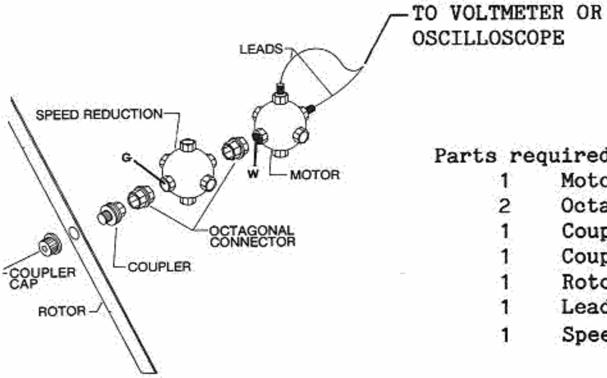
AN ELECTRO-MOTIVE FORCE (EMF) IS CREATED

There are three characteristics of a generator which affect the amount of voltage produced. Spinning the shaft faster will produce higher voltage. More turns of wire or a stronger magnetic field will also produce higher voltage.

In this project, we'll use the VOLTMETER and OSCILLOSCOPE to measure the voltage created by a small generator.

INSTRUCTIONS:

STEP 1> Assemble the project shown below: Notice that you'll be using one of the motors as the generator.



Parts required:

1	Motor
2	Octagonal Connectors
1	Coupler
1	Coupler Cap
1	Rotor
1	Leads to Voltmeter/Scope
1	Speed Reduction

STEP 2> Use the voltmeter cable from the last project. Insert the phono plug into the **IN** (J2) jack on the back of the B100 interface. Plug the two leads into the motor, as shown in the figure.

STEP 3> Now load the voltmeter by typing: **LOAD"MULTIVOLT",8 <return> then RUN <return>.**

STEP 4> Make sure the selector switch is in the METER/SCOPE position.

STEP 5> Now, spin the rotor with your hand. If the meter reads zero, turn the rotor the opposite direction or reverse the leads from the generator.

STEP 6> Spin the rotor at different speeds and watch how the voltage changes. See if you can create enough voltage to exceed the range of the meter.

STEP 7A> Turn your computer **OFF** and then **ON** to clear the multivolt screen.

STEP 7B> Now type: **LOAD"MULTISCOPE",8 <return> then RUN <return>.**

This loads the digital oscilloscope. With it, we can observe the waveform of the voltage produced by the generator.

STEP 8> When the MULTISCOPE screen appears, set the time/division to 1 sec by pressing the F1 key several times. Also set the volts/division to .5 by pressing the F4 key (shift and F3).

Now spin the generator as you did before and watch the screen. See if you can keep the voltage steady.

If you want to know more about the oscilloscope and how to use it, review the MULTISCOPE section of this manual.

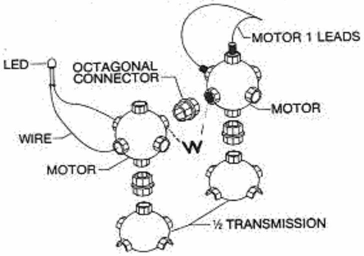
PROJECT #15

GENERATORS II

DESCRIPTION:
In this project we'll use a generator to light and L.E.D.

INSTRUCTIONS:

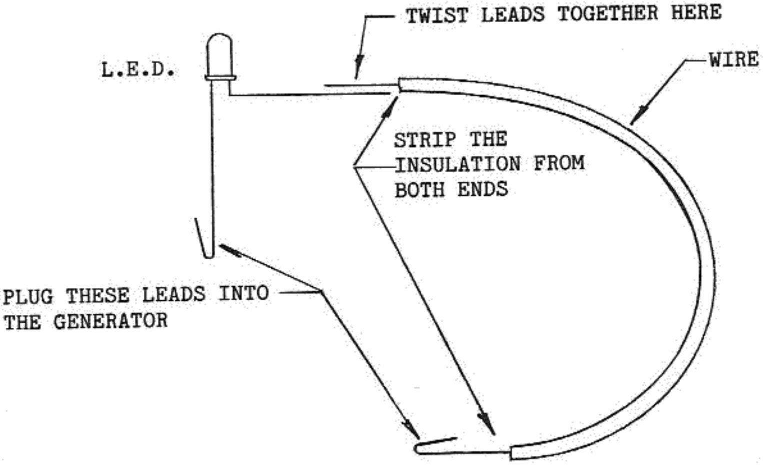
STEP 1> Assemble project shown below



Parts required:

2	Motors
3	Octagonal Connectors
1	Transmission Module
1	Lead for Voltmeter/Scope
1	Light Emitting Diode (LED)
1	Piece of Wire (to hook up LED)

STEP 2> Refer to the following sketch and bend the leads of the LED and the wires.



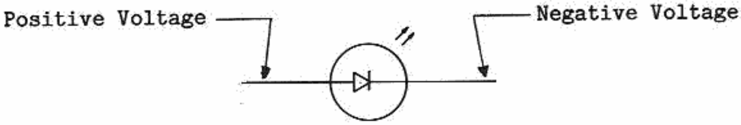
STEP 3> Plug one end of the LED and one end of the wire into the generator as shown in the assembly figure.

STEP 4> Make sure R.O.S. is loaded and that the ON/OFF switch is on, then type: **MOTR1,30 <return>.**

The motor should be running at full speed and the LED should be on. If the LED is NOT on, type: **MOTR1,-30 <return>.**

The motor to which the LED is connected is actually generating a voltage which is lighting the LED. If you slow down the motor, the LED will become dimmer and will go out completely when the voltage reaches about 1.5 V. It takes about 1.5 volts to get the LED to turn on.

The electrical symbol for an LED is show below.



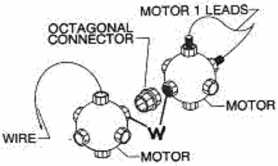
PROJECT #16

EFFECTS OF LOADING A GENERATOR

DESCRIPTION:
Now that you have gained a basic understanding of how generators produce voltage, examine what happens when we take power (energy) from the generator.

INSTRUCTIONS:

STEP 1> Assemble the project shown below:



Parts required:

2	Motors
1	Octagonal Connector
1	Piece of Wire

STEP 2> Make sure R.O.S. is loaded and turn the ON/OFF switch to ON.

Type: **MOTR1,30 <return>.** The motor should come up to speed. Pay attention to the speed of the motor.

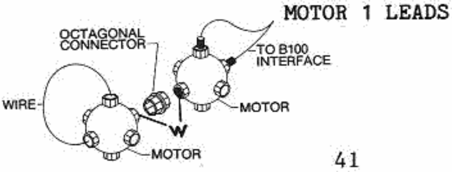
Now uncouple the generator. The motor should spin considerably faster.

Recouple the generator and notice that the motor slows down again.

When the motor was spinning free, it was in what we call a **"NO LOAD"** condition. At this point, it was spinning at its maximum RPM. When it was connected to the generator, a **LOAD** was applied to the motor, mostly in the form of friction, causing it to slow down. There is still no load on the generator, however.

STEP 3> Now take the wire used in the previous project and **"short"** the output of the generator as shown below.

The motor should slow down drastically.



By shorting the output we have produced a large current 'LOAD' on the generator. Since the generator must get its power from the drive motor, the motor also feels the drain (or LOAD) and slows down because of it.

Electrical generators are even more important to our way of life than electric motors. Nearly all the electricity used in the world is produced by large electrical generators. They can obtain their mechanical driving power in many different ways. Some of the most common sources are:

Coal Fired Steam Turbine
Oil Fire Steam Turbine
Nuclear Powered Steam Turbine
Hydro-Electric (Falling Water)

If you have ever experienced a power black out, you know how difficult it is to get along without electricity.

PROJECT #17

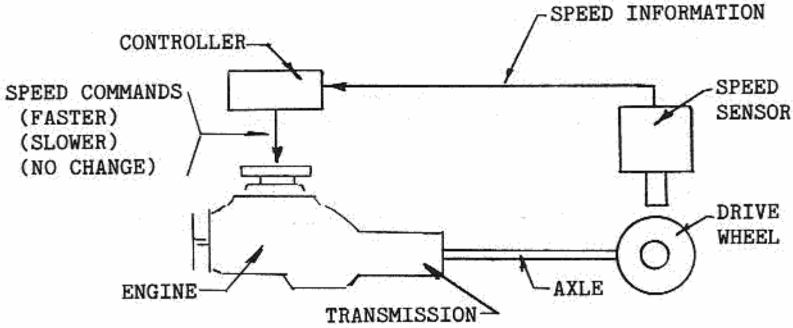
FEEDBACK

DESCRIPTION:
Feedback is a very important principle in electronic and mechanical systems.

This simple example will demonstrate the concept. Suppose you are driving a car and want to hold the speed constant at 50 mph. By watching the speedometer and by pressing or releasing the accelerator, you should be able to stay close to 50 mph. The speedometer is feeding information back to you, so that you can adjust the amount of fuel to the engine. If the speed drops below 50, you'll give it more gas. If the speed goes above 50, you'll give it less gas. This is the concept of feedback. Without it, you have no way of knowing what, if any, corrections to make. Remove the feedback loop (the speedometer), and you'd have a tough time holding 50 mph.

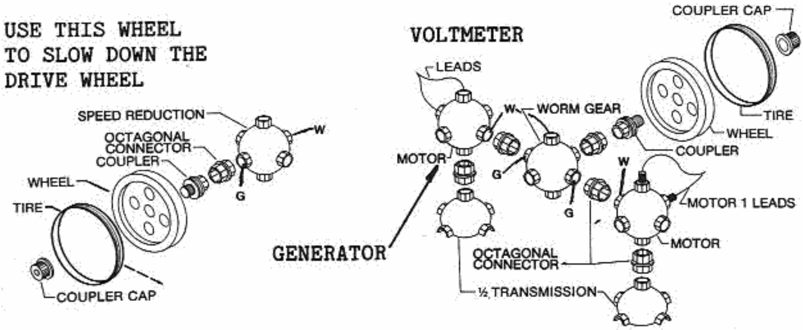
An automatic cruise control works the same way. It adjusts the car's speed based on feedback.

The following diagram illustrates the feedback concept.



INSTRUCTIONS: Build a simple project to demonstrate this process.

STEP 1> Assemble the project shown below:



STEP 2> Using the voltmeter cable, insert the phono plug into the IN (J2) jack on the back of the interface. Plug the two cable leads into the generator as shown in the assembly figure. Make sure R.O.S. is loaded and that the ON/OFF switch is in the ON position. Also move the selector switch to the METER/SCOPE position.

STEP 3> Type: **LOAD"FEEDBACK",8 <return> then RUN <return>.**

STEP 4> After a few seconds, the screen will display voltage and speed. The voltage is a measurement of how fast the wheel is actually spinning. The speed is a measurement of how fast we are telling the motor to go or how much power we give the motor.

We are trying to keep the actual speed (voltage) at 20. Notice that the voltage varies slightly above and below 20. Also notice that the speed varies a few digits.

Now hold the wheel that is attached to the speed reduction capsule next to the spinning wheel. By holding this wheel, we exert a load on the motor and try to slow it down. Keep your eye on the voltage and speed columns. Voltage will drop and then increase, and speed will increase. By providing feedback and sensing that the motor was going too slow, the computer increased the speed of the motor.

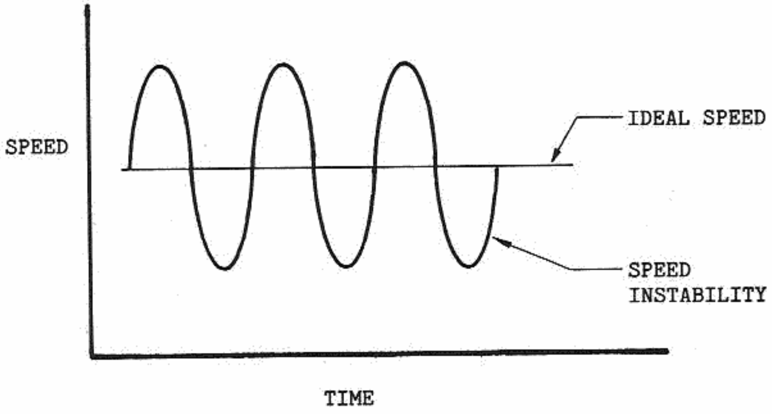
Now quickly remove the wheel. The voltage increases because of a lighter load. The speed will now be reduced automatically.

Program lines 1000 to 2000 contain the program logic (algorithm) which accomplishes this feedback "magic".

PROJECT #18

FEEDBACK STABILITY

Feedback is very important to electrical and mechanical systems, as shown in project #17. It must be used correctly, however, or the process can become 'unstable'. Several factors affect stability, including the accuracy of the measuring and control devices and the response time of the system. Consider the example of the automobile from the previous project. Suppose there is a 20 second delay between the time the car changes speed and the time the new reading is displayed. You would tend to 'overrun' the target speed in both directions. By the time the speedometer reads 50 mph, you're no longer traveling at that speed. Look at the diagram below.

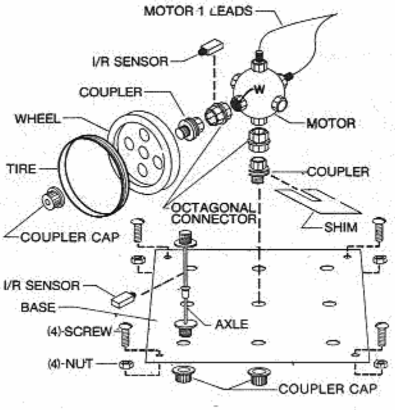


The speed oscillates, rather than being constant. Feedback must be received as soon as possible and must be as accurate as possible (in most cases) to be effective. The corrective response is also important and must not be too large or too small. Stable systems are sometimes difficult to achieve.

INSTRUCTIONS:
Try a project using the I/R sensors to provide the feedback. Refer to project #30 if you need help with the I/R sensors.

Under bright conditions, it may be necessary to cover the sides and back of the RECEIVER to keep extraneous light from affecting the sensor readings.

STEP 1> Assemble the project shown below.



Parts required:

1	Axle
3	Coupler Caps
1	Shim
2	Couplers
2	Octagonal Connectors
1	Motor
1	Wheel With Tire
1	Base
4	Bolts With Nuts
2	I/R Sensors

NOTE: **SENSOR MOUNTING**

We have provided a small strip of double faced tape in the WORKSHOP for mounting the I/R sensors. One side of the tape is very sticky and should be attached to the white plastic sockets which hold the sensors. The other side is not as sticky and is designed for temporary attachment to walls, tables, motor capsules, etc. It can be attached and removed several times.

Cut small pieces from the tape strip and use them until they will no longer adhere properly. If you are careful with the mounting tape, it will last a long time.

STEP 2> Mount the I/R sensors so that the beam will pass through the holes in the wheel and so that no beam will be present when the wheel is rotated. Use **ALIGN 8,7** to verify sensor alignment before proceeding. Make sure the SELECTOR switch is in the SENSOR position.

STEP 3> Make sure R.O.S. is loaded, then type: **LOAD"STABLE FB",8 <return> then RUN <return>.**

First select the 'S' option. After about one minute, notice that the speed (measured by the I/R sensors) doesn't change very much. This is a stable feedback circuit.

Place your hand in front of the I/R sensor and note that the speed given to the motor goes to 30 and the wheel spins very fast. Remove your hand and the system will immedately slow down.

Press any key and select the 'U' option. After about one minute, notice how the speed varies over a wide range. This is an unstable feedback system.

PROJECT #19

HOW JOYSTICKS WORK

DESCRIPTION:
This project will explain how a joystick works and how it can function as an INPUT DEVICE.

Joysticks work differently on different computers. That is why a COMMODORE joystick will not work on an APPLE computer. In this project we will demonstrate the COMMODORE stick, which is also compatible with ATARI and other computers.

The joystick is simply a group of 5 switches in a fancy case with a stick and a button. These 5 small switches are arranged in such a way that moving the stick or pressing the button will close one or more of the switches.

A wire from each is connected to the computer, and the computer can read each line to determine the position of the switch (open or closed). R.O.S. contains a special command called **'JOY'** to read the switches and place their status in a variable. The program can then test the variable and act accordingly.

The following values will be placed in the variable for the corresponding switch closure:

JOYSTICK ACTION	VARIABLE VALUE
UP SWITCH CLOSED	1
DOWN SWITCH CLOSED	2
LEFT SWITCH CLOSED	4
RIGHT SWITCH CLOSED	8
BUTTON SWITCH CLOSED	16

If more than one switch is closed at a time, then the values add together. For example, an UP/LEFT/BUTTON at the same time would leave the value of 1 + 4 + 16 = 21 in the variable.

There are 9 possible stick positions on a normal joystick. Each will correspond to a different reading. And if the button is pushed, each of the 9 values will have 16 added to it. This means that the joystick variable can have 1 of 18 possible values. The combinations are listed below:

STICK POSITION	VALUE WITHOUT BUTTON	VALUE WITH BUTTON
NO DIRECTION	0	16
UP	1	17
DOWN	2	18
LEFT	4	20
RIGHT	8	24
UP/LEFT	5 (1+4)	21
UP/RIGHT	9 (1+8)	25
DOWN/LEFT	6 (2+4)	22
DOWN/RIGHT	10 (2+8)	26

INSTRUCTIONS:

STEP 1> Plug a joystick into the rear joystick port.

STEP 2> Make sure R.O.S. is loaded.

STEP 3> Type the following: **LOAD"STICK",8 <return>** then **RUN <return>**.

This program will read the joystick and show you the values for various stick positions. See if you can find all 18 values listed above.

PROJECT #20

BINARY NUMBERS, THE LANGUAGE OF COMPUTERS

DESCRIPTION:
Most of today's home computers are 8 bit or 16 bit digital microcomputers. The memory of these computers is made up of thousands and thousands of electronic cells called **BITS.** The heart or **BRAIN** of the computer is called the **MICROPROCESSOR.** It **processes** data (information) by following instructions from a computer **PROGRAM.** By manipulating data in memory at unbelievable speeds, the processor is able to perform amazing tasks.

Information is stored in the memory in **BINARY** form. **BINARY** is the name of a numbering system which uses only 2 numbers, **ZERO** and **ONE.** This is different than the **DECIMAL** numbering system that we humans normally use. In **DECIMAL** there are 9 numbers allowed, 0 thru 9. Below is an example of numbers between 0 and 10, expressed in both **BINARY** and **DECIMAL** form.

BINARY	DECIMAL
0	0
01	1
10	2
11	3
100	4
101	5
110	6
111	7
1000	8
1001	9
1010	10

Notice that the BINARY numbers look like some sort of code. Since only 1's and 0's can be used, the numbers are very unfamiliar.

The computer, on the other hand, can only work with 1's and 0's. A single memory BIT can be either ON or OFF (1 or 0). When the microprocessor looks through memory, all it can see are thousands of 1's and 0's.

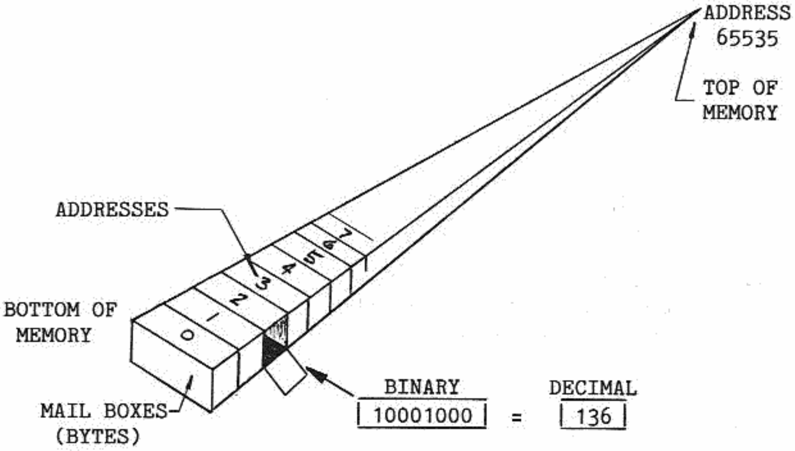
The 8 bit computers (the small ones) have processors that read **8 bits at a time.** This 8 bit group is called a **BYTE. BYTE** is the term which is used when specifying the memory of a microcomputer. The Commodore 64 for example has approximately 64K or 64000 **BYTES** of memory. This is equivalent to 512000 BITS (64000 x 8).

This project will help you become more familiar with BINARY numbers and, in particular, 8 bit binary numbers or **BYTES.**

Whenever you see or here the word **BYTE**, it will be helpful to make the following associations:

MEMORY ADDRESS OF THE BYTE
BINARY VALUE OF THE BYTE
DECIMAL VALUE OF THE BYTE

Each byte has an address or location in memory. Whenever the processor wishes to read the value of the byte, it has to reference that address. On 8 bit computers, addresses range from 0 (Bottom of Memory) to 65535 (Top of Memory). It may be helpful to think of memory as a long row of mail boxes, each with its own address. The processor can locate the correct mail box (by using the address), then open the mail box (byte) and see what's there (**read** a binary value). The processor could also place something new in the mailbox (**write** a binary value). If we humans want to see the contents of the mailbox, we may want to convert it to a decimal value which is more familiar to us.



INSTRUCTIONS:

STEP 1> Switch the ON/OFF switch to OFF and then turn the computer off and on again.

STEP 2> **DO NOT** load R.O.S.

STEP 3> Type: **LOAD"BINARY",8 <return> then RUN <return>.**

STEP 4> Place the selector switch in the SENSOR position.

The lights on the B100 Interface will display the binary value of any decimal number typed in, up to 255. Decimal 255 (binary 11111111) is the largest number that can be represented in 8 bits. The bits are numbered 0 thru 7 from right to left. Each bit is equivalent to a different decimal value and the total decimal value of the byte is obtained by adding up all the individual bit values. The chart below shows the decimal value of each bit.

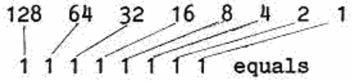
	BINARY	DECIMAL EQUIVALENT
Bit Number....	76543210	
	00000001	1
	00000010	2
	00000100	4
	00001000	8
	00010000	16
	00100000	32
	01000000	64
	10000000	128

All possible values of a byte (0-255) can be represented as a combination of one or more of the values of the above chart.

For example:

```
128  64    8   2
 |  /     /   /
 1 1 0 0 1 0 1 0  equals      10000000   =    128
                             +01000000   =     64
                             +00001000   =      8
                             +00000010   =      2
                              11001010   =    202
```

```
128  64   32   16   1
 |  /   /    /     /
 1 1 1 1 0 0 0 1  equals                      128
                                               64
                                               32
                                               16
                                                1
                                              241
```



```
 128
  64
  32
  16
   8
   4
   2
   1
 ---
 255
```

On a piece of paper, try to determine the decimal equivalents of various binary combinations. Type the decimal number into the computer and check the lights to verify the result. Start with the examples above. Type 202 and hit <return>. The lights on the B100 interface should correspond to the binary value 11001010. Keep experimenting until you understand how the process works. It is not important to be able to do the binary to decimal conversion in your head, and it is even more difficult to convert from decimal to binary. Understanding the process is what's important.

NOTE: If you would like to write your own programs to light the L.E.D.'s, add the following line to the top of your program.

```
POKE56579,255:POKE56577,0
```

Then **POKE** any value between 0 and 255 into address 56577 to set the lights (POKE 56577,VALUE) Remember **NOT** to run R.O.S.

PROJECT #21

BINARY ADDITION

DESCRIPTION:
As we have shown in the previous project on BINARY NUMBERS, the computer must perform all processing, including addition and subtraction, on binary numbers.

This project will show you how the computer adds binary numbers.

There are three simple rules to remember.

1 + 1	=	0, carry 1
0 + 1	=	1
0 + 0	=	0

Binary Addition Rules

Follow this example: Add decimal 2 + decimal 4

```
            00000010    =>  Decimal  2
          + 00000100    =>  Decimal +4
            00000110    =>  Decimal  6
```

That was too easy. Add decimal 5 and decimal 1.

```
     Carry............1.
            00000101    =>  Decimal  5
          + 00000001    =>  Decimal +1
            00000110    =>  Decimal  6
```

Add decimal 7 and decimal 3

```
     Carry...........11.
            00000111    =>  Decimal  7
          + 00000011    =>  Decimal +3
            00001010    =>  Decimal 10
```

On your own add 5 and 3. Don't forget the carries.

INSTRUCTIONS:
To watch this process in action, load the program BINARY ADD. Do not load R.O.S. at this time. Instead type: **LOAD"BINARY ADD",8 <return> and RUN <return>.**

Follow the screen prompts and watch the 8 LED's on the B100 interface. They will show the first number, then the second number, and finally the sum. Check out the LED's and make sure they are correct.

PROJECT #22

READING THE JOYSTICK

DESCRIPTION:
The joystick is read by the computer as a single byte. As we learned in project #19, the joystick consists of 5 individual switches. These 5 switches are connected to the lower 5 BITS of the joystick byte (Bits 0 thru 4). Any switch which is closed will show up as a '1' in the corresponding bit. The figure below illustrates the joystick byte.

BIT NUMBER	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
NOT USED	X	X	X	FIRE BUTTON	RIGHT	LEFT	DOWN	UP
BIT VALUE	128	64	32	16	8	4	2	1

JOYSTICK BYTE CHART

If the joystick is not moved, all the bits will be zero.
binary 00000000 = decimal 0

If the fire button is pushed, bit 4 will change to a '1'.
binary 00010000 = decimal 16

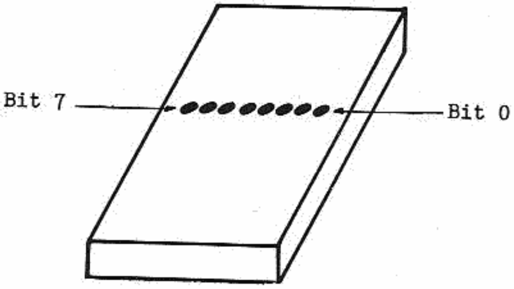
If the fire button is pushed and the stick is pushed up and left, then bits 0, 2, and 4 will change to a '1'.
binary 00010101 = decimal 21

INSTRUCTIONS:

STEP 1> Make sure R.O.S. is loaded and type: **LOAD"JOY BYTE",8 <return> then RUN <return>.**

STEP 2> Plug a joystick into port #2. Use only one stick.

Watch the lights on the B100. When bit 0 is a binary 1, the light is flashing, otherwise it is OFF. The other lights function just the same.



Move the joystick up. The bit 0 LED should now be flashing.

Referring to the **Joystick Byte Chart**, it agrees and indicates that the joystick is up. While the joystick is up, press the **FIRE** button. Notice that bit 4 is "on" as well as bit 0.

If we add the bit values together, we get a decimal 17. This value will now be showing on the screen.

With the **FIRE** button pressed and the joystick **DOWN** and to the **RIGHT**, look at the lights. Bit 4, bit 3 and bit 1 will be flashing. The decimal value is 26.

Experiment with different stick positions. Try to produce all possible joystick combinations. How many are there? (Refer to project #19).

If we're looking at an 8 bit binary number, why aren't there 256 different combinations? Because the upper 3 bits are not used, and because only certain switches can be closed at any one time. For example, without tearing the joystick apart, you cannot close all 5 switches at once.

PROJECT #23

USING JOYSTICKS IN YOUR PROGRAMS

DESCRIPTION:
You should understand projects 19, 20, and 22 before doing this project.

Now that you know how the computer reads the joystick, you can try using the **JOY** command in your own programs.

Remember: It will be very difficult to write your own robotic programs if you do not have a minimal understanding of BASIC programming. You should, at the very least, understand **line numbering**, how **variables** work, **'IF'** statements, **GOSUB's** and **GOTO's** and how to **RUN** and **STOP** programs. Refer to your COMPUTER USER MANUAL for help.

INSTRUCTIONS:

STEP 1> Make sure R.O.S. is loaded, then type **NEW <return>.** This command (NEW) will erase any Basic program that might be in memory. (If it is your program and you want to keep it, you'll have to save it first. Once you type NEW and hit <return> the program will be lost.)

STEP 2> Type in the following program. Type only the lines that are in Bold Face. Press <return> at the end of each line.

```
100 JOY2,A
```

This transfers the status of the joystick (plugged into port 2) to the variable A.

```
105 PRINT A, S
```

This will print the status of the joystick and the variable 'S' to the screen. 'S' will equal 0 when the program starts.

```
110 IF A=1 THEN S = S + 1
```

If the joystick is UP, add 1 to the variable 'S'.

```
120 IF A=2 THEN S = S - 1
```

If the joystick is DOWN, subtract 1 from the variable 'S'.

```
130 IF A=16 GOTO 1000
```

The joystick button is pressed, go to line 1000.

```
140 MOTR1,S
```

This statement will turn on motor #1 at a speed equal to the variable 'S'.

```
150 GOTO 100
```

This repeats the cycle.

```
1000 MOTR0:END
```

This turns off the motor and exits the program.

STEP 3> Now type: **LIST <return>**
The program should be printed on the screen by consecutive line numbers. Check to see that you've entered it correctly.

STEP 4> Now type: **RUN <return>**
All L.E.D.'s on the B100 should be off and the numbers on the screen should be zero.

Move the joystick up briefly. Notice that while the joystick was up, the left most number on the screen (the variable **'A'**) was a 1 and the second number (the variable **'S'**) increased. Notice also that BIT 0 LED is blinking. As you move the joystick up and down, the LED will blink faster and slower. This blinking is a measure of motor speed.

When the speed (the variable **'S'**) goes negative, LED for BIT 1 is on and blinks.

From this you can see that if LED 0 is on, the motor is going forward. If speed is negative, then LED 1 comes on and the motor is going in reverse.

Connect a motor to leads for motor #1 and turn the motor switch on the B100 to ON.

Notice how you have complete control of the motor speed and direction.

NOTE: After speed reaches 30 or -30, the LED's and the motor speed will not change even though the value of speed will continue to change. This is because 30 is the maximum (as set by the **RANGE** command). Normally, we would test the value of **'S'** using another **'IF'** statement and hold it to 30 or less. This could be done by adding two lines to the program.

```
132  IF S > 30 THEN S = 30
134  IF S < -30 THEN S = -30
```

If you would like to try it, do the following.

1. Press **RUN/STOP.**
2. Type in the above lines, hitting <return> after each line.
3. Type: **LIST <return>** and check the program.
4. Type: **RUN <return>.**

When building projects, especially cars, it is fun to control them with the joystick. The **JOY** command in R.O.S. provides easy access to the joystick ports. In the above example we used only one stick, joystick #2. The form of the command was **JOY2,VARIABLE.** The command for stick #1 is **JOY1,VARIABLE.** Joystick #1 will work right along side joystick #2, but it does not work well with the keyboard. This is because stick #1 and the keyboard share part of the same port, and either or both can get confused when using them together in the same program. The safest approach is to never use stick #1 with the keyboard.

We've included a program on disk that you may use as a subroutine to read the sticks for you. To use this subroutine, load the program called **JOYSTICK.** Place your program before line 10000. When you want to read the joysticks, enter a GOSUB 10000. When you return from the subroutine, the following variables will be set.

	JOYSTICK #1	JOYSTICK #2
UP	U1 = 1	U2 = 1
DOWN	D1 = 1	D2 = 1
LEFT	L1 = 1	L2 = 1
RIGHT	R1 = 1	R2 = 1
BUTTON PRESSED	B1 = 1	B2 = 1

All variables will be zero if no movement is detected.

Below is an example of how to use the joystick subroutine.

```
300  GOSUB 10000                     :REM  READ THE STICKS
310  IF U1 = 1 THEN SPEED = SPEED +1
320  IF D1 = 1 THEN SPEED = SPEED -1
330  IF B1 = 1 THEN SPEED = 0
```

ADVANCED PROGRAMMER'S NOTE:
R.O.S. inverts and masks the actual port read (for your convenience) by performing the following:

1. READ PORT (LDA $DC00 or LDA $DC01)
2. INVERT ALL BITS (EOR #$FF)
3. STRIP OFF UPPER 3 BITS (AND #$1F)

PROJECT #24

EFFECTS OF THE RANGE COMMAND

DESCRIPTION:
As you have seen in other projects, the motor's speed is controlled by a technique called PULSE WIDTH MODULATION (See Project #6). In default mode, there are 30 cycles associated with each motor. A speed of 5 would give us 5 ON cycles and 25 OFF cycles (5 + 25 = 30). The **RANGE** command changes the number of cycles for each motor.

For example, suppose we gave the command: **MOTR1,15.** The motor would be ON 15 cycles and OFF 15 cycles. We can get smoother operation of the motor if we change the range to 2 and set the speed to 1 (1 ON and 1 OFF). The speed will be about the same, but it will run smoother.

On the other hand if we need very fine speed control, we can increase the range to provide more steps in the speed process.

INSTRUCTIONS: Try this experiment:

STEP 1> Make sure R.O.S. is loaded.

STEP 2> Connect motors to the motor #1 leads and the motor #2 leads.

STEP 3> Type: **RANGE2,2 <return>.** This will change the motor #2 range from 30 (the default value) to 2.

STEP 4> Turn both motors on at half speed by typing **MOTR1,15:MOTR2,1 <return>.** Notice how much smoother motor #2 is running even though the speeds are close to the same.

STEP 5> Type: **MOTR0 <return>.**

STEP 6> Now Type: **LOAD"RANGE",8 <return> then RUN <return>.**

Follow the prompts on the screen to select different ranges and speeds for motor #2. Motor #1 speed will be adjusted as close as possible to that of motor #2 so that you may see the difference.

PROJECT #25

USING THE ADVANCED MOTR COMMAND

DESCRIPTION:
The Multibotics R.O.S. system provides a wide variety of speed control using the normal **'MOTR'** command and the **'RANGE'** command. There is also an advanced version of the **'MOTR'** command which is useful.

The **SYNTAX** of the command is as follows:

MOTR VAR1,VAR2,VAR3,VAR4,

```
VAR1=   Motor# (1-3)
VAR2=    1 = Motor forward
         0 = Motor off
        -1 = Motor reverse
VAR3=   Number of ON cycles (0-255)
VAR4=   Number of OFF cycles (0-255)
```

INSTRUCTIONS:
Try out some examples. Make sure R.O.S. is loaded. Connect a motor to leads #1 and make sure the ON/OFF switch is ON.

Type: **MOTR 1,1,10,200 <return>.**

Notice the jerky motion of the motor.

Type: **MOTR 1,-1,10,200 <return>.**

The motor still has jerky motion, only in reverse. Notice LED1. The motor receives power only when LED1 is ON.

Type: **MOTR 1,1,1,3 <return>.**

You can hardly see the LED flash because it is going ON and OFF so fast. Notice how "smooth" the motor is running.

Type: **MOTR 1,10 <return>.**

This is the same speed as the previous example, but notice the motor is not running as "smooth" as before. This is the benefit of the advanced **MOTR** command. You have complete control of all motor parameters.

PROJECT #26

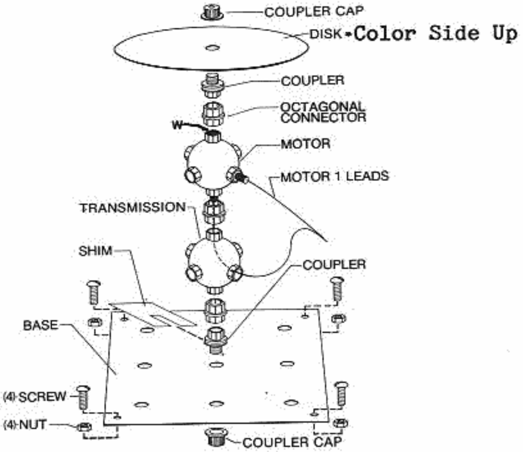
LIGHT AND COLOR

DESCRIPTION:
Before the time of Sir Issaac Newton white was considered a color along with all other colors that our eyes could perceive. In fact, white was thought to be the simplest of colors. We now know, however, that white light (sunlight) is not simple, but is made up of six major colors; red, orange, yellow, green, blue and violet. These are called the colors of the **VISIBLE LIGHT SPECTRUM** and are tne same as the color of a **RAINBOW.** When white light shines on a surface, some colors are reflected and some are absorbed. If most of the light is reflected in equal amounts, the surface will appear white. If most of the light is absorbed in equal amounts, the surface will appear black. If all of the colors are absorbed except red, then the surface will appear to be red, and so on.

In this project we will attempt to produce the colors of the spectrum from only 3 primary colors, (red, yellow, and blue). In theory this will work, because orange is the sum of red and yellow, green is the sum of yellow and blue, and violet is the sum of blue and red.

INSTRUCTIONS:

STEP 1> Assemble the project shown below.



Parts required:

2	Coupler Caps
1	Base
2	Couplers
3	Octagonal Connectors
1	Transmission
1	Motor
1	Color Wheel
1	Shim

Notice that the motor lead passes through the hollow transmission capsule.

STEP 2> Make sure R.O.S. is loaded. Then type: **LOAD"COLOR WHEEL",8 <return> then RUN <return>.**

STEP 3> The keys **A** thru **Z** will select the speed of the disk the <-- (left arrow) will reverse directions. The **SPACE** bar will stop rotation. Notice how the colors blend.

Did we get all the colors of the SPECTRUM.

RED
ORANGE
YELLOW
GREEN
BLUE
VIOLET

Well, the answer is **NO.** We cannot get a good **green** to appear between **yellow** and **blue.** The color wheel is approximating color blending by forcing the eye to switch rapidly from one color to another. In the case of **RED to YELLOW** and **BLUE to RED** the eye (or brain) makes a compromise which results in **ORANGE** and **VIOLET** respectively. However, in the case of **YELLOW to BLUE,** the colors seem to cancel rather than compromise, and the result is a neutral or gray tone.

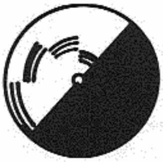
Color blending is the basis of color television. Most T.V.'s have 3 color 'guns', red, green, and blue, all other colors are produced by color blending.

PROJECT #27

BENHAM'S DISK

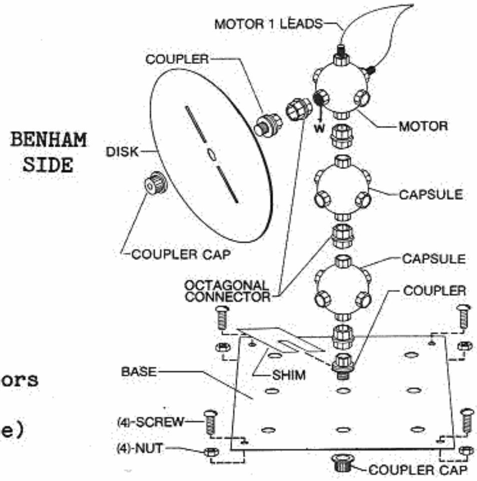
DESCRIPTION:
When white light is suddenly cut off from the eye, the sensations of all the colors do not disappear at the same time. The blue rays seem to persist a fraction of a second longer. On the other hand, when white suddenly appears from behind black, the red rays apparently affect the eye first. If we make a disk with black upon white, the BENHAM disk, and then rotate it at a certain speed, in one direction, the outer rings appear red and the inner ones blue, while those in between give intermediate hues. Now if we rotate the disk in the opposite direction, the inner rings will appear red and the outer will be blue. If the rotation is very rapid the effects will be simply gray. It must be just rapid enough so that the impressions of the arcs merge in the eye into sensations of complete circles.

On the reverse side of the color wheel is printed a version of BENHAM's disk.



INSTRUCTIONS:

STEP 1> Assemble the project shown below:



Parts Required:

1	Base
2	Coupler Caps
1	Shim
2	Couplers
4	Octagonal Connectors
1	Benham Disk
2	Capsules (Any type)
1	Motor

STEP 2> Make sure R.O.S. is loaded then Type: **LOAD"BENHAM DISK",8 <return> then RUN <return>.**

STEP 3> Follow the screen prompts.

The colors appear most plainly when viewed from a distance of about six feet or less, and when the disk is held in a fairly strong light. The red has most intensity and becomes a deep maroon when the disk is held close to an artificial light. Remember not to spin the disk too fast.

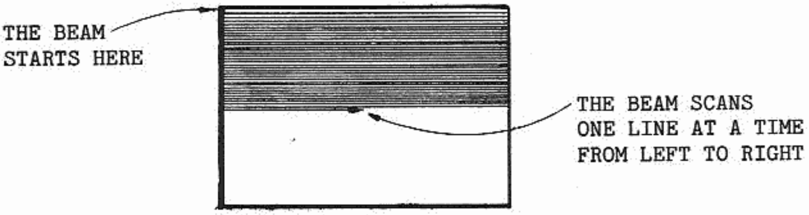
PROJECT #28

STROBE EFFECT

DESCRIPTION:
Have you ever seen an old cowboy movie on T.V. and noticed that the wagon wheel spokes seem to stand still at times? The shutter speed of the movie camera is synchronizing with the speed of the wagon wheel. Each time the shutter opens to take a picture, the wheel has rotated just enough so that it appears to be in the same position as the previous picture. If the wheel is just slightly "out of sync" with the camera, then it will appear to rotate slowly one direction or the other. This is called the "STROBE EFFECT".

In this project, we will create the same effect using the Robotic Workshop and your T.V. or monitor.

The screen area of the television picture tube consists of thousands of tiny phosphor dots. A narrow electron beam is projected from the rear of the tube onto the screen where it can strike these dots and make them glow. A black and white T.V. has one electron beam and a color T.V. will normally have three beams. The electronic circuitry can determine exactly where the beam is pointing. The picture is created in the following manner: The beam starts in the upper left hand corner (as we face the screen) and travels horizontally from left to right striking and illuminating the phosphor dots as it goes. The speed is such that it will strike the dots just long enough to make them glow. When the beam reaches the right edge of the screen, it quickly shuts off and returns to the left edge of the next line. This return period is called the **HORIZONTAL RETRACE.** The beam then turns back on and begins scanning the next line at the same rate as the first. This process continues until it has scanned all the lines and has reached the lower right corner of the screen.

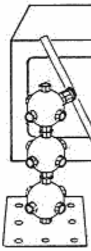


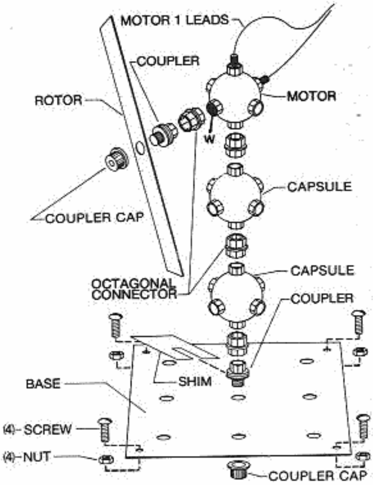
The number of scan lines varies slightly for different computers. The Commodore 64 (60Hz version) has about 262 scan lines. When the beam has reached the lower right corner of the screen, it again shuts off and quickly returns to the upper left corner of the screen. This return period is called the **VERTICAL RETRACE** or **VERTICAL BLANK.** It takes only 1/60th of a second for the entire screen to be scanned and for the beam to return to its starting position. Thus the T.V. is showing us 60 new frames every second. We see only continuous motion, because our eyes are unable to separate the images at that rate.

Suppose we place an object in front of the screen, and spin it at 60 revolutions per second. Every time the screen is scanned, the object will have made one complete rotation and should be in exactly the same location. It should therefore appear to be standing still.

INSTRUCTIONS:

STEP 1> Assemble the project shown below.





Parts required:

1	Base
2	Coupler Caps
1	Shim
2	Couplers
4	Octagonal Connectors
2	Capsules (Any type)
1	Motor
1	Rotor

STEP 2> Place the project in front of the monitor or T.V. so that you must look 'through' the rotor to see the screen.

STEP 3> Make sure R.O.S. is loaded and then type: **LOAD"STROBE",8 <return> then RUN <return>.**

STEP 4> Press the **SPACE** bar to start the motor. Press **F1** to increase the speed and press **F7** to decrease the speed. Any other key will stop the motor.

At 60 revolutions per second, the rotor will appear to be standing still, but will have a dramatic curvature. This is because on one side the rotor is traveling with the beam (towards the bottom of the screen) and on the other side the rotor is traveling against the beam.

If the rotor is spinning at 30 RPM, there will appear to be two images. This is because the rotor has completed only half a revolution by the time the beam comes by again. See if you can find other multiple images and determine what the RPM is.

This strobe technique is used often in industry to determine the RPM of motors, turbines and other rotating equipment. A special light, called a strobe light is used in place of the T.V. screen. It is very bright and flashes on and off quickly. The strobe light is pointed at a portion of the equipment which is rotating, such as a drive shaft or flywheel. Then the rate of flashing is gradually adjusted until the object appears to stand still. The flashing rate of the strobe light, in flashes per second, is the same as the RPM of the rotating object. If you have access to a strobe light, repeat the project above, using the strobe instead of the T.V.

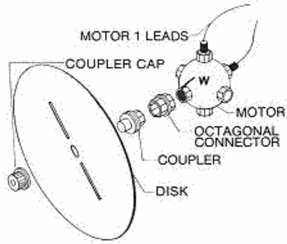
PROJECT #29

COLOR WHEEL STROBE

DESCRIPTION:
The color wheel has two slits that can be used to observe the strobe effect.

INSTRUCTIONS:

STEP 1> Hook up a motor to the #1 leads and the color wheel to the motor, as shown below.



STEP 1B> Punch out slits in color wheel.

STEP 2> Hold the motor and point the color wheel so that you have to look through the color wheel to see the computer screen.

STEP 3> Make sure R.O.S. is loaded and then type: **LOAD"COLOR WHEEL STROBE",8 <return> then RUN <return>.**

STEP 4> Adjust the speed with the letters from **A** to **Z.** The <-- (left arrow) key changes the direction. The **SPACE** bar stops the program.

As you spin the color wheel at various speeds, wait a short time between changes to ensure that the motor gets up to speed.

Watch the slits as they stand still and move slowly.

Hold the wheel close to the screen so that the light from the screen lights up the wheel and observe it from the front. Now remove the color wheel and try it with a regular wheel without the tire.

Refer to the previous project for information on how the strobe effect works.

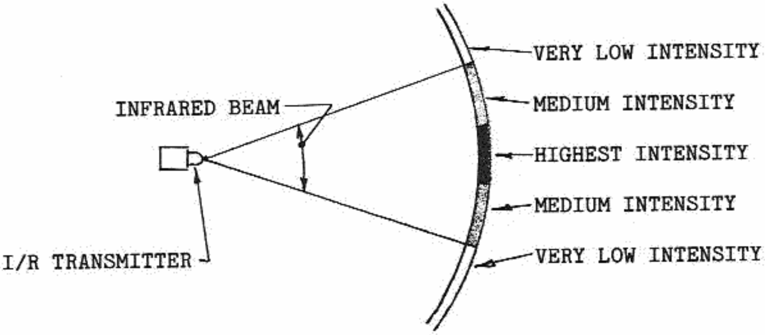
PROJECT #30

ALIGNING THE I/R SENSORS

DESCRIPTION:
The infrared (I/R) sensors are among the most interesting items in the ROBOTIC WORKSHOP. They can be used for many functions such as:

Measuring RPM
Measuring Speed
Counting Events
Burglar Alarm
Etc.

The sensors must work in pairs. One is called the **TRANSMITTER** and is similar to a miniature flood light. It shines an invisible beam of light which spreads out as it gets further from the source. Refer to the figure below.

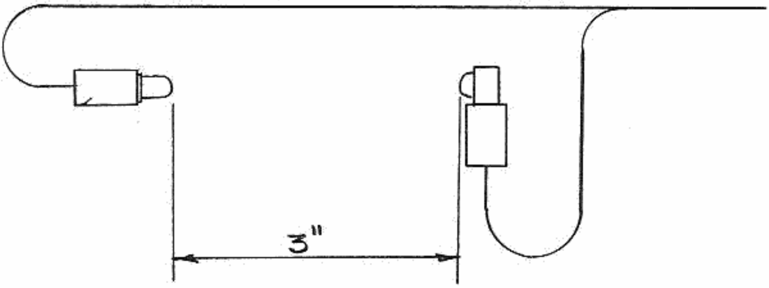


The other sensor is called the **RECEIVER** and is sensitive to Infrared light. It can signal the computer when it detects a strong I/R beam. Infrared light is invisible to the human eye. It is just **below** the VISIBLE LIGHT SPECTRUM that we discussed in project #26. Its wavelength is a little longer than the red light which we can see and hence the name, INFRARED. Common lamps emit certain amounts of infrared light along with the visible light. Because of this, it is important to avoid bright external lights when performing I/R experiments. In some cases you may have to shield the RECEIVER on all sides except straight ahead (the direction of the transmitter). This can be done with black tape, a black tube, or some other opaque material.

Normally the sensors will be used in the following manner. First, the sensors are positioned facing each other such that the receiver is detecting the beam. This is called the **ALIGNING** process. At this point, R.O.S. can tell that the beam is established. When an object passes between the transmitter and receiver and breaks the beam, R.O.S. will detect a transition. When the object is moved and the beam is restored, R.O.S. will detect another transition. A command is available which will count transitions and report them to the program. **Remember: A <u>transition</u> is either the breaking <u>or</u> the restoring of the beam.**

INSTRUCTIONS:
To help with the ALIGNMENT procedure, a special command has been added to R.O.S. This command works only in the direct mode (not while a program is running). This project is to show you how to use the **ALIGN** command.

STEP 1> Set the I/R sensors flat on a table, facing each other about 3 inches apart. See the figure below.



Remember: Under bright conditions, it may be necessary to cover the sides and back of the RECEIVER to keep extraneous light from affecting the sensor readings.

STEP 2> Make sure R.O.S. is loaded and that the selector switch is in the SENSOR position.

STEP 3> Now type: **ALIGN8,7 <return>.** The screen display will show when the sensors are aligned or misaligned. Move the sensors in and out of alignment to get the feel of how they work and their range, etc.

STEP 4> Press the **RUN/STOP** key to terminate the alignment process.

Before using the I/R sensors in any project, it is a good idea to check the alignment with the **ALIGN** command.

NOTE: **SENSOR MOUNTING**
We have provided a small strip of double faced tape in the WORKSHOP for mounting the I/R sensors. One side of the tape is very sticky and should be attached to the white plastic sockets which hold the sensors. The other side is not as sticky and is designed for temporary attachment to walls, tables, motor capsules, etc. It can be attached and removed several times.

Cut small pieces from the tape strip and use them until they will no longer adhere properly. If you are careful with the mounting tape, it will last a long time.

NOTE: **SENSOR REPLACEMENT**
The TRANSMIITER and RECEIVER must be plugged in to their own sockets in the proper orientation. If they are removed and must be replaced, refer to the technical section of the manual for proper location and orientation.

PROJECT #31

MEASURING RPM

DESCRIPTION:
This project will show you how to use the I/R sensors to measure the RPM (Revolutions Per Minute) of a rotating object. Once you've learned the technique, you'll be able to perform similar projects yourself.

You'll need to learn two new commands in order to understand how RPM is calculated. It is not necessary, however, that you understand these commands to perform the projects.

The first new R.O.S. command is called **SENSR.** The SENSR command is used to set up the sensors so that R.O.S. knows what you wish to do.

First, we'll turn on the **TRANSMITTER**, which produces the I/R beam. This is done by setting up the TRANSMITTER (sensor #7) as an OUTPUT sensor. The command is:

```
SENSR7,2              Where 7 = sensor #(1-8)
                            2 = output (1 = input)
```

The next step is to set up the **RECEIVER** (sensor #8). This is done in a similar fashion except for one exception, as you'll see.

The command is;

```
SENSR8,1,333          Where 8 = Sensor #(1-8)
                            1 = Input
                          333 = Counting Interval
```

In the above command we have added a third variable (333). If the sensor is set up for INPUT (which this one is), we can tell R.O.S. to count transitions of the beam for a period of time and save that value. This third variable is called the **COUNTING INTERVAL.** It is expressed in cycles rather than minutes or seconds. In normal mode, a cycle is 3/1000 of a second (.003 seconds). Thus a counting interval of 333 cycles represents 1 second. The value of a cycle can be changed if needed, but in most cases .003 seconds will be plenty fast enough.

This may all seem a little complicated, so if you didn't understand it very well, just remember this:

1st Set up the TRANSMITTER with:
SENSR7,2

2nd Set up the RECEIVER to count for 1 second with:
SENSR8,1,333

NOTE: If you want to count for more than one second, say 3 seconds, simply use:

SENSR8,1,333*3 (333*3 = 3 seconds)

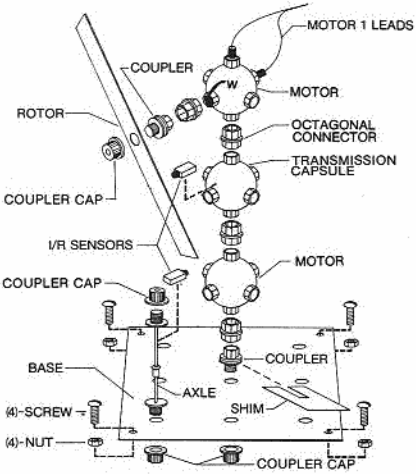
So far we've only talked about the 1st new command. The 2nd new command is called **SPCNT**, which stands for SENSOR PREVIOUS COUNT. This command is used to read the number of transitions of the I/R beam. Remember that we told R.O.S. to count transitions for 1 second (333 cycles) and to save the value. SPCNT will retrieve the value and place it in a variable.

The form is:

SPCNT 8,A   Where 8 = Sensor #(1-8)
A = Variable to hold the value

INSTRUCTIONS:

STEP 1> Assemble the project shown below:



Parts required:

1	Base
3	Coupler Caps
1	Shim
4	Octagonal Connectors
2	Couplers
1	Transmission Capsule
2	Motors
1	Rotor
1	Axle
2	I/R Sensors
1	I/R Sensor Mounting

STEP 2> Make sure R.O.S. is loaded and set the selector switch to SENSORS and the ON/OFF switch to ON.

STEP 3> Adjust the sensors using the **ALIGN 8,7 <return>** command. With the rotor in a horizontal position, the sensors should be adjusted so that they are aligned. When the rotor is moved in between the sensors, they should be misaligned.

Press **RUN/STOP** to exit.

STEP 4> Now type: **LOAD"RPM",8 <return> then RUN <return>**

The screen is displaying the RPM of the motor. The **F1** key increases the motor speed while the **F7** key decreases the speed.

RPM is measured by the program for 15 seconds and then the screen is updated. The program is structured as follows: (If you are not familiar with BASIC Programming, you may skip this section.)

```
1000     SENSR8,1,333*15     Set up receiver to count for 15 seconds
1010     SENSR7,2            Set up transmitter
1040     SPCNT8,A            Read the count into variable 'A'
1050     R=A/4               Number revolutions in 15 seconds
1060     RPM = R*4           Number of rotations per minute
1070     Print RPM           Print the answer
1080     GOTO 1040           Do it again
```

Notice that, after the count is obtained in line 1040, we have to perform some calculations in order to get the answer in revolutions per minute (RPM).

What we have is the number of transitions which have occurred in 15 seconds. Transitions need to be changed into revolutions and the time period needs to be one minute. Remember that a transition is counted when the beam is broken and again when the beam is restored. Two blades of the rotor pass through the beam on each revolution and each blade produces two transitions. Therefore, 4 transitions (counts) represent one revolution, and the total number of revolutions is equal to the count (A) divided by 4 as in line 1050. The time interval during which the count was taken was only 15 seconds. We have to multiply the revolutions times 4 to get RPM as is done in line 1060. We then print the result to the screen in line 1070 and loop back to line 1040.

CAUTION!

DO NOT PLACE ANYTHING IN THE PATH OF THE ROTOR WHILE IT IS SPINNING

PROJECT #32

CALCULATING SPEED FROM RPM

DESCRIPTION:
In the previous project, we used an infrared sensor to determine RPM. RPM (revolutions per minute) is a measure of **rotational speed** and is not the same as regular speed.

Regular speed is expressed in units of length divided by time, such as MILES/HR.

Suppose we connected our measuring device to a car and could determine the RPM of one of the wheels. Would it be possible, to calculate the speed of the car in miles per hour? It would be, if we knew the diameter of the tire.

Here's an example of how to calculate the speed of a car by knowing the RPM of the wheel and the size of the tires.

EXAMPLE:

Let's say we measure 1000 RPM and the tire is 24 inches in diameter.

```
Rotation Speed = 1000 RPM
Tire Diameter  = 24 inches
```

As the tire makes one complete revolution, how far will the car travel (assuming no slippage)?

The amount that the car moves will be equal to the circumference of the tire or the distance around the tire. The circumference of any circle is give by the formula:

```
C = PI * D             C = circumference
                      PI = 3.14 (a constant)
                       D = diameter of the circle
```

Therefore, the circumference of our tire is equal to:

```
C = 3.14 * 24 inches
C = 75.36 inches
```

This is also the distance the car moves in one revolution of the tire.

Recall that the wheel was spinning at a rotational speed of 1000 revolutions per minute.

Since we now know the distance traveled per revolution, we can calculate the distance traveled per minute as follows.

```
SPEED = (1000 Rev/Minute) (75.36 inches/revolution)
SPEED = 75360 inches/minute
```

All that is left to do now is to convert from inches per minute into miles per hour.

Since there are 60 minutes in 1 hour.

SPEED = (75360 inches/min)*(60 min/hr)
SPEED = 4,521,600 inches/hr

Also there are 12 inches in 1 foot so;

$$\text{SPEED} = \frac{4{,}521{,}600 \text{ inches/hr}}{12 \text{ inches/ft}}$$

SPEED = 376,800 ft/hr

And there are 5,280 ft in 1 mile,

$$\text{SPEED} = \frac{376{,}800 \text{ ft/hr}}{5{,}280 \text{ ft/mile}}$$

SPEED = 71.36 miles/hr
= 71.36 MPH

If you didn't follow the math, that's okay. What's important is the relationship of rotation speed (sometimes called angular velocity) to regular speed (velocity).

Notice that the diameter of the tire was all that was needed to calculate miles per hour. What would happen to the speed of the car if the tires were only 1/2 the diameter? For the same RPM,the speed would be only half as much or 35.68 MPH.

PROJECT #33

MEASURING THE SPEED OF A BASEBALL BAT (OR ANY OTHER OBJECT)

DESCRIPTION:
In this project, we'll use the I/R sensors to measure the speed of an object passing through the invisible beam.

Recall that the sensors will detect a transition of the beam from **ON to OFF** or from **OFF to ON.** As an object passes between the sensors, it will break the beam upon entering and restore it upon leaving. We can measure the amount of time that the beam was interrupted, and by knowing the thickness of the object, we can determine its speed.

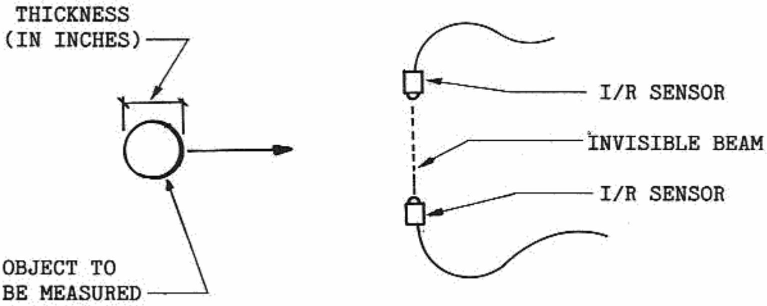
INSTRUCTIONS:

STEP 1> Place the I/R sensors in a suitable position, far enough apart so that an object (like a baseball bat) can pass between them, but close enough to get a good alignment.

STEP 2> Make sure R.O.S. is loaded and set the selector switch to sensors.

STEP 3> Type: **ALIGN8,7 <return>** and adjust the sensors for proper alignment. Press **RUN/STOP** to exit.

STEP 4> Type: **LOAD"MEAS SPEED",8 <return> then RUN <return>**



Follow the screen prompts. First answer the questions about the thickness of the object being tested. Then pass it through the beam and the program will measure its speed. To reset for another measurement, press any key. Try different speeds from very fast to very slow.

Remember that $\text{Speed} = \frac{\text{Distance}}{\text{Time}}$

In this case $\text{Speed} = \frac{\text{Thickness}}{\text{Time}}$

R.O.S will report the number of cycles it took for the object to pass through the beam. A cycle (in normal mode) is .003 seconds. This cycle duration can be decreased if needed. See the Advanced Programming section.

The equation, which will calculate speed based on the thickness of the object and the number of cycles, is shown below.

$$\text{Speed} = \frac{\text{thickness}}{\text{cycles} * \frac{.003 \text{ seconds}}{\text{cycle}}}$$

$$\text{Speed} = \frac{\text{thickness(inches)} * \frac{1 \text{ ft}}{12 \text{ inches}} * \frac{1 \text{ mile}}{5280 \text{ ft}}}{\text{cycles} * \frac{.003 \text{ sec}}{\text{cycle}} * \frac{1 \text{ hr}}{3600 \text{ sec}}}$$

$$\text{Speed} = \frac{\text{thickness} * 18.939394}{\text{cycles}} \quad \text{miles/hour}$$

PROJECT #34

COUNTING THINGS

DESCRIPTION:
This project will demonstrate how the ROBOTIC WORKSHOP can be used to count objects. Using the infrared sensors, R.O.S. can count up to 8,288,607 objects before starting over.

INSTRUCTIONS:

STEP 1> Position the I/R sensors about 3 inches apart and place the selector switch in the SENSORS position.

STEP 2> Align the sensors as you have done previously using the **ALIGN8,7** command. You may also have noticed by now that LED #7 will be **ON** when the sensors are aligned and **OFF** when the sensors are misaligned.

STEP 3> Now type: **LOAD"I/R COUNT",8 <return> then RUN <return>.**

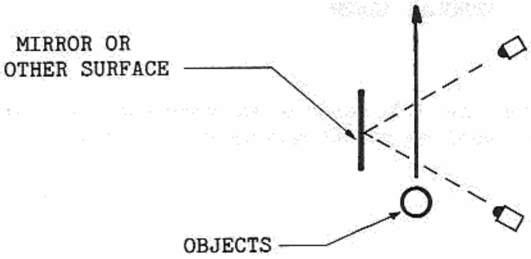
The program will now count anything which interrupts the invisible beam. Try passing several objects between the sensors and see if it works.

There is a new command, which is used in this program, that we have not discussed before, The command is **SCCNT** (Sensor Current Count). It may be used in the following manner.

```
1000     SENSR 7,2               Set up the transmitter for output
1010     SENSR 8,1               Set up the RECEIVER for input (No
                                 Timing Interval)
1030     SCCNT 8,A               Place the number of transitions
                                 that R.O.S. has detected into the
                                 variable 'A'.
```

Since no timing interval was specified in line 1010, R.O.S. will interpret the SENSR 8,1 command as **continuous counting mode** rather than **timed mode.** R.O.S. will simply keep track of sensor transitions unless told to stop. We can get the current value from R.O.S. by issuing the **SCCNT 8,A** command. It tells R.O.S. to transfer the count for sensor #8 (the I/R receiver) into the variable 'A'. Remember that 2 transitions will occur for each object passing through the beam, so be sure and divide the variable 'A' by 2 to get the actual object count.

Another mounting arrangement for the sensors is shown below:



The I/R sensors are sensitive devices and will even reflect off of tables, walls, etc.

NOTE: R.O.S. is working when BASIC isn't. Once this program has been executed, even if you stop it, R.O.S. will keep counting! To try this, press the **RUN/STOP** key which will stop the program. Now, interrupt the I/R beam several times, and then type: **CONT <return>**. This will let the program resume.

Notice that the count is correct!

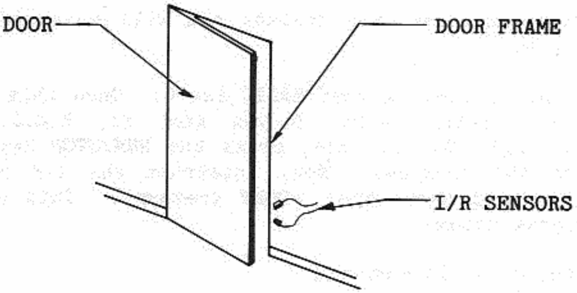
In its default condition, R.O.S. can count up to 333 items per second. In the ADVANCED `PROGRAMMING section, you can learn how to make it count at even faster rates.

Think of some uses in your own program for this counting feature. How about counting the number of times a door opens. Counting devices similar to this one are used in industry to count all types of products from tuna fish cans to golf balls.

PROJECT #35

BURGLAR ALARM

DESCRIPTION:
In this project we'll use the I/R sensors to construct a simple burglar alarm. The sensors must be positioned as shown below.



BEAM REFLECTS
OFF THE EDGE OF
THE DOOR WHEN
OPENED SLIGHTLY

INSTRUCTIONS:

STEP 1> Attach the sensors to the door frame as shown above.

STEP 2> Make sure R.O.S. is loaded and that the selector switch is in the SENSORS position.

STEP 3> Type: **ALIGN8,7 <return>.**
Open the door slightly and adjust the sensors so that the beam from the transmitter bounces off the edge of the door and back to the receiver. The sensors should be **aligned** only when the door is **slightly open.**

When the door is closed or wide open, the sensors should be **misaligned.**

STEP 4> Now type: **LOAD"BURGLAR ALARM",8 <return> then RUN <return>**

The alarm is now activated. When the door is opened the alarm will sound. Once the door has been opened, "CAUTION!!!" will remain on the screen to let you know the door was opened.

The command which is used to the alarm program is call **SDIR.** It is used in the following manner.

```
100  SENSR 7,2                        Set up the sensors as before
110  SENSR 8,1
120  SDIR 8,A                         Read the sensor
                                      (0 = broken    1 = not broken)
130  IF A = 1 THEN 120                Loop to line 120 if not broken yet
```

PROJECT #36

USING THE 'MASTR' COMMAND

DESCRIPTION:
The **MASTR** command is the most powerful command in R.O.S. It allows very precise control of several devices at the same time.

In this project, the **MASTR** command will be used to enhance the BURGLAR ALARM program. (See project #35.) The alarm will be set to activate after the door has been opened and closed once. This will permit a person to leave the protected room without setting off the alarm. When the door is opened the next time, the alarm will sound and motor #1 will turn on.

The command line appears as follows in the enhanced alarm program.

```
MASTR 1,47,3,177,10,0
```

- The **1** is for master byte #1 (up to 16 are possible)
- The **47** tells R.O.S. to watch the sensor count
- The **3** means to count 3 transitions before turning on the motor
- The **177,10,0** will turn on motor #1, continuously

INSTRUCTIONS:

STEP 1> Use the same setup as you did for project #35 and connect a motor to motor 1 leads.

STEP 2> Make sure R.O.S. is loaded and switch the selector switch to SENSORS, and the ON/OFF switch to ON.

STEP 3> Type: **LOAD"MASTER BURGLAR",8 <return> then RUN <return>.**

After 3 transitions of the beam,the alarm will sound. The program should be started with the door open wide. As the door is closed, 2 transitions will occur. When the door is opened the next time, the 3rd transition will be recorded and will set off the alarm and start the motor. Changing the 3 in the command line will change the number of transitions before the alarm sounds.

The **MASTR** command is quite complicated. It was used in this project to demonstrate its power and flexibility but only advanced programmers should attempt to use it in their own program. More information on the MASTR command can be found in the Advanced Programming Section of this manual.

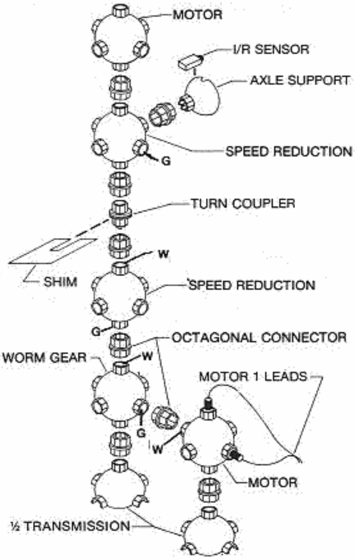
PROJECT #37

RADAR

DESCRIPTION:
This project is a simple infrared radar unit. One I/R sensor is connected to a rotating capsule. Its direction is controlled by motor #1 through a set of 2 gear capsules. The person performing the experiment (or a helper) holds the other I/R sensor and points it towards the radar tower at a distance of about 12 inches. The capsule will rotate slowly until it finds and "zeros in" on the I/R beam.

INSTRUCTIONS:

STEP 1> Assemble the project shown below:

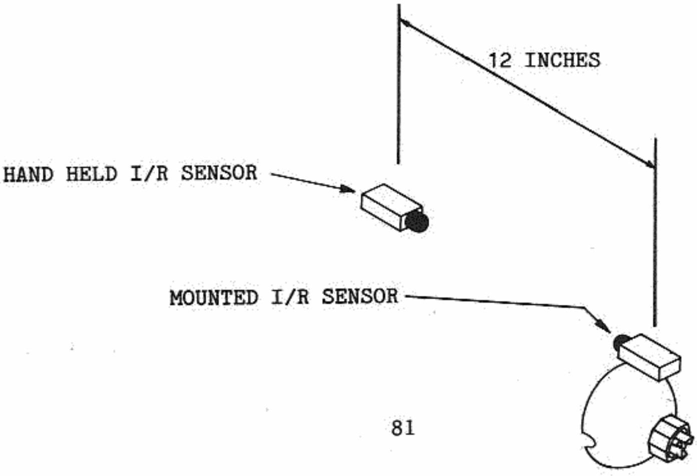


Parts required:

1	Transmission
7	Octagonal Connectors
2	Motors
1	Worm Gear
2	Speed Reduction
1	Axle Support
2	I/R Sensors

STEP 2> Make sure R.O.S. is loaded then set the selector switch to SENSOR and the ON/OFF switch to ON.

STEP 3> Hold the 2nd I/R sensor about 12" away from the model, **NOT** pointing at the mounted sensor.

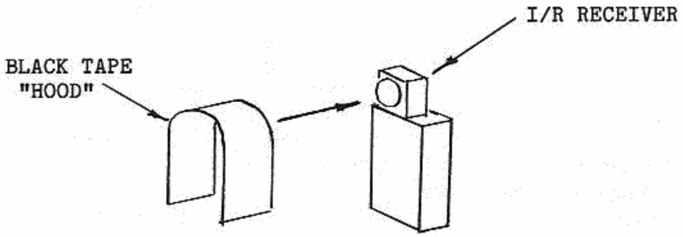


STEP 4>   Type: **LOAD"RADAR",8 <return> then RUN <return>.**

The motor will start spinning and the capsule will rotate. When it **'finds'** the hand-held sensor, the motor will slow down and reverse. When it finds the sensor again, it slows down and reverses again. This process will continue until the motor has stopped. It should happen very quickly.

NOTE: If the radar has trouble finding the beam, keep an eye on LED 7. When the sensors are pointed at each other the LED should be off.

If you're still having trouble, you may have to wrap the receiver with black tape, as bright light sources can fool the sensors.



PROJECT #38

CHECKING BATTERIES

DESCRIPTION:
The ROBOTIC WORKSHOP contains a built in D.C. voltmeter. This voltmeter, called **MULTIVOLT,** can measure up to +3.6 volts. Measurements above +3.6 VDC will require the use of a **shunt** resistor. See the MULTIVOLT section for specific details.

In this project we will measure the voltage of a single AAA, AA, C or D battery. You may actually remove and test the batteries in the B100 interface if you wish. The batteries are used only to drive the motors and may be removed without effecting the other operations of the interface.

INSTRUCTIONS:

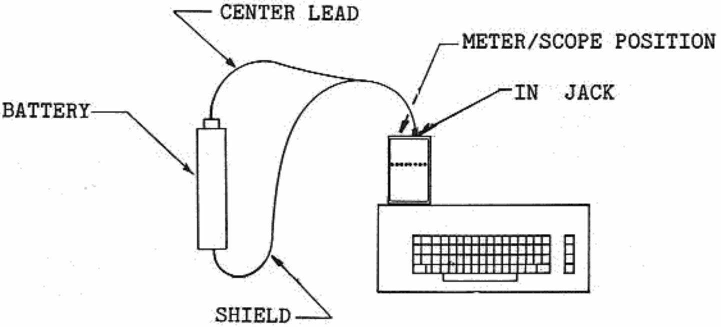
STEP 1> **Reset** your computer, if it has a reset switch, or turn it **OFF** and then back **ON** again. This project requires that R.O.S. **NOT** be running.

STEP 2> Type: **LOAD"MULTIVOLT",8 <return> then RUN <return>**

The screen display represents a D.C. voltmeter. The readings are in both ANALOG (the needle) and DIGITAL (the numbers) form.

STEP 3> Plug the 3 foot gray test cable into the **IN** jack on the back of the interface and move the selector switch to the METER/SCOPE position. (NOTE: Refer to **Project #13** for information on how to prepare the cable.)

STEP 4> Touch the SHIELD (outside cable lead) to the negative end (bottom) of the battery to be tested. Touch the center lead to the positive end (top) of the battery. See the figure below.



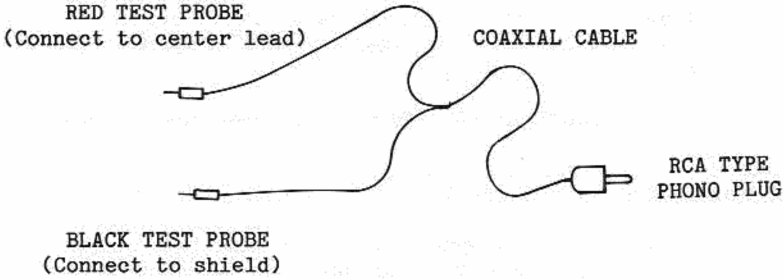
The voltmeter should read slightly above 1.5 volts for a new battery. Anything less than 1 volt means the energy is very low. If the reading is zero, try reversing the test leads.

PROJECT #39

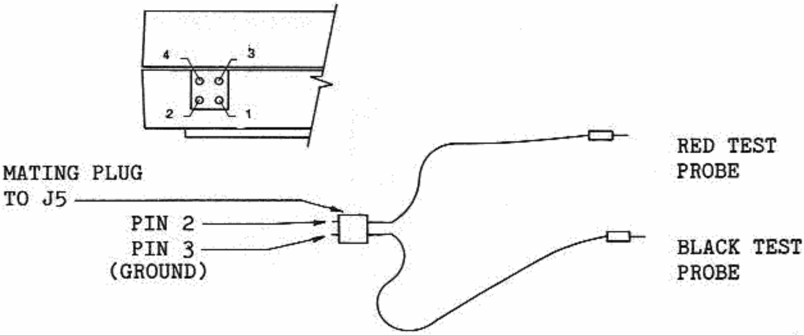
MAKING YOUR OWN VOLTMETER PROBES

DESCRIPTION:
You may wish to make your own test leads (PROBES) for use with the voltmeter and oscilloscope. This project provides two examples of how this can be done. The materials may be purchased at your local electronics store.

INSTRUCTIONS:
In the first example, shown below, the RCA phono plug connects to the **IN** jack in the back of the B100 interface. The selector switch must be set to the METER/SCOPE position.



As an alternative, you may construct a cable which plugs into jack J5 located on the side of the interface.



Information on changing scales to read voltages greater than 3.6 volts is available in the MULTIVOLT section of this manual.

PROJECT #40

USING THE VOLTMETER TO MEASURE RESISTANCE

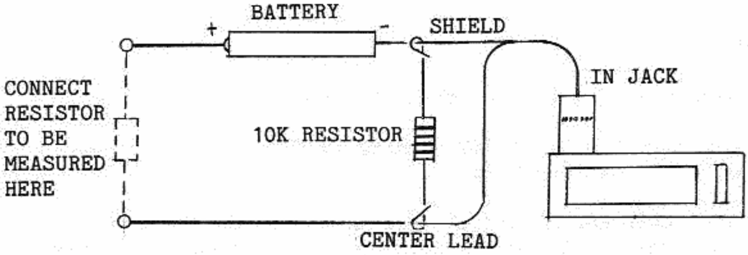
DESCRIPTION:
The MULTIVOLT meter is designed to measure voltage; however, with two external components, it can function as an OHMMETER and measure resistance as well.

To perform this experiment, you will need the following:

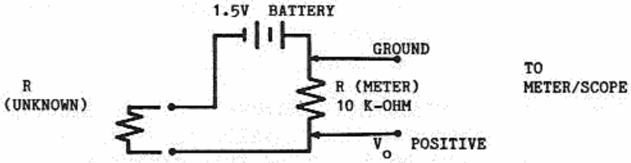
1	AAA, AA, C, or D battery (1.5 volts)
1	10 K-ohm (10000 ohm) resistor
1 (or more)	Resistors of various values
2 feet	Extra wire

INSTRUCTIONS:

STEP 1> Connect the components together as shown below.



The figure below represents the SCHEMATIC diagram of the above circuit.



STEP 2> LOAD R.O.S. then type: **LOAD"OHM METER",8 <return> then RUN <return>.**

STEP 3> Calibrate the meter by touching both leads together (without the resistor to be tested) and pressing any key.

STEP 4> Now place a resistor across the leads. The program will calculate and display the resistance based on the following equation.

$$R\ (\text{unknown}) = 10000 * \frac{(VB - 1)}{Vm}$$

VB = Voltage of the battery (measured during calibration)
Vm = Voltage across the 10K resistor (measured during the test)

PROJECT #41

USING THE MULTISCOPE FOR EXTERNAL MEASUREMENTS

DESCRIPTION:
This project explains how to use the MULTISCOPE digital storage oscilloscope. You will need to be familiar with oscilloscopes in general to fully understand this project.

The MULTISCOPE is designed to read positive D.C. voltages in the range of 0 to 3.6 volts. The range may be easily extended by installing a shunt resistor across pins 1 and 3 of J5. For specific details, see the section on MULTIVOLT.

A 3 foot gray "cable" (signal probe) is included with the workshop and plugs into the **IN** jack on the back of the B100 interface. You may also make your own "signal probe". Refer to project #39 for some ideas. The probe may connect to either of two locations. The 'RCA' type phono jack (IN) on the back of the B100 interface or between pins 2 and 3 (GND) on J5.

INSTRUCTIONS:

STEP 1> Place the selector switch in the METER/SCOPE Position.

STEP 2> Make sure R.O.S. is **NOT** running by turning the computer OFF and ON or by resetting it.

STEP 3> Type: **LOAD"MULTISCOPE",8 <return> then RUN <return>.** The screen will come up looking like a normal oscilloscope.

STEP 4> Connect the "signal probe" to a signal source. If you don't have a variable D.C. voltage of less than +3.6 volts, refer to project #14.

CAUTION: DO NOT ATTEMPT TO MEASURE VOLTAGES ABOVE 5 VOLTS WITHOUT USING SHUNT RESISTORS. YOU SHOULD KNOW WHAT YOUR DOING TO USE THIS DEVICE FOR VOLTAGES ABOVE 12 VOLTS. <u>THIS IS NOT A TOY</u>!!!

The FUNCTION keys control the Time per Division and Volts per division scales as well as other scope features as shown below.

Time/D	F1 UP F2 DOWN
Volt/D	F3 UP F4 DOWN
R. Time Stored	F5 toggles between real time and stored display
Trigger Slope	F7 toggles trigger on or off F8 toggles slope +/-

The **CRSR UP/DOWN** keys control the trigger threshold. Once you press this key the threshold point is displayed as a blue dot on the left-hand side next to the border.

The **CLR/HOME** key clears the trace.

The border color is changed by the **B** key, the trace color is changed by **T** key.

To store a waveform, simply press **'S'**. To stop storage, press **'S'** again.

To "zero" the trace (adjust the zero voltage reference point), use the **(+)** and **(-)** keys. They are auto repeating for your convenience.

Pressing **E**, stops the display from erasing the trace when it plots a new point. Pressing **E** again will cause the erase function to resume, but will not remove previously displayed information. Use **CLR/HOME** to clear the screen.

PROJECT #42

FIRST CAR

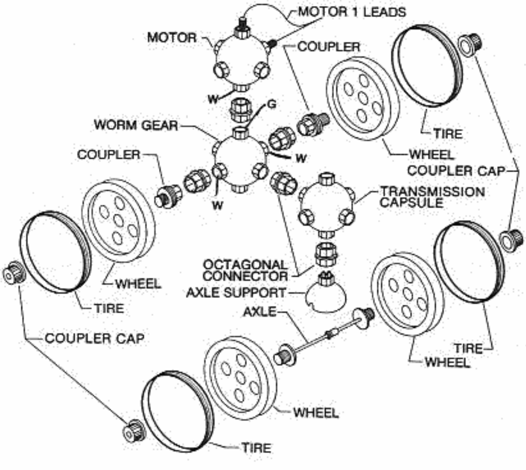
DESCRIPTION:
This project, and several to follow, will serve as examples and give you ideas for designing moving vehicles.

INSTRUCTIONS:

STEP 1> Assemble the project shown below. Connect the motor to the Motor #1 (Red) Leads.

Parts required:

4	Wheels
4	Tires
5	Octagonal Couplers
4	Coupler Caps
1	Axle
1	Axle Support
1	Transmission Capsule
1	Worm Gear
1	Motor
2	Couplers



STEP 2> Make sure R.O.S. is loaded, then Type: **LOAD"FIRST CAR",8 <return> then RUN <return>.**

Make sure the ON/OFF switch is in the **ON** position.

Use the keys 1 thru 8 to increase the speed of the car. Use **'R'** to reverse the car, **'F'** to make it go forward.

When you're finished, LIST the program and try to determine how the program controls the car.

The main loop of the program is between lines 1160 and 1260.

PROJECT #43

"GO ANYWHERE" MOBILE

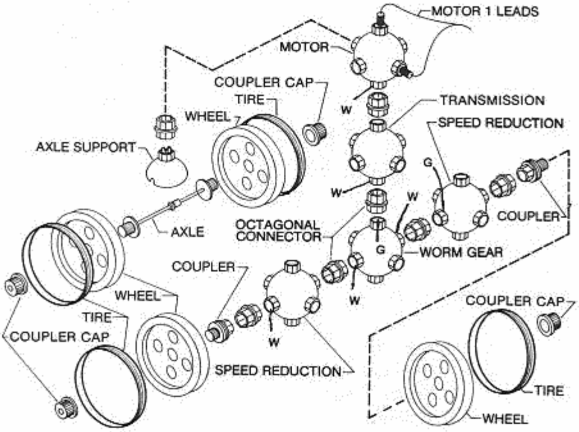
DESCRIPTION:
This vehicle uses two gear capsules to achieve high torque, much like a 4 wheel drive truck in LOW RANGE.

INSTRUCTIONS:

STEP 1> Assemble the project shown below. Connect the motor to the motor #1 leads.

Parts required:

4	Coupler Caps
2	Couplers
7	Octagonal Connectors
2	Speed Reduction
1	Worm Gear
1	Transmission
1	Motor
1	Axle Support
1	Axle
4	Wheels
4	Tires



STEP 2> Make sure R.O.S. is loaded and then type: **LOAD"GO ANYWHERE",8 <return> then RUN <return>.**

This program is controlled by the joystick (port 1 or port 2) or the keyboard according to the following table.

Action Desired	Joystick	Keyboard
Forward	Up	F1
Reverse	Down	F7
Stop	Button	Space
Faster	Right	F
Slower	Left	S

Please list this program to see how it was done. Feel free to change the program and/or modify the vehicle.

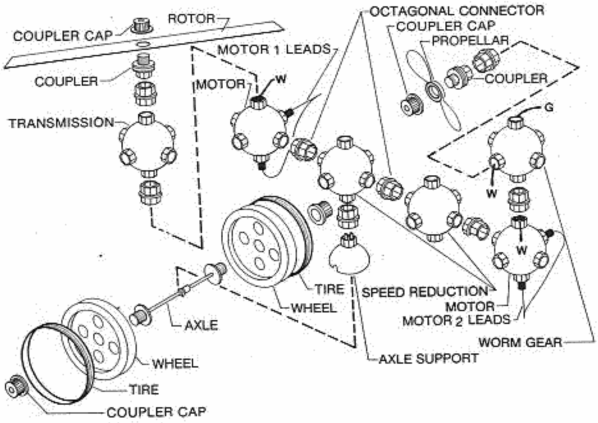
PROJECT #44

CHOPPER

DESCRIPTION:
This project uses two motors. One motor moves the stabilizer and the other motor spins the rotor.

INSTRUCTIONS:

STEP 1> Assemble the project shown below.



Parts required:

2	Motors
8	Octagonal Connectors
1	Worm Gear
2	Speed Reduction
2	Couplers
4	Coupler Caps
1	Axle Support
1	Axle
2	Wheels with Tires
1	Propeller
1	Rotor
1	Transmission

STEP 2> Make sure R.O.S. is loaded and then type: **LOAD"CHOPPER",8 <return> then RUN <return>.**

Be sure to turn up the sound on your T.V. or monitor for this project.

To stop at any time, press any key.

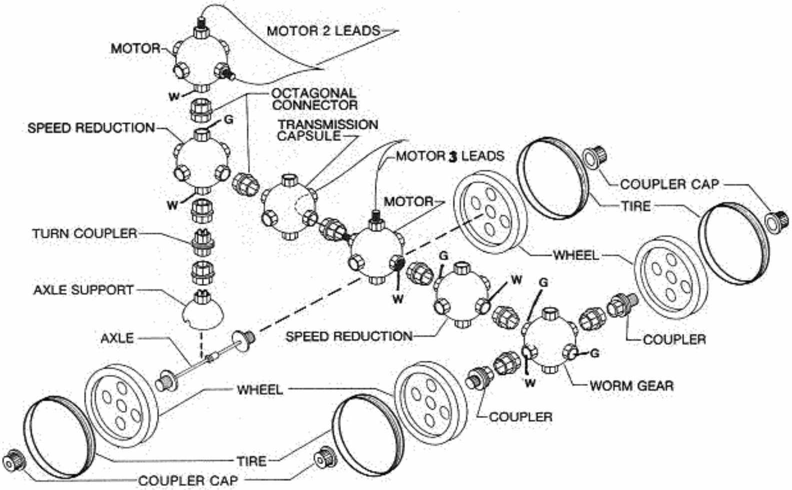
PROJECT #45

STEERING CAR 1

DESCRIPTION:
This car uses a second motor for steering.

INSTRUCTIONS:

STEP 1> Assemble the project as shown below. Be careful in routing the motor #3 wire through the transmission capsule.



Parts required:

4	Coupler Caps
4	Wheels with Tires
2	Couplers
9	Octagonal Connectors
1	Worm Gear
2	Speed Reduction
2	Motors
1	Transmission without Shaft
1	Axle Support
1	Axle

STEP 2: Make sure R.O.S is loaded then type: **LOAD"STEERING CAR1",8 <return> then RUN <return>.**

A joystick is required in port #2.

Joystick	U	=	Forward
	D	=	Reverse
	L	=	Left
	R	=	Right
	B	=	Brake
Keyboard	S	=	Slower
	G	=	Gas

The car moves slowly, but nas a lot of power.

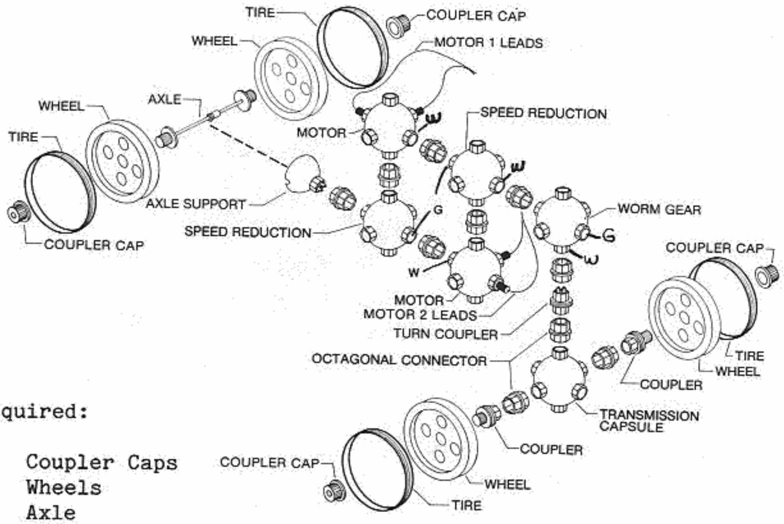
PROJECT #46

STEERING CAR 2

DESCRIPTION:
This car will run faster and turn smoother than the previous car.

INSTRUCTIONS:

STEP 1> Assemble the project as shown below.



Parts required:

4	Coupler Caps
4	Wheels
1	Axle
1	Axle Support
10	Octagonal Connectors
2	Speed Reduction
2	Motors
1	Worm Gear
1	Turn Coupler
2	Couplers
1	Transmission

STEP 2> Make sure R.O.S is loaded and then type: **LOAD"STEERING CAR2",8 <return> then RUN <return>.**

This uses the same joystick and keyboard control as car 1.

Joystick	U	=	Forward
	D	=	Reverse
	L	=	Left
	R	=	Right
	B	=	Brake
Keyboard	S	=	Slower
	G	=	Gas

Look the program over and see what changes you can make to improve performance.

PROJECT #47

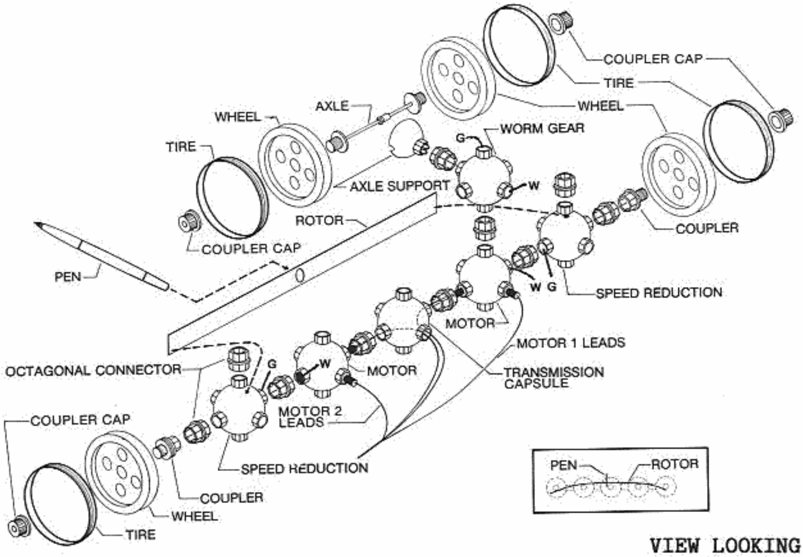
CRAZY PLOTTER

DESCRIPTION:

This project will take some ingenuity to be able to complete. In addition to the standard Multibotics components, you will need a large sheet of paper, a hard flat surface and a felt tip pen. The pen we used was a PILOT SC-UF. It fit perfectly. The rotor is inserted so that the pen will just touch the paper.

INSTRUCTIONS:

STEP 1> Assemble the project as shown below.



VIEW LOOKING AT EDGE OF ROTOR

Parts required:

4	Coupler Caps	1	Worm Gear
4	Wheels	1	Axle Support
2	Couplers	1	Axle
10	Octagonal Connectors	1	Rotor
2	Speed Reduction	1	Pen (user provided)
2	Motors	1	Sheet of Paper (user provided)
1	Transmission (feed motor wires through hole)		

STEP 2> Set a large sheet of paper on a flat surface. Place the CRAZY PLOTTER on the sheet of paper. Adjust the pen so that it makes contact with the surface. If your pen is the wrong diameter, it can be taped in position.

STEP 3> Make sure R.O.S. is loaded, then type: **LOAD"CRAZY PLOTTER",8 <return> the RUN <return>.**

Use the joystick in port 2 to control the direction. The **F1** key will speedup the plotter. The **F7** key will slow it down. The **FIRE** button or **SPACE** bar will temporarily stop it.

See if you can "plot" a tree or write your name!

PROJECT #48

DIGITAL SPEECH PLAYBACK

DESCRIPTION:

As you learned in project #20, computers "store" information as **DIGITAL** information (ONES and ZEROS). Audio signals are in **ANALOG** form. In order for the computer to "store" audio information, it must first be converted to **DIGITAL** form. This conversion process is done on the B100 interface by a method called **DELTA-MODULATION.** For a more complete description of **DELTA-MODULATION,** please see the ELECTRONIC SPEECH section of this manual.

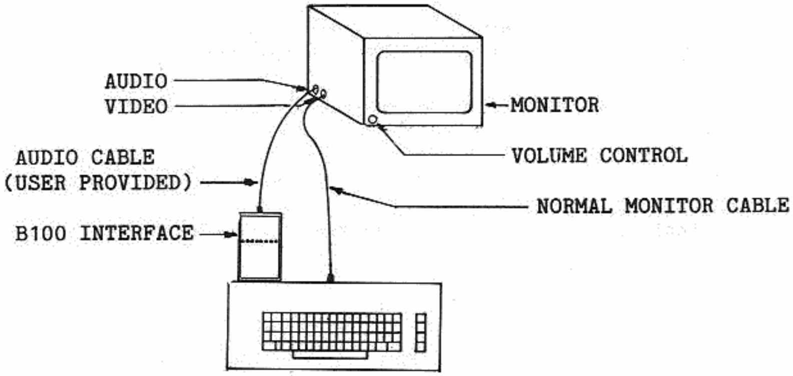
In this project you will play back previously recorded speech that is stored on your disk.

Parts required:

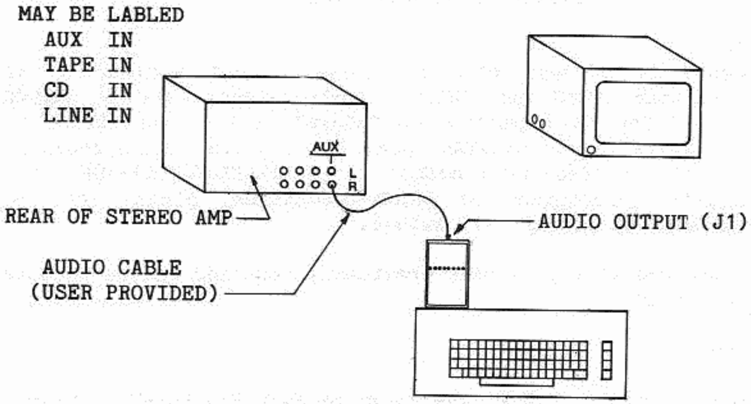
1 Audio Cable with RCA Type Plugs on Each End (user provided)
1 Computer Monitor with Audio Input
or
Hi-Fi Stereo Amplifier with Aux Input

INSTRUCTIONS:

STEP 1> Connect the components together as shown in **either** of the following configurations.



MONITOR WITH AUDIO INPUT



HI-FI STEREO AMPLIFIER WITH AUX INPUT

If you are connecting to anything other than these two devices (such as the audio in on the Commodore 64 computer), the audio output of the B100 interface is approximately 1V P-P.

STEP 2A> Make sure R.O.S. is not running.

STEP 2B> To play back speech, load the program AUDIO into the computer. Type: **LOAD"AUDIO",8,1 <return>.**

STEP 3> Turn on the audio system with the volume set at about mid level.

STEP 4> Put the interface selector switch into the AUDIO position.

STEP 5> Select the LOAD option on the computer. This will take 1-2 minutes.

STEP 6> Select the PLAY option.

The sounds you will hear are much better than anything your computer has provided before! The memory capacity is approximately 27 seconds, which works out to be about 1,800 computer bytes per second of speech (1.8K/SEC).

You can select the PLAY option as many times as you like. Once the audio data is in memory (remember it was loaded from disk), it can be converted to speech at anytime by using the PLAY option.

NOTE: If you are using a stereo amplifier, try adjusting the treble control for a cleaner sound.

PROJECT #49

RECORDING SPEECH/AUDIO

DESCRIPTION:

In the previous project, the speech was already recorded on disk. Now we will try to record speech or sound from a microphone or stereo system. You'll need to have the following parts or components:

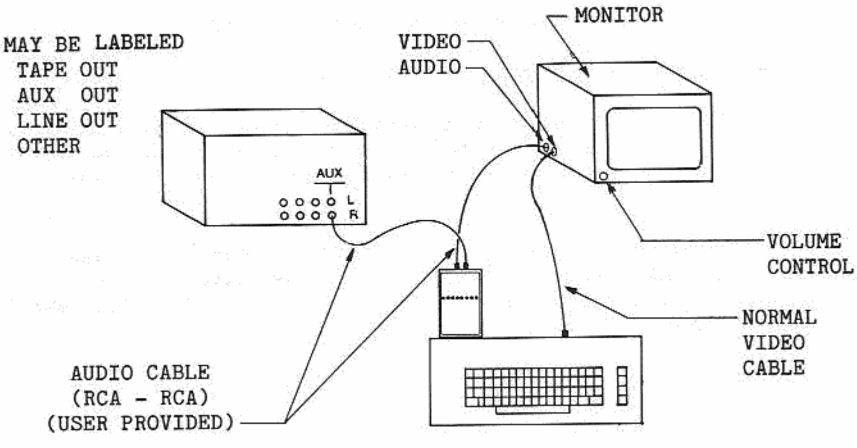
1 Source of Audio (Microphone and preamplifier or stereo system)

1 Source to play back (Monitor with audio input or stereo system)

2 RCA to RCA cables (phono to phono)

INSTRUCTIONS:

STEP 1> Connect the components together as shown:



RECORDING CONFIGURATION

NOTE: Audio IN (J2) requires approximately 1 volt P-P. It may come from any audio source which has adequate level, such as an audio pre-amp, earphone out of cassette recorder, etc.

STEP 2> Adjust the audio controls for **"average"** sound levels.

STEP 3> Turn on the computer. LED's 0-7 on the interface should light. LED 7 may be OFF. Place the selector switch in the AUDIO position. Any sound at the input will now pass through the B100 interface. Adjust the audio input and output for normal levels of sound. The sound coming out of the monitor is digitized. It is being automatically digitized by the interface. The sound source can be speech, music, or sound effects.

STEP 4> Once the level is set, load the AUDIO software. Type: **LOAD"AUDIO",8,1.**

STEP 5> Turn down the monitor (output) sound level to avoid feedback. Select the RECORD option from the menu.

Once this option is selected, the screen will go blank and the recording process will start. The speech digitizer will record for about 27 seconds. When it is finished, the screen will return with the menu.

STEP 6> At this point you may listen to your computer by turning down the input level and by turning up the monitor level. Then select the PLAYBACK option. Listen to the "recording" as many times as you wish until the computer is turned off.

NOTE: If there is a problem, repeat steps 1 thru 5. Make sure the selector switch is in the AUDIO position and all cables are connected correctly.

If you would like to experiment further with digitized audio and add these sounds to your own programs, you might like to purchase our S100 SPEECH DIGITIZATION ENHANCEMENT PACKAGE.

PROJECT #50

SAVING DIGITIZED SPEECH TO DISK

DESCRIPTION:
This project will show you how to save your digital recordings by saving them to a floppy disk.

INSTRUCTIONS:

Before proceeding, connect the B100 interface to both input and output audio sources as shown in project #49 on RECORDING SPEECH. In order to save recorded audio, you **must have a separate disk.** The disk **must have at least 200 blocks free**. You can determine the number of blocks free by listing the disk directory.

If you are using a new disk, it must first be formatted. **DO NOT USE THE DISK THAT COMES WITH THE ROBOTIC WORKSHOP**.

WHEN YOU FORMAT A DISK, EVERYTHING ON THE DISK WILL BE ERASED. SO BE CAREFUL!!!

CAUTION: REMOVE THE ROBOTIC WORKSHOP PROGRAM DISK !!!!
and insert a blank disk.

To format a disk, type: **OPEN15,8,15,"NO:MULTIBOT,01" <return>**

The process will take about 2 minutes to complete. After the disk light goes out, type: **CLOSE 15 <return>**

Now LOAD the AUDIO program as you have done before, and record some audio.

Play back the audio. When it is ready to save, select the **SAVE** option from the menu. This will take about 2-1/4 minutes, so be patient.

Each disk will hold one audio file. An audio file may be loaded anytime that the AUDIO program is running.

If your sound system has a tone control or treble control, try adjusting it for the best sound possible.

V

ALL ABOUT R.O.S.

The **Robot Operating System (R.O.S.)** was developed by Access Software especially for the Robotic Workshops. R.O.S. adds 25 special commands to the BASIC language to enable you to access all the power of the Multibotics B100 interface. Since these commands are added to BASIC, you can use normal BASIC commands with R.O.S. commands. This will make it very easy to get powerful programs up and running quickly without having to learn an entirely new language.

LOADING R.O.S.

To load R.O.S. on the Commodore 64, simply type the following:

LOAD"ROS",8 <return>

When the computer responds with 'READY', type: **RUN** <return>

After a short time, the R.O.S. screen will come up:

```
ROBOT OPERATING SYSTEM ENABLED!

READY
```

At this time all BASIC and R.O.S. commands may be used.

COMMANDS

Below is a list of the R.O.S. commands. We suggest that you LIST and study the programs in the project section to get an understanding of how the commands are used.

- ALIGN
- BEEP
- FLIP
- HELP
- JOY
- *MASTR
- *MCNT
- *MCYCF
- *MCYCN
- MOTR
- MRANGE
- *MSET
- *MSRSET
- RANGE
- *SCCNT
- SCN
- SDIR
- SDUR
- SENSR
- SPCNT
- *SSET
- STME
- TMER
- TMESET
- VOLT

The commands which are marked with an asterisk (*) are <u>advanced</u> commands. These allow very precise timing of events and very precise motor control. (Accuracies of better than 0.001 seconds are easily possible.) The advanced commands will be fully described in the ADVANCED section of this manual.

ALIGN Allows for the easy alignment of the infrared (I/R) sensors. Before any project using sensors can be completed, they must be aligned - that is face each other properly. The '**ALIGN**' command will help insure that this is done properly.

```
SYNTAX:   ALIGN VAR1,VAR2
          VAR1=  Input sensor number   (1-8)
          VAR2=  Output sensor number  (1-8)
```

Normal usage with the I/R sensor is:

ALIGN 8,7

This command may only be used in direct mode and not under program control.

The screen will be cleared and the text on the screen will show which sensor was requested as the input and which as the output. When the sensors are misaligned, text on the screen around the word **MISALIGNED** will be in reverse field and a low tone will sound. When the sensors are aligned, text on the screen around the word **ALIGNED** will be in reverse field and a higher pitch tone will sound. When the alignment process is complete, pressing the **RUN/STOP** key **will return you to R.O.S.**

BEEP Used to generate a tone through the normal computer audio system.

SYNTAX: BEEP

It may be used at any time to signal the end of an event or to get someone's attention.

FLIP Is one way to reverse the direction of the motor.

SYNTAX: FLIP VAR VAR= Motor Number (1,2,3)

This command may be used to reverse the current direction of the specified motor. Sometimes when attaching the leads to a motor, the leads may be reversed. For example, if the motor was programmed to run forward, it would actually be running in reverse. When the **FLIP** command is used, it will affect all motor direction operations until that motor is shut off or until the **FLIP** command is again invoked to re-reverse the direction of the motor.

Example: FLIP 3 <return>. When used in direct mode, this statement would reverse the direction of motor #3.

```
10 MOTR 1,15           :REM  Turn on motor #1
20 FLIP 1              :REM  Reverse the direction
                             of motor #1
30 FOR B=1 TO 50:NEXT  :REM  Pause
40 GOTO20              :REM  Back to line 20 and
                             reverse again.
```

HELP Will display a list of the R.O.S. commands on the screen. **May only be used in direct mode.**

SYNTAX: HELP

Because so many new R.O.S. commands have been added to the already large number of existing BASIC commands, it appears helpful to have an easily available list of the R.O.S. commands while programming. This list can be displayed on the screen by typing the **HELP** command in direct mode.

Example: HELP <return>.

JOY Reads the status of the joystick port into a variable.

```
SYNTAX:   JOY VAR1,VAR2       VAR1-  Port# (1 or 2)
                              VAR2-  Variable to hold status
```

Since so much of Multibotics is controllable with the joysticks, this command is provided to allow easier access to the status of the joysticks. The first variable must be either a 1 or 2 to specify either joystick #1 (front) or joystick #2 (rear). The second variable will then be assigned the value (or status) of the specified joystick.

Example:

```
10 JOY 2,A          :REM  Put joystick #2 in variable 'A'
20 F=(A and 16)     :REM  If the fire button is pressed,
                          then F = 16.
30 U=(A and 1)      :REM  If the joystick is pushed
                          forward, then U = 1.
40 D=(A and 2)      :REM  If the joystick is pulled back,
                          then D = 2.
50 L=(A and 4)      :REM  If the joystick is pushed left,
                          then L = 4.
60 R=(A and 8)      :REM  If the joystick is pushed right,
                          then R = 8.
```

See the PROJECTS section of this manual.

NOTE TO ADVANCED PROGRAMMERS: R.O.S. inverts and masks the actual port read. The JOY command returns the upper 3 bits = 0 and the appropriate lower bits = 1 if the stick is active.

MASTR* (Master) Used to set up precise conditions that are used by the interrupt system. See ADVANCED PROGRAMMING section.

MCNT* (Motor Count) Assigns the number of interrupt cycles that have occurred since a motor was turned on to a variable. See ADVANCED PROGRAMMING section.

MCYCF* (Motor Cycles Off) Assigns the number of OFF cycles being generated for a particular motor to a variable. See ADVANCED PROGRAMMING section.

MCYCN* (Motor Cycles On) Assigns the number of ON cycles being generated for a particular motor to a variable. See ADVANCED PROGRAMMING section.

MOTR (Motor) Is the easy way to totally control the motors. There are three variations of the Syntax.

SYNTAX: **MOTR 0** This turns off all motors.

SYNTAX: **MOTR VAR1,VAR2**

VAR1= Motor number 1-3.

VAR2= Speed.
Negative numbers are reverse, positive numbers are forward. The default maximum speed is 30; however, any number up to ± 255 may be used. The RANGE command will allow numbers greater than 30 to have an effect.

Example:

```
10 MOTR 1,15      :REM Motor#1, 1/2 speed, forward.
20 MOTR 3,-10     :REM Motor#3, 1/3 speed, reverse.
```

Example:

```
10 FOR I = 0 TO 30
20 MOTR 1,I
30 Next I
```

Keeps increasing motor speed in a FOR/NEXT LOOP.

Example:

```
10  INPUT X       :REM Get motor number
20  INPUT Y       :REM Get motor speed
30  MOTR X,Y      :REM Turns on motor X
                       at speed Y
```

SYNTAX: **MOTR VAR1,VAR2,VAR3,VAR4**
(Advanced Feature)

VAR1= Motor # (1-3)

VAR2= Motor direction
 1 = forward
 0 = stop
-1 = reverse

VAR3= Number of ON cycles for the motor.
(Must be between 0-255).
See PULSE WIDTH MODULATION (PWM) project.

VAR4= Number of OFF cycles for the motor.
(Must be between 0-255).

MRANGE (Motor Range) This is used to retrieve the range setting (total of ON + OFF cycles) for a particular motor.

SYNTAX: **MRANGE VAR1,VAR2**

VAR1= Motor # (1-3)

VAR2= The variable which will receive the data.

Example: To find out the range setting for motor #1, you could do the following:

```
10   MRANGE  1,A
20   PRINT A
```

See '**RANGE**' command.

***MSET** (Motor Setting) Allows the program to see how a particular motor is "set up". See ADVANCED PROGRAMMING section.

***MSRSET** (Master Setting) Allows the examination of how a particular Master Byte is set. See ADVANCED PROGRAMMING section.

RANGE Gives you complete control of the motor speed process. RANGE is the total number of ON and OFF motor cycles. (See project on PULSE WIDTH MODULATION.) The default selection by R.O.S. is 30. Any value between 2 and 255 is valid. The maximum speed of a motor is equal to its range setting. A large range allows more motor speeds, but may cause the motor to pulse or become jerky. The lower the range, the smoother the motor will run; but it will have fewer speed increments. Try this command and experiment with it. The optimum range setting will vary from project to project.

```
Example:  20 RANGE 1,20     :REM sets maximum range to 20
```

***SCCNT** (Sensor Current Count) Assigns a variable to the current number of transitions of the I/R beam. This command is used to count objects, people, etc. See ADVANCED PROGRAMMING section.

SCN (Screen) Turns the video screen **ON** or **OFF.** It is used when precise timing is needed for a project. The screen "steals" clock cycles from the processor and can make precise timing difficult in advanced projects. Normally the screen would be left **ON.**

SYNTAX: **SCN VAR**
VAR= 1 turns the screen **ON**
0 turns the screen **OFF**

During the normal operation of the C64, the VIC-II video chip "steals" some of the processing time from the 6510 microprocessor. This loss of machine cycles normally has no bearing on a BASIC program, but with the sophistication of R.O.S. it is possible to control motor and sensor activities at any extremely time-critical level. So, for very time-critical motor/sensor operations, the **SCN** command allows the timing to proceed uninterrupted by the VIC-II video chip.

```
Example:  SCN 0 Turns the screen OFF.
          SCN 1 Turns the screen ON.
```

SDIR (Sensor Direction) Assigns a variable to the sensor direction. Sensor direction indicates if the I/R sensor beam is broken or unbroken. (That is - Did someone walk in front of a sensor?)

SYNTAX: **SDIR VAR1,VAR2**

VAR1= Input sensor number (1-8). Usually 8.

VAR2= Any variable for the result. VAR2 will be a "0" if the sensor beam is broken. It will be a "1" if unbroken.

Example:

```
100  SDIR 8,B               :REM Put status in B
110  IF B=1 THEN 100        :REM Wait for beam to
                                 be broken.
120  MOTR1,20               :REM Turn on motor #1
                                 if beam is broken.
```

SDUR (Sensor Duration) Assigns the number of cycles that occured while the beam was interrupted to a variable. Useful for measuring speed. See project: MEASURING SPEED OF BASEBALL BAT.

Example:

```
100  SCCNT 8,A              :REM Count Beam
                                 Transitions
110  If A<2 THEN 100        :REM If less than 2 transitions,
                                 then object hasn't passed.
120  SDUR 8,T               :REM Get the number of interrupt
                                 cycles that occured while
                                 beam was off.
```

SENSR (Sensor) Used to set up sensors. This command has 3 forms.

SYNTAX: **SENSR 0** Turns off all sensors.

SYNTAX: **SENSR VAR1,VAR2**

VAR1= Sensor number 1-8 (Usually 7 or 8)

VAR2= If 1, then sets up for input sensor.
If 2, then sets up for output sensor.

Example:

```
20 SENSR 7,2                :REM sets sensor 7 for output
30 SENSR 8,1                :REM sets sensor 8 for input
```

SYNTAX: **SENSR VAR1,VAR2,VAR3**

VAR1= Selects sensor 1-8 (Usually 7 or 8)

VAR2= Sets up for 1 = input, 2 = output

VAR3= If input sensor, then sets up for timed mode. The VAR3 sets the counting interval in cycles. R.O.S. will count how many times the beam made a transition during that interval. The variable can be between 1 and 16 million ($1-256^3$). This is useful for measuring RPM easily and accurately.
See the project on MEASURING RPM.

SPCNT (Sensor Previous Count) Sets a variable to the number of transitions which occured during the timing interval, if sensor is in timed mode. Useful for measuring RPM, etc. See project: MEASURING RPM.

SYNTAX: SPCNT VAR1, VAR2

```
          VAR1=    Sensor number 1-8  (Almost always 8)
          VAR2=    Variable where count is to be stored.
```

Example:

```
100 SPCNT 8,R
110 PRINT R
```

This will print the maximum count reached by the input sensor during the previous timing interval.

***SSET** (Sensor Setting) Transfers the current setting of the sensor to a variable. See ADVANCED PROGRAMMING section.

STME (Sensor Time) Transfers the current setting of the sensor timing interval to a variable.

SYNTAX: STME VAR1,VAR2

```
          VAR1=  Sensor number (1-8)
          VAR2=  Destination variable
```

Example:

```
100 STME 8,A(3)                 :REM Transfers time setting
                                  to array variable A(3)
```

See 'SENSOR' command

TMER (Timer) Sets up the <u>interrupt timing interval</u> for R.O.S. All timing and counting by R.O.S. is done in terms of cycles. The exact duration of the cycle is determined by **TMER**.

SYNTAX: TMER VAR1

```
          VAR1=   Can be any number between 1 and 65,536.
```

The default timer setting is 2760. This equates to 3 milliseconds (3/1000 second).

Values much under 2 milliseconds (2760 x 2/3) will start to slow BASIC down. However, these low values are acceptable, and cycle times less than 1 millisecond work well for precise timing applications.

TMESET (Timer Setting) Returns the value of the current <u>interrupt timing interval</u> of R.O.S.

SYNTAX: TMESET,VAR1

Example:

```
10 TMESET,X
20 PRINTX
```

X will contain the value of the timer setting.

The default value is 2760 for a cycle time of 3 MS.
See **TMER** command.

VOLT (Voltmeter) Returns the current relative voltmeter reading to a variable. This is a very powerful command and may be used to implement automatic testing equipment, antenna positioners, temperature measurement, etc.

SYNTAX: **VOLT,VAR1**

VAR1= Is the variable which will contain the voltmeter reading. The value of VAR1 will be between (0 and 127).

Example:

```
100 VOLT,A
120 PRINT A
```

'A' will contain the value of the voltmeter - analog to digital convertor.

The reading from volt will not be an actual voltage, but a relative value. To convert this to an approximate voltage, use this procedure.

```
100 VOLT, A
110 V = A * .02698
120 PRINT V
```

"V" will be very close to the actual voltage.

To be even closer, V = (A-.5) * .02698 + .19 (in line 110)

See section on MULTIVOLT to ensure that the voltmeter is hooked up correctly.

ADVANCED VOLTMETER TECHNIQUE

For more precise measurements, calibrate the meter from a known standard for each value of VOLT (0-127), and use a look up table.

NOTES ABOUT R.O.S.

A **R.O.S.** command **cannot** follow a BASIC **THEN** statement.

For example:

```
     100   IF A = 5 THEN MOTRO      WILL NOT WORK!!!
```

You may do one of the following:

```
     100   IF A = 5 THEN X = X: MOTRO
```

Where 'X = X' is a dummy operation

OR

```
     100   IF A = 5 THEN 200
     110   GOTO 210
     200   MOTRO
     210   REM   CONTINUE
```

EITHER WILL WORK!

VI

ELECTRONIC SPEECH

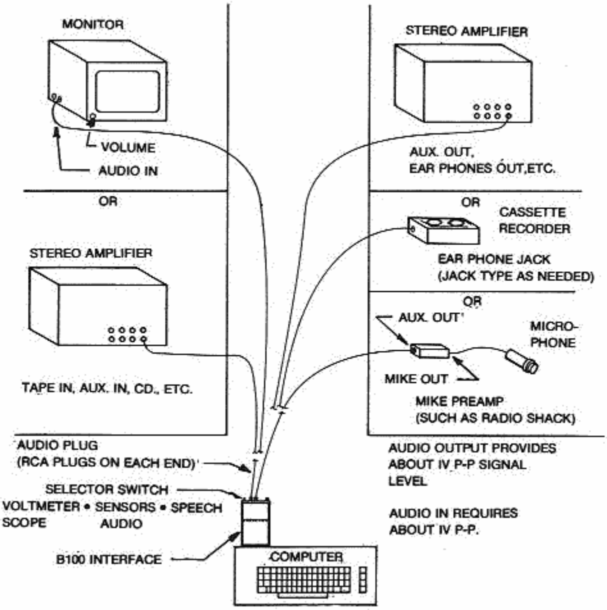
The ELECTRONIC SPEECH or audio portion of the B100 interface performs two separate functions:

1. It digitizes any audio frequency signal - speech, music, sound effects, etc.

2. It plays back that audio.

Approximately 27 seconds of speech can be stored in the Commodore 64's memory. Once stored, it may be saved to disk for play back at any time.

The sketch below shows how the unit should be connected to your system.



Testing the levels for proper adjustment is very easy.

Turn the computer on. LED's 0-6 should be on. (Bit 7 may be on.) Set the selector switch to the AUDIO position.

Turn on the input signal. When audio is present, the output signal will be on. Now adjust levels for normal sound. You are listening to digitized audio.

The unit is ready to go! Just load the AUDIO program into your computer. Type: **LOAD "AUDIO",8 <return>, then RUN <return>.**

Follow the screen directions for easy menu selection.

ADVANCED AUDIO

The speech/audio system uses a technique of digitization called DELTA MODULATION. Rather than digitize an 8 bit sample as most A/D techniques do, the Delta Modulation system takes a 1 bit sample on the difference between the last sample and the present sample. This 1 bit sample is taken at approximately a 14khz rate (one sample every 72 microseconds). This sample rate results in speech data being stored at the rate of about 1.8 kilobytes/second.

The resultant playback maintains full fidelity of the original. There is a small bit of quantization noise in playback mode. This sounds like a low hiss. The hiss can be significantly reduced by adjusting the tone or treble control. The quantization noise could be almost completely removed by using a higher sample rate. Unfortunately, the speed of the computer limits sampling rates to those defined above.

VII

MULTIVOLT

The B100 interface is designed with a true D.C. Voltmeter built in. This meter can be used for any electronic voltmeter application within its specifications. The meter, without shunts, can read voltages between 0 and +3.6 volts D.C.

STAND ALONE VOLTMETER

To use the MULTIVOLT function:

1. Set selector switch to METER/SCOPE position.

2. Connect a test lead to the voltmeter input jack (J2). (The center conductor is (+) or positive lead.)

3. Make sure R.O.S. is **NOT** running.

4. Load MULTIVOLT by typing: **LOAD "MULTIVOLT",8 <return>, then RUN <return>.**

You are now ready to measure any voltage. To adjust the scale factor of the display, use the 'X'. It covers the ranges of X.1, X.2, X.5, X1, X2, X5, X10, X20.

TO READ VOLTAGE FROM A PROGRAM

The R.O.S. (Robotic Operating System) has a built in command for reading a voltage and returning it as a variable in your basic program. This is a very powerful feature of the B100 interface.

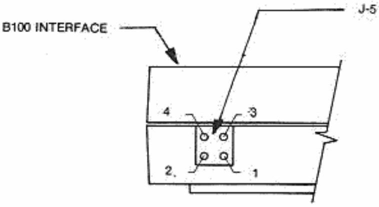
The command is structured as: **VOLT,VAR.** VAR may be any valid basic variable, or array. The variable will return a number between 0 and 127 ($00-$7F). To convert it to a real voltage reading rather than a reference voltage, use the following formula:

$$\text{Voltage} = (\text{VAR} - .5) * .02698 + .19$$

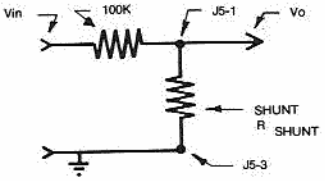
This can be used for such projects as antenna positioning, feedback, position sensing, production testing, temperature measurement, etc. We use this command at the factory to test new B100's.

CHANGING SCALE: ADVANCED INFORMATION

The range may be changed by adding a shunt resistor across pins 1 and 3 of J5.



The input circuit looks like this:



To scale the voltage V_0 the shunt R_s can be installed.

$$V_o = V_{in} * \frac{R_s}{R_s + 100K}$$

To find the value of the shunt for a given voltage ratio:

$$R_s = \frac{V_o}{V_{in} - V_o} * 100K$$

or if the ratio of V_o to V_{in} is A then:

$$R_s = \frac{A}{1 - A} * 100K$$

As an example, let's change the full scale from 3.6 volts to 36V. The ratio is $\frac{3.6}{36} = .1$

$$R_s = \frac{.1}{1 - .1} * 100K$$

$$R_s = 11.11 \text{ K OHMS}$$

This shunt resistor should be connected between pins 1 and 3 of J5.

Remember to multiply the voltage readings by 10.

This same procedure is used to scale the voltage input for the oscilloscope.

TYPE OF CONVERSION

The B100 interface converts the analog voltage to a digital reading by using the successive approximation method. The computer does most of the work. A complete conversion for a voltage takes about 196 clock cycles on the Commodore 64, that's about 188 micro-seconds (.000188 seconds).

The digital word returned is 7 bits wide ($00-$7F).

MULTISCOPE

The MULTISCOPE portion of the B100 interface is a very versatile piece of test equipment. It functions as a true digital real-time or storage oscilloscope.

If you are not familiar with oscilloscopes, it might help to have a friend who is knowledgable explain them in more detail.

To use the MULTISCOPE:

1. Place the selector switch in the METER/SCOPE position.

2. Load the MULTISCOPE software **LOAD "MULTISCOPE",8 <return>, then RUN <return>.**

Lets now go through the options and functions of MULTISCOPE. Please refer to the sketch below.

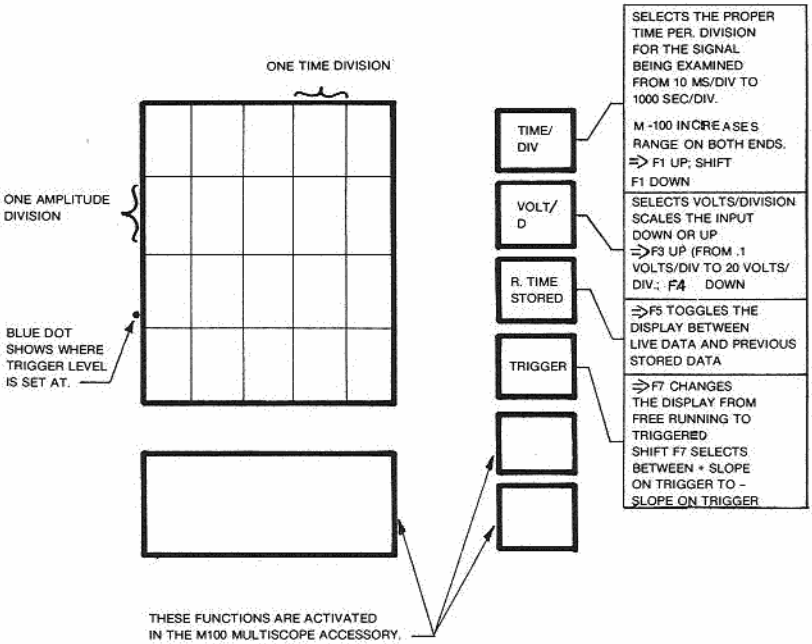

ONE TIME DIVISION
ONE AMPLITUDE DIVISION
BLUE DOT SHOWS WHERE TRIGGER LEVEL IS SET AT.
TIME/ DIV
VOLT/ D
R. TIME STORED
TRIGGER
SELECTS THE PROPER TIME PER. DIVISION FOR THE SIGNAL BEING EXAMINED FROM 10 MS/DIV TO 1000 SEC/DIV.
M -100 INCREASES RANGE ON BOTH ENDS.
=> F1 UP; SHIFT F1 DOWN
SELECTS VOLTS/DIVISION SCALES THE INPUT DOWN OR UP
=>F3 UP (FROM .1 VOLTS/DIV TO 20 VOLTS/ DIV.; F4 DOWN
=>F5 TOGGLES THE DISPLAY BETWEEN LIVE DATA AND PREVIOUS STORED DATA
=>F7 CHANGES THE DISPLAY FROM FREE RUNNING TO TRIGGERED SHIFT F7 SELECTS BETWEEN + SLOPE ON TRIGGER TO - SLOPE ON TRIGGER
THESE FUNCTIONS ARE ACTIVATED IN THE M100 MULTISCOPE ACCESSORY.

TABLE OF KEY FUNCTIONS

F1	Increases time/division	
F2	Decreases time/division	
F3	Increases volts/division	
F4	Decreases volts/division	
F5	Toggles between real-time and stored waveforms	
F7	Toggles trigger ON or OFF	
F8	Toggles trigger slope (+) or (-)	
CLR/HOME	Erases trace	
T	Changes trace color	
B	Changes border color	
S	Starts storage When storage starts, the screen blanks to slightly increase timing accuracy and speed. Pressing 'S' a second time will end the storage process.	
E	Stops the display from erasing the trace when it plots a new point. Pressing 'E' again will re-engage the erase feature, but will not remove previously plotted data. Use CLR/HOME to clear the screen.	
+	Moves the trace up	- Used for centering.
-	Moves the trace down	

The V. Diff, T. Diff, Cursor 1 and Cursor 2 functions are not implemented in the workshops. The M100 MULTISCOPE ENHANCEMENT MODULE will implement these features, add increased time/div ranges, add faster speed, save stored data to the disk, display stored and real-time simultaneously, plus many other exciting features.

Several projects are included in the projects section to help you become better acquainted with the MULTISCOPE.

IX

ADVANCED PROGRAMING

The Robotic Operating System (R.O.S.), as you have noticed by now, is very sophisticed and flexible. There are many unique and clever ways to do things. The powerful R.O.S. commands allow you to accomplish a great deal in very little programming space. Most programs in the project section are less than 4 blocks long!

We suggest that you experiment with the commands to get the most from them.

Below is a list of advanced commands:

MASTR
MCNT
MCYCF
MCYCN
MSET
MSRSET
SCENT
SSET

Remember that the entire system is based on the fact that R.O.S. interrupts the processor periodically to accomplish what ever task needs to be done. The time interval between interrupts can be set by the user with the **TMER** command. The default is 2760 which corresponds to 3 milliseconds on a 60Hz C-64. We call this time interval a **CYCLE** and most discussion will be in terms of cycles rather than seconds, milliseconds, etc.

MASTR (Master Byte) Is the the most powerful and most complicated command in R.O.S.

The purpose of **MASTR** is to watch over events and to perform functions with machine language speed while you are in BASIC. For example, **MASTR** can be set up to watch for a certain number of counts from the I/R sensors, and when that event happens, turn on a motor at a particular speed. This will be accomplished within one time cycle (default is 3 milliseconds) regardless of where the BASIC program is executing code!

There are 16 **MASTR** bytes, so up to 16 different events can be watched and controlled with extreme precision!

There are 3 forms of **SYNTAX** for this command depending on the action being watched.

1. Self-Timed

'Self-Timing' is just counting the number of interrupt cycles before an action is to take place.

SYNTAX: **MASTR VAR1,VAR2,VAR3,VAR4,VAR5,VAR6**

VAR1= Number of this master byte 1-16.
'0' turns off all masterbytes.
VAR2= 128 for self timing mode.
VAR3= Number of cycles to watch
1-16.7 million (1 thru 256^3-1).
VAR4= Motor status byte of motor to be controlled plus the motor number.
See MSET.
VAR5= Motor **ON** cycles 0-255.
VAR6= Motor **OFF** cycles 0-255.

As an example, consider a **MASTR** byte in the 'Self-Timing' mode. After 3333 timing cycles (about 10 seconds), we want to turn on motor #1 at 10 **ON** cycles and 10 **OFF** cycles (half-speed).

Example:

```
5    MOTRO
10   MASTR 1,128,3333,177,10,10
20   X=X+1
30   PRINT X
40   GOTO 20
```

This program is on disk as MASTER TIMED.

When the program is executed, line 10 will set up the **MASTR** byte. The program will just count while not doing anything to affect the motors. After 10 seconds, the motor will come on!

2. Sensor Controlled

'Sensor Controlled' is very similar to 'Self-Timed' except that a sensor is being watched.

SYTNAX: **MASTR VAR1,VAR2,VAR3,VAR4,VAR5,VAR6**

VAR1- Number of this master byte
VAR2- Number of sensor 0-7
+
+0 => watch transitions
+40=> watch current count
+64=> watch previous count
+96=> watch current timing count

```
VAR3=  Number of counts to watch for.
VAR4=  Motor/Sensor status byte
       + = Motor, - = Sensor
           +
       Motor or sensor number
VAR5=   If VAR4 is sensor then sensor timing
       value (1 thru 256^3-1)
           or
VAR5=  Motor ON cycles (0-255)
VAR6=  Motor OFF cycles (0-255)
```

As an example, set up the I/R sensors and watch the current count. When 10 transitions (5 objects) pass through the sensors, then we will turn on motor #1 at half-speed.

```
Example:  100 SENSR 7,2
          110 SENSR 8,1  REM set up sensors
          120 MASTR 1,40,10,177,10,10
```

This program is on disk as MASTER SENSOR.
Run the program, then interrupt the sensors. When the proper number of counts has occurred, the motor will come on, regardless of what else the computer is doing!

3. Motor Controlled

This is exactly the same as 'Sensor Controlled', except that

```
VAR2=  Motor number (0-2)
VAR3= Number of counts that motor is on (MCNT)
```

We hope you can see how powerful this command is. It is providing machine language speed with BASIC flexibility.

MCNT (Motor Count) Assigns the number of cycles that have occurred since the motor was turned on to a variable.

SYNTAX: **MCNT VAR1,VAR2**

```
VAR1=  Motor number (1-3)
VAR2=  Destination variable
```

Example:

```
10 REM - To determine how long a motor has been
            on
20 X=1
30 MCNTX,TME   :REM TME contains number of cycles
                    the motor has been on
40 PRINT TME * 3/1000
```

This program will print the time that a motor has been on in seconds. It assumes that TMER has not been changed and is still at 3 milliseconds per cycle.

The following program will turn on a motor for 10 seconds, then turn it off.

Example:

```
5     MOTR1,0     :REM  Initialize MCNT
10    MOTR1,30
20    MCNT1,A
30    IF A * 3/1000 < 10 THEN 20
40    MOTR1,0
```

MCYCF (Motor Cycles Off) This sends the current setting for the number of **OFF** cycles to a variable.

SYNTAX: **MCYCF VAR1,VAR2**

```
VAR1=  Motor number (1-3)
VAR2=  Destination variable
```

Example:

```
10    RANGE1,30
20    MOTR1,12
30    MCYCF1,B
40    PRINT B
```

'B' will contain the current number of **OFF** cycles set by the motor speed command. This example will show B to be equal to 18 (30-12).

The number of **ON** cycles would be 12.

See **MCYCN.**

MCYCN (Motor Cycles On) Returns the current setting for number of **ON** cycles.

SYNTAX: **MCYCN VAR1,VAR2**

VAR1- Motor number (1-3)
VAR2- Destination variable

Example:

```
10 RANGE 2,50
20 RANGE 1,20
30 MOTR 2,5
40 MOTR 1,15
50 FOR I= 1 TO 2
60 MCYCN I, A(I)
70 PRINT A(I)
80 NEXT I
```

This will print a 15 for motor #1 and a 5 for motor #2.

See **MCYCF.**

MSET (Motor Setting) Returns current motor setting (status) to a variable.

SYNTAX: **MSET VAR1,VAR2**

VAR1- Motor number (1-3)
VAR2- Destination variable

VAR2 will contain the following information:

Bit 7	l= Motor active 0= Motor inactive
Bit 6	1= Continuous mode 0= Timed mode
Bit 5	l= Motor up to speed 0= Motor not up to speed
Bit 4	l= Forward 0= Reverse
Bit 3	l= Motor currently on 0= Motor currently off
Bit 2	Not assigned
Bit 1	Not assigned
Bit 0	Not assigned

MSRSET (Master Setting) Returns the current master status byte setting to a variable.

SYNTAX: **MSRSET VAR1,VAR2**

VAR1- Points to the proper master byte (1-16)
VAR2- Contains status information as follows:

Bit 7	1 = active, 0 = inactive
Bit 6 Bit 5	00 = self-timing, 01 = watch motor 10 = watch sensor, 11 = illegal
Bit 4 Bit 3	00 = watch sensor for transition 01 = watch " for current count 10 = watch " for previous count 11 = watch " for current time
Bit 2 Bit 1 Bit 0	Sensor or motor number

SCCNT (Sensor Current Count) Assigns the total number of transitions which have occurred at the input sensor.

SYNTAX: **SCCNT VAR1,VAR2**

VAR1- Sensor number 1-8, usually 8.
VAR2- Destination variable.

Example: 20 SCCNT 8,A

If the sensor was set up as a 'Timed Sensor', it will contain the count for the current timing cycle - at the end of the cycle the value is transferred to **SPCNT** and **SCCNT** is reset to 0. This process will just continue.

SCCNT will count transitions (count up) if the input sensor was set up in continuous mode. The max value **SCCNT** can contain is 256^3.

The main purpose of **SCCNT** is to count objects with the I/R sensors.

Once the variable is set up, the BASIC program may be stopped, but **'SCCNT'** will still be counting. This is true of most R.O.S. commands. That is why timing can be so precise, even within a BASIC program.

Remember: **SCCNT** counts transitions. If an object interrupts the I/R beam that is 1 transition, when the object restores the beam, that is another transition. When counting objects, **SCCNT** should be divided by 2, however, an odd count indicates that an object is still within the I/R beam. This can be useful for many applications.

SSET: (Sensor Setting) This command is similar to the **MSET** command except it is for the sensors.

SYNTAX: **SSET VAR1,VAR2**
VAR1- Sensor number 1-7
VAR2- Destination variable

Bit 7 Activity 1 = on
0 = off

Bit 6 Direction 1 = output
0 = input

Bit 5 Previous state (reading during last interrupt cycle)

Bit 4 Mode 1 = Continous
0 = Timed

Bit 3 Timer initialized
1 = yes
0 = no

Bit 2
Bit 1 Don't care
Bit 0

APPENDIX 1

TROUBLESHOOTING
(IN CASE OF DIFFICULTY)

So you've tried everything and it still doesn't work.

The system was designed to be very reliable and trouble free. Here is a list of things to check.

CHECK LIST:

1. Make sure the batteries are installed.

2. Make sure the batteries are facing the proper direction.

3. Check the batteries to be sure they are not dead.

4. Check to see if the proper leads are hooked up.

5. Is the ON/OFF switch turned ON. (Most common)

6. Check to see if the correct software is loaded. (R.O.S., MULTIVOLT, MULTISCOPE, AUDIO).

7. Check the selector switch. Is it set properly? (METER/SCOPE, SENSORS, AUDIO/SPEECH)

8. Use **ALIGN 8,7 <return>** command to check the sensors.

9. Turn the computer and disk drive OFF and START OVER.

10. Run the **TEST PROGRAM** (see below) to check the B100 interface unit.

11. Sometimes plugging in a joystick will cause the keyboard to malfunction. If this occurs, you'll have to turn the computer off and try again. Leave the stick plugged in.

TEST PROGRAM

MAKE SURE R.O.S. IS NOT RUNNING. (Turn the computer off and on).

Load **TEST** by typing: **LOAD"TEST",8 <return> then RUN <return>.**

Follow the screen prompts.

If all else fails, call our office at (801) 298-9077 from 10:00 AM to 3:00 PM Mountain Time.

APPENDIX 2

OTHER PROJECT IDEAS

As you have seen by now, the Robotic Workshop is very versatile, and there are numerous uses for the B100 interface. The real fun of robotics is experimenting on your own and designing your own projects. Here are some examples.

HAM ANTENNA INDICATOR

By using the B100 as a voltmeter (from R.O.S.) with a 1 resistor shunt (see MULTIVOLT section), you can easily write a one line program to read out antenna position. Simply attach the voltage probe between ground and pin 3 on a CDE Ham I control unit.

HAM ANTENNA POSITIONER

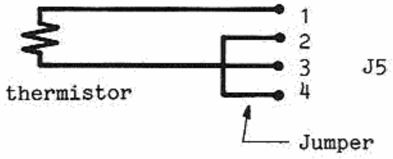
With the addition of 3 relays (operated by the motor leads!), you could easily control the antenna position more accurately than the normal control unit. You could even write the program to point to a specific location by just entering the destination CALL LETTERS!

EXERCISOR

By physically connecting a MULTIBOTICS motor to an exercise bicycle, you can visually monitor your progress. The motor will act as a generator. Use the MULTISCOPE on a slow time/division scale to plot your speed for the entire ride!

TEMPERATURE PROBE

A simple thermometer can be built. Just by adding a thermistor to the B100 interface.



A simple basic program in R.O.S. would read out temperature or would plot temperature for over 5 hours on the MULTISCOPE.

TEMPERATURE ALARM

By using a temperature probe, you could write a simple program which would set off an alarm when the temperature was out of a specific range.

MOTOR EFFICIENCY

A simple test jig could be built which would measure motor efficiency of small D.C. motors.

LUBRICATION TESTOR

A small setup using gears and motors could be made which would compare the effectiveness of different lubricants.

RELAY CONTROL

Relays can control many electrical devices. One way of operating a relay from the B100 is to connect a 5 volt relay to the motor leads. The MOTR command may then be used to operate the relay.

APPENDIX 3

R.O.S. MEMORY MAP

R.O.S. sits entirely in the memory space C000-CFFF. The starting location is C000, a SYS to 12*4096 (SYS 49152) will start R.O.S.

There are several important R.O.S. variable locations. They are listed in the variable table below. This information is provided for advanced machine language programmers.

```
                    : =======================================
                    : = MOTOR STATUS BYTE (MTRSTA)
                    : BIT7-MOTOR ON/OFF 1=ON 0=OFF
                    : BIT6-CONTINUOUS/TIMED MODE 1=CONT 0=TIMED
                    : BIT5-MOTOR UP TO SPEED 1=UTS 0=SPEEDING UP
                    : BIT4-FWD/RVS 1=FWD 0=RVS
                    : BIT3-CURRENT MOTOR CYCLE 1=ON 0=OFF
                    : BIT2-DON'T CARE
                    : BIT1-DON'T CARE
                    : BIT0-DON'T CARE
C003 - C005         MTRSTA    .BYT $00.$00.$00
                    : ======================================
                    : SPEED FOR MOTOR TIMING.  UPPER
                    : NYBBLE CONTAINS # OF ON CYCLES,
                    : LOWER NYBBLE CONTAINS # OF OFF
                    : CYCLES.      #1      #2      #3
C006 - C008         MSPDON    .BYT $00,$00,$00
C009 - C00B         MSPDOF    .BYT $00,$00,$00
                    : ==================
C00C - C00E         MTRTMR    .BYT $00,$00,$00
C00F - 0011         MCNTLO    .BYT $00,$00,$00
C012 - C014         MCNTMD    .BYT $00,$00,$00
C015 - C017         MCNTHI    .BYT $00,$00,$00
C018 - C01B         MTROFF    .BYT $FC,$F3,$CF,$3F
C01C - C01F         MOTRON    .BYT $03,$0C,$30,$C0
C020 - C023         MTRFWD    .BYT $01,$04,$10,$40
C024 - C027         MTRRVS    .BYT $02,$08,$20,$80
                    : ======================================
                        : ========================================
                        : SENSOR STATUS BYTES (SNRSTA)
                        : BIT7-SENSOR ACTIVE 1=ON 0=OFF
                        : BIT6-SENSOR DIRECTION 1=OUT 0=INPUT
                        : BIT5-PREVIOUS STATE 1/0
                        : BIT4-CONTINUOUS VS. TIMED COUNTING 1=C 0=T
                        : BIT3-SENSOR TIMER INITIALIZED 1=Y 0=N
                        : BIT2-DON'T CARE
                        : BIT1-DON'T CARE
                        : BIT0-DON'T CARE
                                       SENSOR NUMBER
                        _____________________________________________
                                     1   2   3   4   5   6   7   8
                        _____________________________________________
                        : ========================================
C033 - C03A             SNRSTA   .BYT $00,$00,$00,$00,$00,$00,$00,$00
C03B                    : ==================
C03B - C042             SMTMLO   .BYT $00,$00,$00,$00,$00,$00,$00,$00
C043 - C04A             SMTMMD   .BYT $00,$00,$00,$00,$00,$00,$00,$00
C04B - C052             SMTMHI   .BYT $00,$00,$00,$00,$00,$00,$00,$00
C053 - C05A             SCTMLO   .BYT $00,$00,$00,$00,$00,$00,$00,$00
C05B - C062             SCTMMD   .BYT $00,$00,$00,$00,$00,$00,$00,$00
C063 - C06A             SCTMHI   .BYT $00,$00,$00,$00,$00,$00,$00,$00
C06B - C072             SPCTLO   .BYT $00,$00,$00,$00,$00,$00,$00,$00
C073 - C07A             SPCTMD   .BYT $00,$00,$00,$00,$00,$00,$00,$00
C07B - C082             SPCTHI   .BYT $00,$00,$00,$00,$00,$00,$00,$00
C083 - C08A             SCNTLO   .BYT $00,$00,$00,$00,$00,$00,$00,$00
C08B - C092             SCNTMD   .BYT $00,$00,$00,$00,$00,$00,$00,$00
C093 - C09A             SCNTHI   .BYT $00,$00,$00,$00,$00,$00,$00,$00

                        : ========================================
                        : MASTER STATUS BYTES (MSTSTA)
                        :
                        : BIT7-ACTIVE OR INACTIVE 1=ON 0=OFF
                        : BIT6-00=SELF TIMING    01=WATCH MOTOR
                        : BIT5-10=WATCH SENSOR 11=DON'T CARE
                        : BIT4/3-00=WATCH SENSOR FOR TRANSITION
                        :        01=WATCH SENSOR CURRENT COUNT
                        :        10=WATCH SENSOR PREVIOUS COUNT
                        :        11-WATCH SENSOR CURRENT TIME
                        : BIT2-SENSOR #
                        : BIT1-OR MOTOR #
                        : BIT0-TO WATCH
                        : ==================
                                            MASTER BYTE
                                          1    2    3    4    5    6    7    8
                                        9   10   11   12   13   14   15   16
C09B - C0AA        MSTSTA     .BYT $00,$00,$00,$00,$00,$00,$00,$00
                              .BYT $00,$00,$00,$00,$00,$00,$00,$00
C0AB - C0BA        WVALLO     .BYT $00,$00,$00,$00,$00,$00,$00,$00
                              .BYT $00,$00,$00,$00,$00,$00,$00,$00
C0BB - C0CA        WVALMD     .BYT $00,$00,$00,$00,$00,$00,$00,$00
                              .BYT $00,$00,$00,$00,$00,$00,$00,$00
C0CE - C0DA        WVALHI     .BYT $00,$00,$00,$00,$00,$00,$00,$00
                              .BYT $00,$00,$00,$00,$00,$00,$00,$00
C0DE - C0EA        CHGSTA     .BYT $00,$00,$00,$00,$00,$00,$00,$00
                              .BYT $00,$00,$00,$00,$00,$00,$00,$00
C0EB - C0FA        CVALLO     .BYT $00,$00,$00,$00,$00,$00,$00,$00
                              .BYT $00,$00,$00,$00,$00,$00,$00,$00
C0FB - C10A        CVALMD     .BYT $00,$00,$00,$00,$00,$00,$00,$00
                              .BYT $00,$00,$00,$00,$00,$00,$00,$00
C10B - C11A        CVALHI     .BYT $00,$00,$00,$00,$00,$00,$00,$00
                              .BYT $00,$00,$00,$00,$00,$00,$00,$00
                   ; =======================================
```

APPENDIX 4

ACCESSORIES

The following accessories are available now or coming soon for your Robotic Workshop:

1. Additional Capsela® parts are available at better toy and hobby stores everywhere. (Gears, motors, etc.)
2. BE100 Battery Eliminator will replace the batteries in the B100 interface.
3. M100 Multiscope Enhancement Package. This will increase the speed and add many new features to the multiscope.
4. D100 Dynamics Module. More projects and theory in velocity and motion.
5. IE100 Instrumentation and Control Module will allow you to control more "real world" devices.
6. IRC100 Infrared Remote Control Module. This will allow you to build models which are not connected to the computer with a cable.
7. SR100 Switching and Relay Module. Control almost anything electronic with your B100 and computer.
8. R101 Robotics Module 1. More projects with Robotic devices.
9. R102 Robotics Module 2. The Robot Arm.
10. S100 Speech Digitization Enhancement Package. More on speech and sound effects and how to use them in your programs.

Many projects are being added to this list. If you have any suggestions for new modules or ideas for improving existing modules, please let us know.

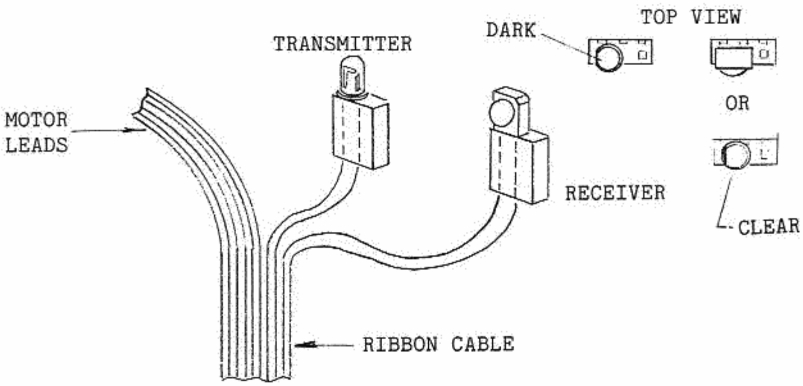
To keep up-to-date on available options, changes, hints, etc., make sure you send in your Warranty Registration Card.

APPENDIX 5

TECHNICAL NOTES

I/R SENSOR REPLACEMENT:

If the infrared sensors must be replaced, refer to the following diagram for proper location and orientation.



TECHNICAL INFORMATION:

The following information is available upon request at no charge.

1. How to interface with R.O.S. through machine language.
2. B100 Schematic Diagrams.
3. Motor and Sensor data sheets.

Call our office at (801) 298-9077 or write to:

MULTIBOTICS, INC.
2561 South 1560 West
Woods Cross, Utah 84087

Please specify which materials you desire and include your return address and phone number.