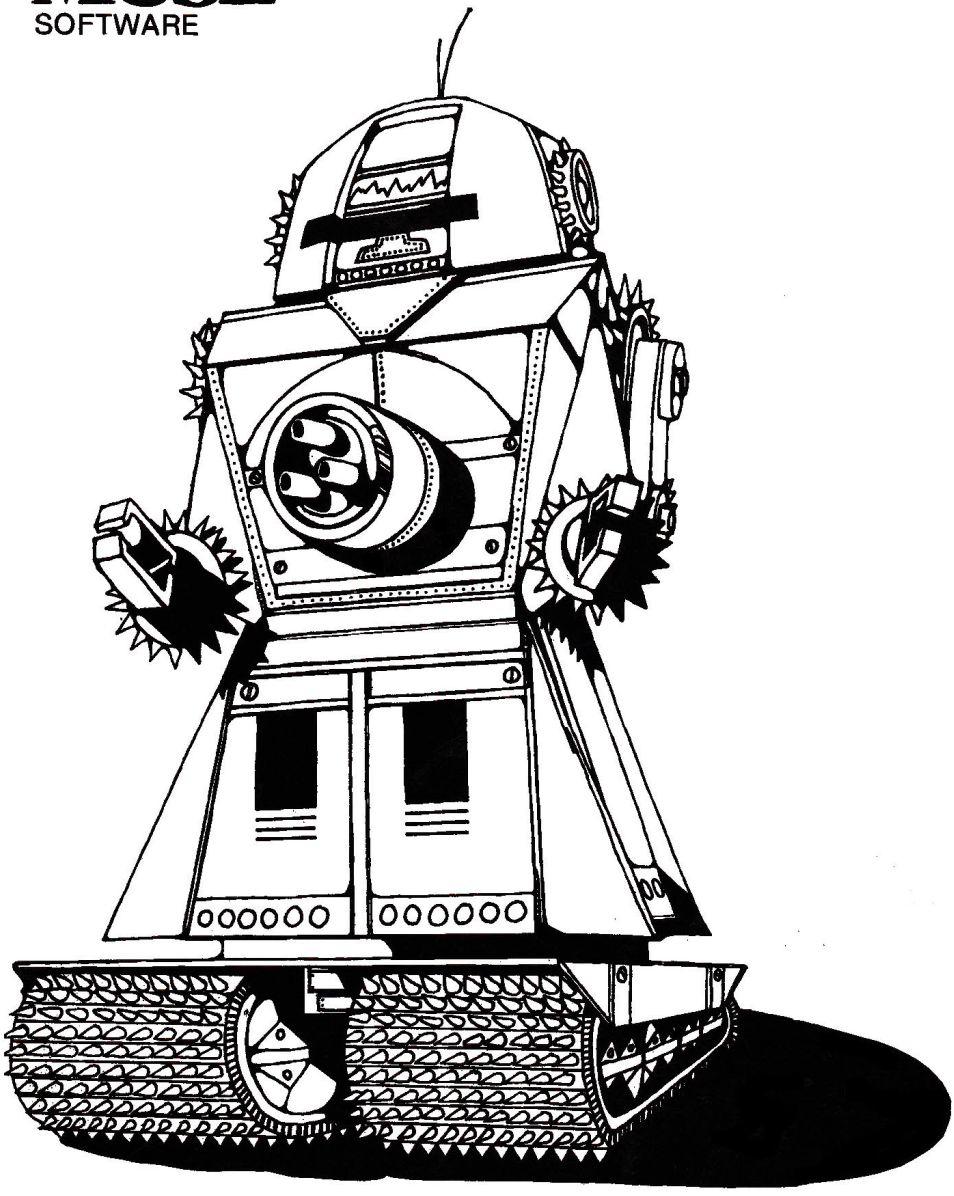
MUSE™
SOFTWARE

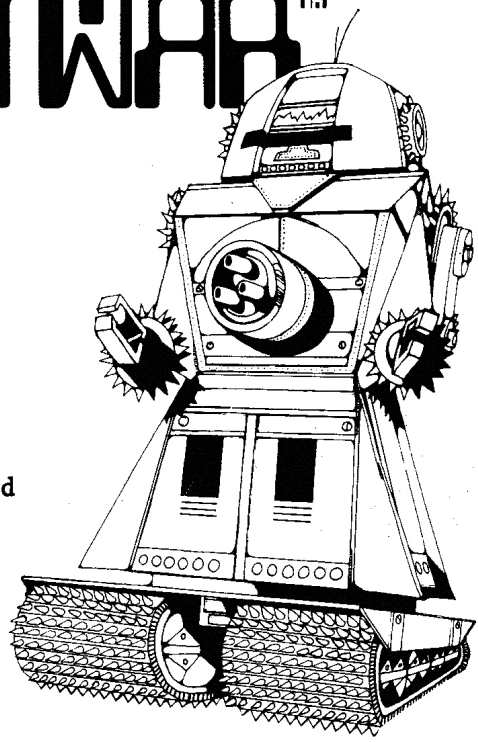


ROBOTWAR™

For Apple II or Apple II Plus with 48K,
Applesoft ROM

Apple is a TM of Apple Computer, Inc.

ROBOTWAR™



By Silas Warner

Published by

MUSE SOFTWARE, Inc.
347 N. Charles Street
Baltimore, MD 21201

For Apple II or Apple II Plus
Requires 48K and Applesoft ROM

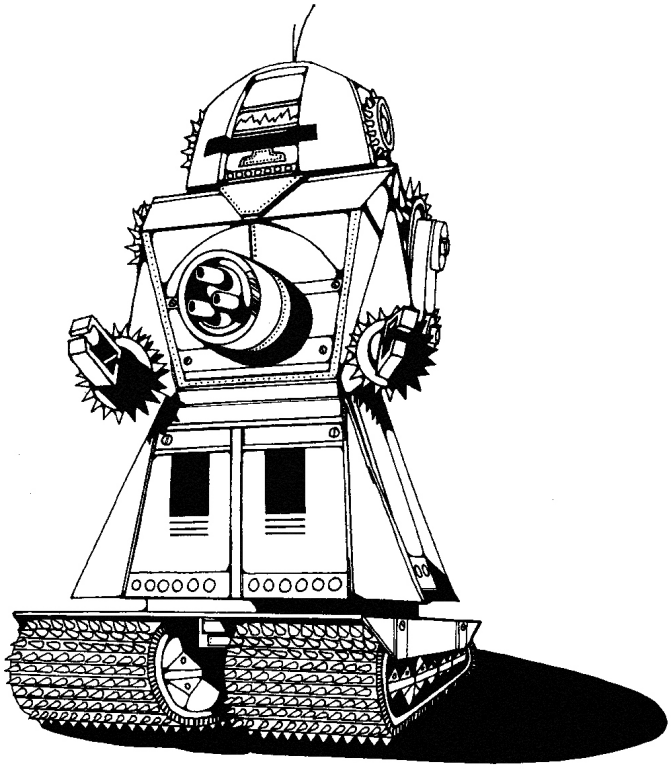
DO NOT UPDATE this disk with other versions of the disk operating system (DOS). If you do it will destroy this program disk.

REPLACEMENT - If this disk becomes worn or damaged, Muse Software will gladly replace it. Send the damaged disk with proof of purchase and $10.00 to:

Muse Software
347 N. Charles Street
Baltimore, MD 21201

This documentation manual was prepared and printed using Super-Text - the professional word processor for the Apple II, from Muse. Always ask for Muse Quality Software at your local Computer store.

TABLE OF CONTENTS



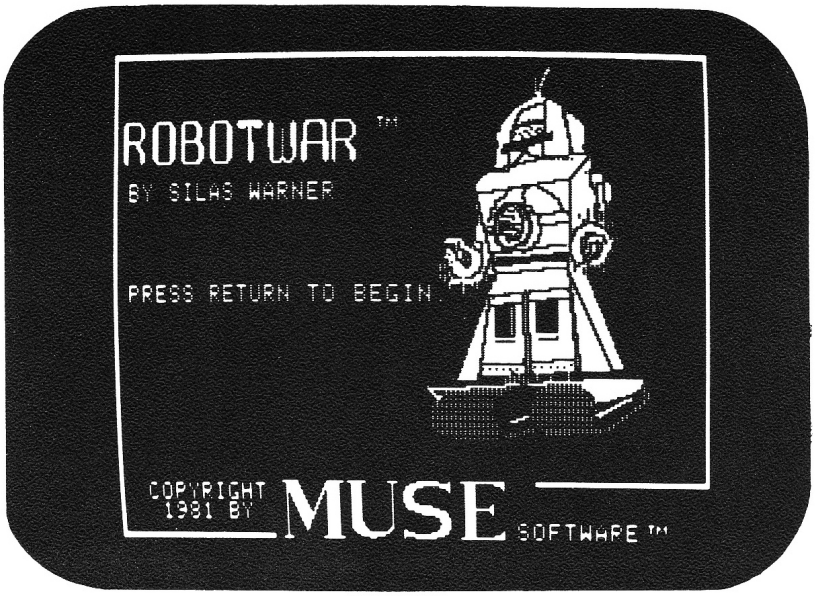
WELCOME TO THE BATTLEFIELD OF THE FUTURE! It is the year 2002. Wars still rage, but finally, they have been officially declared hazardous to human health. Now, the only warriors are robots - built in secret and programmed to fight each other to the death!

Your country has just developed the most efficient battle robot to date. It should be unbeatable - but part of its micro-computer "brain" is still blank. Only when a strategy is programmed into its memory will the robot be able to fight.

The task set before you is:

TO PROGRAM A ROBOT,

THAT NO OTHER ROBOT CAN DESTROY!



INTRODUCTION

RobotWar is a fascinating and highly competitive game where robots battle each other to the death! RobotWar is not a game using manual dexterity, instead the robots are controlled by pre-programmed instructions. This makes RobotWar a game of elegant strategies and high spectator interest.

As well as providing hours of entertainment, RobotWar is designed to teach and sharpen the skills of creative computer programming. Whether you are a beginner or an accomplished programmer, RobotWar will prove to be fun and challenging.

RobotWar players design and write robot programs. The program is written with the help of a text-editor, and then translated by an assembler into robot-understandable instructions. The program can then be tested on a simulated robot to make sure it is working properly. Once the player is assured that the program is running as planned, it is installed in a battle robot and sent out to do battle with the other robots.

The RobotWar Manual

This manual is intended to teach the player all of the skills necessary to create and battle robots. These skills involve the basics of programming and the operation of text-editors. The skills are easily learned if each chapter is read in its proper order.

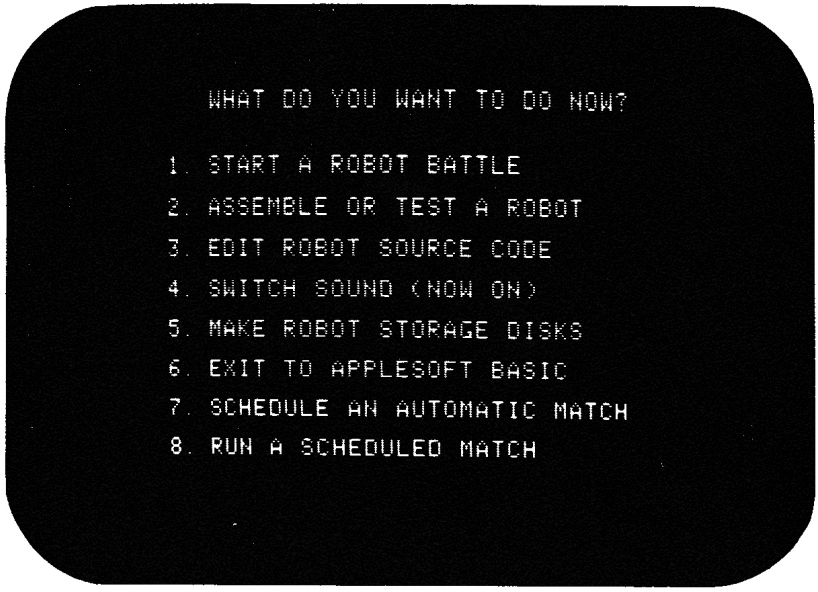
Starting The Game

To begin, boot the RobotWar disk. After the program loads, the title page will appear on the screen and prompt you to press

RETURN

After pressing RETURN, the main menu will appear

Exhibit 1: The Main Menu



The player can access any section of the RobotWar program from the main menu by selecting one of its eight options. These options are described below:

Option 1) This will access the Battle branch where the player can set up and execute one robot battle. See "ROBOTS AND ROBOT BATTLES".

Option 2) This will access the RobotWar Assembler and Testing branch where the programs are translated and checked for errors, or tested on a simulated robot. See "THE ASSEMBLER" and "THE TEST BENCH".

Option 3) This will access the Text-Editor where an existing program can be edited or a new program can be written. See "WRITING AND EDITING SOURCE CODE".

Option 4) This is a simple control that turns the battle sounds on or off. Pressing the 4 key will change the position of the sound switch.

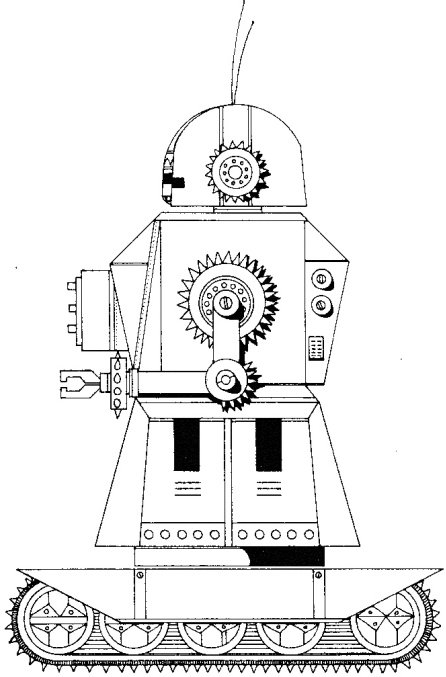
Option 5) This will access the Disk Storage branch where a disk can be initialized for storing robot code. See "STORING ROBOTS ON AUXILLARY DISKS".

Option 6) This will cause the computer to exit from the RobotWar program to Applesoft basic.

Option 7) This will access the Match Scheduling branch where the player can schedule and execute a series of battles. See "ROBOTS AND ROBOT BATTLES".

Option 8) This will allow the player to run a previously scheduled or interrupted match (a series of battles). If you resume a previously interrupted match it will begin with the battle after the one which was interrupted.

Note: If no option is selected from the main menu, the program will automatically select option eight.



ROBOTS AND ROBOT BATTLES

Locomotion

Each robot is moved by tracks mounted on a 1.5 meter square chassis. The two independent motors, driving the tracks, enable the robot to move vertically (north/south) and horizontally (east/west).

Power Supply

The power supply will take the severest damage from the enemy shells. It is built into the central body of the robot, along with damage sensors. These sensors monitor the damage to the power supply and when 100% damage is attained, the robot will explode!

Radar

On top of the robot is a radar unit that emits a beam in any desired direction. This beam reflects from walls and other robots and returns to the robot. The beam is accurately timed, enabling the robot to find its position and to spot enemy robots.

Guns and Ammunition

Your robot is equipped with one gun that swivels through 360 degrees and is automatically loaded. It uses time-fused shells that can be set to explode at any specified distance. The gun also has a cooling period between each shot to keep it from overheating.

The Brain

Inside the robot is a micro-computer "brain" that executes the instructions exactly as they have been programmed. The brain has several parts: an accumulator where a robot performs all arithmetic operations, a program storage area where the instructions are stored in memory, and registers where numbers are stored. The brain links to input sensors monitoring damage and position as well as to the drive motors, radar, and gun. While the robot is on the battlefield the brain is in complete control!

The Battlefield

Robot battles take place on a square battlefield inside four strong walls. Each wall is 260 meters long and strong enough that a robot cannot crash or shoot through it. As many as five robots can fight at once, but only one will emerge as the winner.

There is an observation station, directly above the battlefield, enclosed in blast-proof glass to protect you and the other observers.

Damage

Robots are eliminated from battle by incurring over 100% damage. When a shell hits a robot or explodes nearby, the robot is damaged. The extent of that damage depends on the proximity of the shell to the robot. A shell exploding directly on top of a robot can do 30% damage.

A robot can also be damaged through collisions with walls or other robots. The extent of damage would depend on the angle of collision. A head-on collision between two robots can do 25% damage to both robots.

The Scoring System

Each robot has a score associated with it. As each battle is fought the robots earn points which are added to its cumulative score. Every time a robot's program is changed, its score is reset to 0.

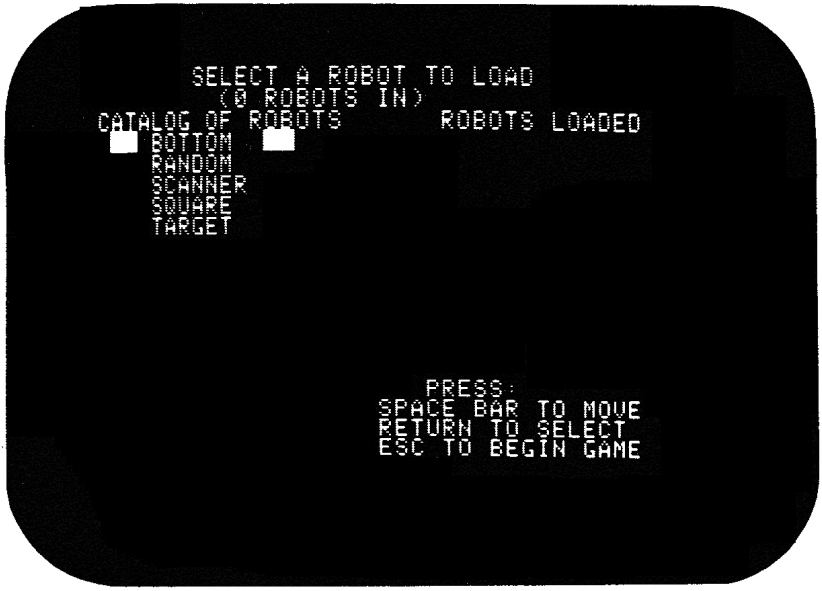
Robots earn points in the following manner. During a battle, every time a robot is destroyed, 1 point is earned by all of the survivors. Thus in a five-robot battle, the first to be destroyed receives 0 points. For outlasting that first robot, all other robots on the battlefield earn 1 point. For outlasting 4 other robots, the winner of a 5-robot battle earns 4 points!

Battling It Out

To start a robot battle, select option one from the main menu by pressing

A catalog of available robots will appear on the screen. RobotWar comes with of five pre-programmed robots. As you create more robots, their names will be added to the list (see Exhibit 2).

Exhibit 2 : Selecting Robots From The Catalog



To select a robot: move the cursor, by pressing the space bar, until it is next to the robot of your choice. Then press

RETURN

to select the robot at the cursor.

Note: this selection process is utilized throughout the RobotWar program whenever it is neccessary to choose a robot from a catalog.

After you select a robot, its name is entered under the "robots loaded" column. Repeat the selection procedure until you have loaded all the desired robots.

A maximum of five robots can be loaded, after which, the program will automatically enter the battle.

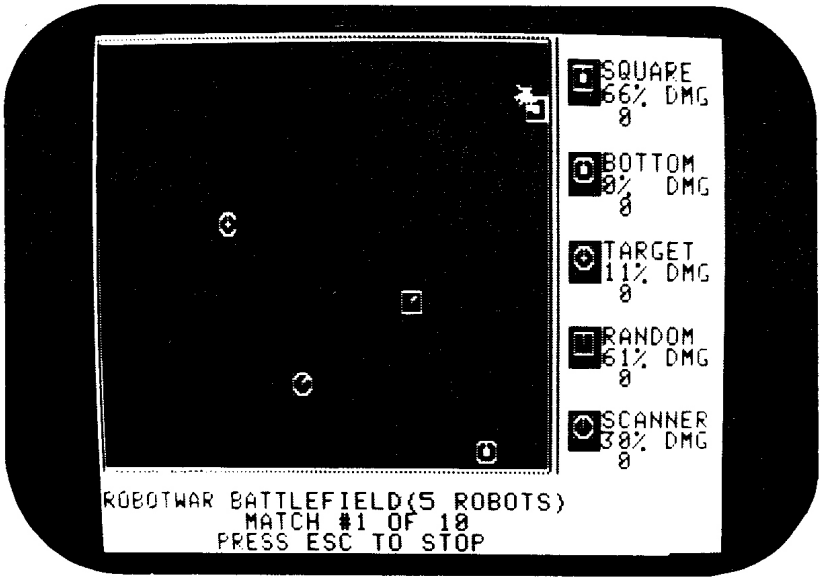
If you are selecting less than five robots, press

ESC

to enter the battlefield.

The screen will now display the battlefield. On the right side of the screen each robot's assigned symbol, its percent of damage and cumulative score are displayed.

Exhibit 3 : View From The Observation Station



To start the battle, press

RETURN

If you wait, the battle will start automatically.

The robots are designed to fight to the death. However, the RobotWar program will automatically stop any battle where the robots fail to damage each other. Of course, as referee you may stop the battle anytime you wish by pressing the ESC key and returning to the main menu.

You may also wish to turn the sound system on or off, while the battle is in progress. Press the Q key to turn the sound off and the S key to turn it back on.

The scores of the robots will change as the robots are eliminated from the field. When the victorious robot emerges, the screen will display its name. Press

RETURN

to exit to the main menu.

Note: If you wait, the computer will automatically return to the main menu.

Scheduling A Match

Robots can fight a series of battles against each other by scheduling a match. The match will execute all of the battles automatically, one right after the other. To schedule a match, access the main menu and press

At this point, the screen will display the catalog of robots available for battle. Select the robots for the match by using the catalog selection process described earlier. When all of the desired robots are loaded, press

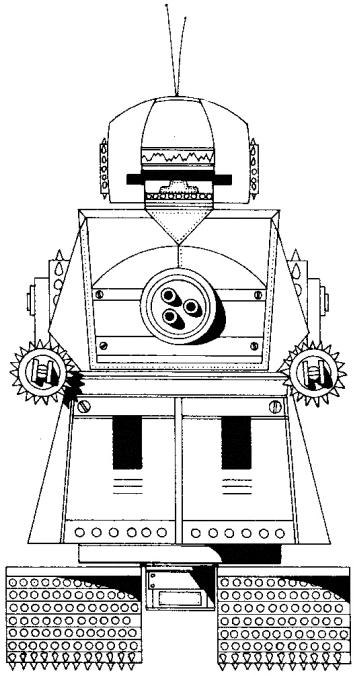
The following message will appear on the screen.

HOW MANY BATTLES DO
YOU WANT TO RUN?

Type in the number of battles you would like to run and press

to begin the match!

After the last battle, the screen will display the total results of all the scheduled battles. To exit to the main menu press



CONTROLLING ROBOTS

A robot computer contains 34 registers. The 34 registers are divided into three categories:

1. Memory registers which are used to contain numbers for later recall.

2. Input/Output (I/O) registers which are used to monitor and control specific robot functions.

3. The Index/Data pair of registers which are used to access the other registers by their numbers instead of their names.

1. Memory Registers

There are 24 memory registers used to store numbers. The memory registers are named A through W and Z (X and Y are not included - they are input registers as described below).

2. Input/Output Registers

There are nine I/O registers that allow the computer to control the robot's actions. Each controls or monitors a specific robot function as described below:

a) The X register:

The X register is used to monitor the horizontal position of the robot. It always contains the current horizontal position of the robot on the battlefield, as a number from 0 to 256. 0 is at the extreme left of the battlefield and 256 is at the extreme right (see Exhibit 4).

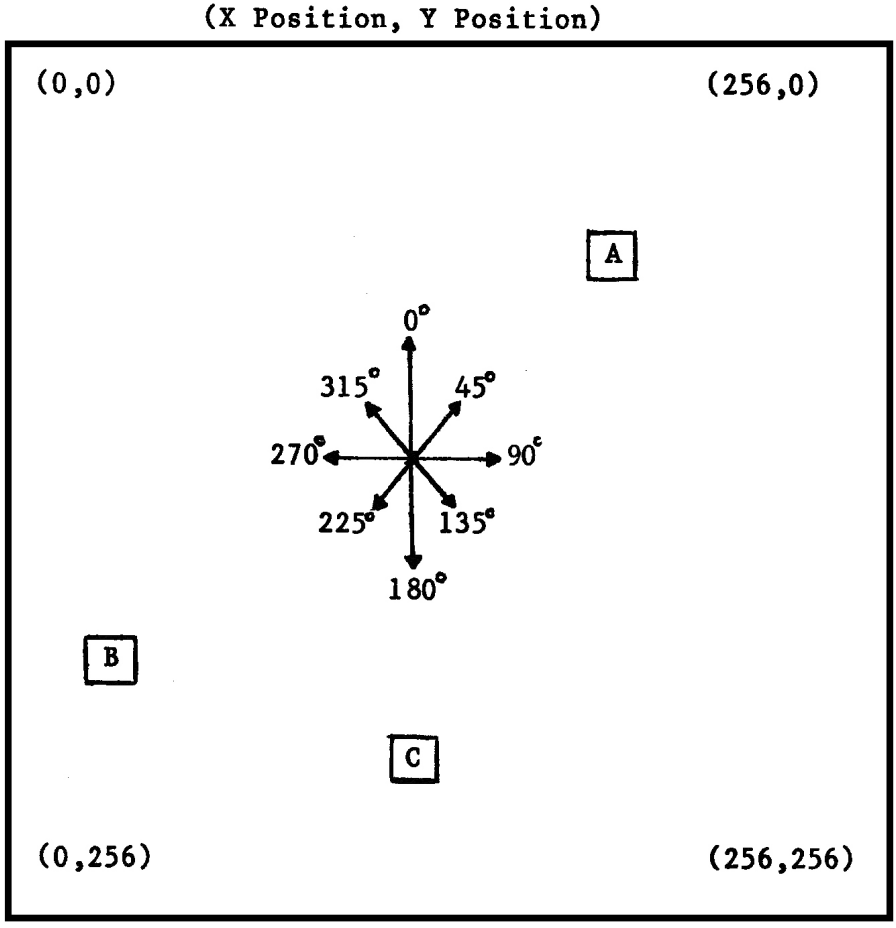
b) The Y register:

The Y register is used to monitor the vertical postion of the robot. 0 is at the top of the battlefield and 256 is at the bottom (see Exhibit 4).

c) The AIM register:

The AIM register is used to monitor and control the angle at which the gun is aimed. When a number from 0 to 359 is stored in the Aim register, the robot's gun will turn to that angle. 0 aims the gun due north, 90 aims it due east, etc. (see Exhibit 4). The AIM register always contains the current angular position of the gun.

Exhibit 4: Diagram Of The Battlefield



In the diagram above:

Robot A has an X position of 200 and a Y position of 50 (200,50).

Robot B has an X position of 10 and a Y position of 185 (10,185).

Robot C has an X position of 128 and a Y position of 210 (128,210).

d) The RADAR register:

The RADAR register is used to control the radar unit on top of the robot and monitor the results of the radar beam. Storing a number from 0 to 359 in the RADAR register, sends a beam out in that direction (see Exhibit 4).

When the beam returns to the robot, the RADAR register will contain the results as a number. This number can range from 0 to 400 and be either positive or negative. If the number is positive, there is a wall, that many meters away, in the direction the radar was aimed. If the number is negative, there is an enemy robot in that direction. The distance to the enemy robot is equal to the negative of the radar result.

For example: If you store a 90 in the RADAR register, a beam will be emitted in the 90-degree direction. The beam will then bounce off, either another robot or a wall, and return. If the RADAR register then contains a **+150**, it means that a wall lies 150 meters away in the 90-degree direction. If the RADAR register contains a **-68**, it means another robot is presently 68 meters away in the 90-degree direction.

e) The SHOT register:

The SHOT register is used to fire the robot's gun and monitor the state of readiness of the gun. Storing a new number in the SHOT register: sets the timer on the shell so that it will travel that number of meters before exploding, and then fires it. After a shot is fired the SHOT register will contain the state of the gun's cooling process. When the SHOT register contains a zero the gun is ready to be fired again.

f) The DAMAGE register:

The DAMAGE register is used to monitor the amount of damage detected by the damage sensors. The DAMAGE register starts at 100 at the beginning of each battle and decreases towards 0 as damage is incurred. When the register reaches 0, the robot is completely destroyed and will disappear from the battlefield. The DAMAGE register always contains the current extent of damage.

g) The SPEEDX register:

This register is used to control and monitor the horizontal speed of the robot. The number stored in the SPEEDX register can range from -255 to 255 and controls the direction and speed of the robot. A negative number moves the robot to the left at that many decimeters/second, and a positive number moves the robot to the right at that many decimeters/second. If a zero is stored in this register the robot will stop moving in the horizontal direction. The SPEEDX register always contains the horizontal speed of the robot.

h) The SPEEDY register:

Acts the same as the SPEEDX register, only in the vertical direction. A positive number is in a downward direction and a negative number is in an upward direction.

i) The RANDOM register:

This register is used to control the random number generator. Storing a number in the RANDOM register sets the limit for the generator. Then, each time the RANDOM register is accessed, it will contain a different integer (whole number) between 0 and the random number limit which was previously set.

3. The Index/Data Registers

The robot registers are usually referenced by their names. The Index/Data pair allows registers to be accessed by number instead of name.

Storing a number from 0 to 34 in the INDEX register causes the corresponding register to be used whenever the DATA register is referenced.

For example, assume the INDEX register contains 27. When the DATA register is referenced in an instruction, register #27 (AIM) will be substituted for DATA.

ROBOT REGISTERS

Number	Name	Type
1	A	Storage
2	B	Storage
3	C	Storage
4	D	Storage
5	E	Storage
6	F	Storage
7	G	Storage
8	H	Storage
9	I	Storage
10	J	Storage
11	K	Storage
12	L	Storage
13	M	Storage
14	N	Storage
15	O	Storage
16	P	Storage
17	Q	Storage
18	R	Storage
19	S	Storage
20	T	Storage
21	U	Storage
22	V	Storage
23	W	Storage
24	X	Current X position
25	Y	Current Y position
26	Z	Storage
27	AIM	Control gun aim
28	SHOT	Fires the gun
29	RADAR	Pulse radar
30	DAMAGE	Monitor damage
31	SPEEDX	Control horizontal speed
32	SPEEDY	Control vertical speed
33	RANDOM	Random number generator
34	INDEX	Index to other registers

THE LANGUAGE OF ROBOTS

The Source Code

Robot programs are written in source code and then translated by the assembler into robot-understandable object code. Source code is composed of comments, labels, and instructions.

1. Comments:

Comments are used for documenting the source code. Comments can appear anywhere in the program as long as they are preceded by a semi-colon.

```
] A TO B      ;THIS STORES A IN B
```

This is an example of a comment on the same line as an instruction.

2. Labels:

A label is a reference point used to identify sections within a program. Labels are used in instructions to change the order of execution of the program.

A label is composed of a group of 2 or more alpha-numeric characters immediately following a RETURN (]). A label must start with an alpha character (A to Z) and must be less than 32 characters long. A label can not be the same as any of the register names or command words.

3. Instructions:

Instructions are used to control the robot's micro-computer brain. Instructions may contain register names, command words and numbers (-1024 to +1024)

Commands

TO	Stores a value in a register.
+	Adds two values.
-	Subtracts two values.
*	Multiplies two values.
/	Divides one value by another.
IF	Compares two values and alters program sequence.
GOTO	Goes to a label in the program.
GOSUB	Executes a subroutine.
ENDSUB	Returns from a subroutine.

The TO command

The TO command is used to store a value in a register.

```
] 240 TO A
```

This example line of source code causes the computer to load the accumulator with a value of 240 and then store it in the A register.

```
] B TO A
```

This example causes the computer to load the accumulator with the contents of the B register and then store it in the A register.

```
] 0 TO SPEEDX TO SPEEDY
```

This example causes the computer to load the accumulator with 0 and store it first in the SPEEDX register and then in the SPEEDY register. In a real robot this instruction could be used to stop horizontal and vertical movement.

Note: Negative numbers can be stored as in the following example:

```
] -240 TO SPEEDX
```

But, you CANNOT store the negative of a register in that manner. For example:

```
] -B TO A
```

will NOT store the negative of B in A. To store a negative of a register subtract the register from zero. For example:

```
] 0 - B TO A
```

Arithmetic commands (+ - * /)

Arithmetic operations can be performed on a value stored in the accumulator. Whenever the program encounters one of the arithmetic signs it performs the calculation using the contents of the accumulator and the value that follows. It then stores the results of the calculation in the accumulator.

```
] 240 + 100 TO A
```

This example loads 240 into the accumulator, adds 100 to it, and stores the result (340) in the A register.

```
] A * 2 TO B
```

This example loads the accumulator with the contents of the A register, multiplies it by 2 and stores the result in the B register.

The IF command

The IF command is used to compare a value with the contents of a register. It can test to see if a register is less than (<), greater than (>), equal to (=), or not equal to (#) a value. If the comparison is true the computer executes the next TO, GOTO, GOSUB or ENDSUB command. If the comparison is false the computer skips the next TO, GOTO, GOSUB or ENDSUB command.

```
     ]SCAN
     ]  AIM + 5 TO AIM            ;MOVE GUN
     ]  AIM TO RADAR              ;SEND RADAR PULSE
     ]LOOP
*    ]  IF RADAR < 0 GOSUB FIRE   ;TEST RADAR
     ]  GOTO SCAN
     ]FIRE
     ]  0 - RADAR TO SHOT         ;FIRE THE GUN
     ]  ENDSUB
```

In the above example the IF command (marked with *) tests the results of a radar scan. If the radar register contains a value less than zero (indicating an enemy robot) the computer will execute the 'GOSUB FIRE' command. If the radar is not less than zero the computer will skip the 'GOSUB FIRE' command.

The GOTO command

A GOTO command causes the program to change its sequence of execution by going to a designated label and continuing its execution from there. A GOTO instruction must always be followed by a label.

```
      ]SCAN
      ]  AIM + 5 TO AIM            ;MOVE GUN
      ]  AIM TO RADAR              ;SEND RADAR PULSE
      ]LOOP
      ]  IF RADAR < 0 GOSUB FIRE   ;TEST RADAR
*     ]  GOTO SCAN
      ]FIRE
      ]  0 - RADAR TO SHOT         ;FIRE THE GUN
      ]  ENDSUB
```

In this example the 'GOTO SCAN' command causes the program to go to the label 'SCAN' and continue scanning for an enemy robot.

The GOSUB command

Another way to change the execution sequence is to use a GOSUB command. A GOSUB instruction is similar to a GOTO instruction. GOSUB must always be followed by a label. GOSUB will cause the program to go to the designated label and continue the execution until it reaches an ENDSUB. When it encounters the ENDSUB, the program will then return to the next instruction after the GOSUB.

```
     ]SCAN
     ]  AIM + 5 TO AIM            ;MOVE GUN
     ]  AIM TO RADAR              ;SEND RADAR PULSE
     ]LOOP
*    ]  IF RADAR < 0 GOSUB FIRE   ;TEST RADAR
     ]  GOTO SCAN
     ]FIRE
     ]  0 - RADAR TO SHOT         ;FIRE THE GUN
     ]  ENDSUB
```

In this example the 'GOSUB FIRE' command will cause the computer to execute the subroutine that starts with the label 'FIRE' and ends with the 'ENDSUB' command. After executing the subroutine the computer will continue with the line that contains the 'GOTO SCAN' command.

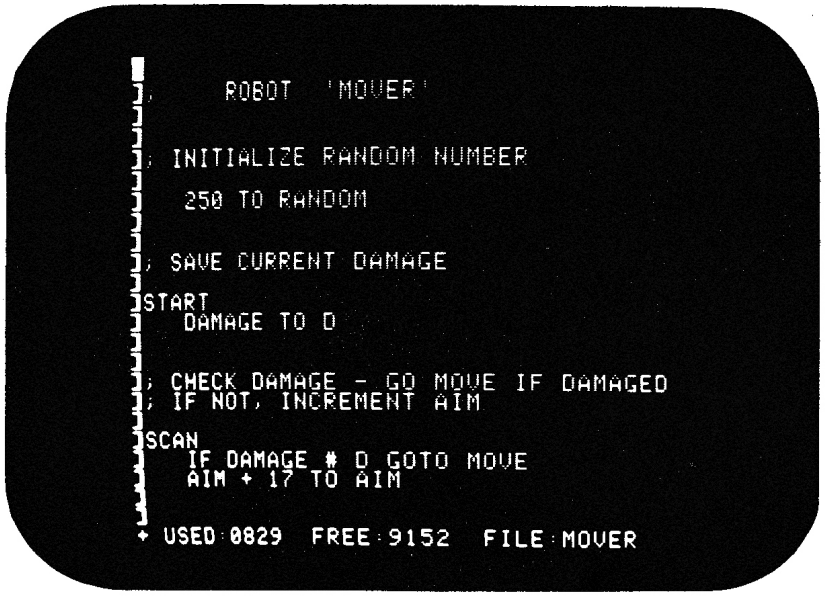
LEGAL INSTRUCTIONS

A TO B	the simplest assignment, stores the value of A in B
100+B-C*D/E TO F	arithmetic operations can be done
AIM TO A TO C	you can store a value in more than one place
LABEL TO A	labels can be stored because they translate as numbers
GOTO LABEL	goes to a label and continues execution
GOTO 3	goes to instruction 3
GOTO A	goes to instruction whose number is in register A
IF A+3>B GOTO LABEL	arithmetic operations are allowed before the comparison sign
IF A=B C*4 TO C	conditional assignment
IF A=B GOSUB LABEL	conditional GOSUB
IF A=B ENDSUB	conditional ENDSUB

ILLEGAL INSTRUCTIONS

A*(B+C) TO D	no parentheses allowed
20.35 TO E	only integer numbers allowed
-A TO B	use 0 - A TO B instead
IF A GOTO LABEL	there must be a condition sign
IF A+B=C*3 GOTO LABEL	no arithmetic after the condition sign

Some of these illegal statements will be translated by the assembler, but then will do odd things when executed.



PROGRAMMING A ROBOT

In order to make a robot perform, you must construct a program using the RobotWar language and your own strategy. This chapter gives examples of how instructions can be constructed, using registers, numbers, and commands, and how those instructions can be labeled and sequenced to create program routines.

Movement

Moving about the battlefield is an action a robot performs. To start a robot moving, store a value in the SPEEDX or SPEEDY register.

For example:

```
] 30 TO SPEEDX
] 250 TO SPEEDY
```

would start the robot moving down and to the right. However, the robot would continue to move in those directions, and would eventually hit a wall. Therefore, you must stop it at some point, by storing a zero in the SPEEDX and SPEEDY registers.

For example:

```
] 0 TO SPEEDX
] 0 TO SPEEDY
```

A robot can only accelerate or brake at 40 decimeters/second. Even though 120 is entered into the SPEEDX register, it takes 3 seconds of acceleration to obtain that speed. Conversely, if your robot is travelling at 120 decimeters/sec it takes 3 seconds to stop the robot, after storing 0 in the SPEEDX register.

A movement routine can be established, by incorporating the starting and stopping procedures into a test loop.

```
] 256 TO SPEEDX
]MOVER1
] IF X > 230 GOTO STOP
] GOTO MOVER1
]STOP
] 0 TO SPEEDX
```

moves the robot to the right until its X position is tested to be greater than 230 and then stops it.

Monitoring Damage

Monitoring damage is vital to a robot's survival. When a robot detects a hit, it usually moves to avoid being repeatedly hit by the enemy. By using the DAMAGE register, a damage detection routine can be established. This routine is usually nested inside another routine's loop so that the robot can be checking for damage while it is performing some other action.

For example:

```
] DAMAGE TO D
```

saves the current damage in register D.

```
]DAM1
] IF DAMAGE # D GOTO MOVE
```

When any damage is incurred, the DAMAGE register will change, but register D will not. Therefore, any difference between the two registers will indicate that the robot has been hit. In this example any difference will cause the program to go to the label MOVE.

Scanning

Another important action a robot performs is scanning. When a robot scans it is using its radar beams to detect the location of other robots and walls. To emit a radar beam, store a number, between 0 and 359, in the RADAR register.

```
] 90 TO RADAR
```

will send a radar beam in the 90-degree direction, and when the beam returns, its value will be stored in the RADAR register. A routine to determine if the robot has spotted another robot is:

```
]LOOK
]  AIM + 5 TO AIM
]  AIM TO RADAR
]  IF RADAR < 0 GOTO SHOOT
]  GOTO LOOK
```

When the program executes this routine, it first encounters the label LOOK and goes on to the next instruction. This instruction (AIM + 5 TO AIM) increments the angle in which the gun is aimed, five degrees. The next instruction (AIM TO RADAR) aligns the angle of the radar to the angle of the gun, emits a radar beam in that direction, and then stores the results of that beam in the RADAR register.

The next instruction (IF RADAR < 0 GOTO SHOOT) analyzes the results of the radar's findings. If the RADAR register contains a positive number, there are no robots in that direction and the comparison will be false. Since the comparison is false, the next command will be ignored and the program will go on to the next command (GOTO LOOK). This command will cause the program to go to the label LOOK. This completes the loop and the scan routine will continue until a robot is found.

If the RADAR register contains a negative number, after the beam returns, the comparison (IF RADAR < 0) will be true. Therefore, the next command (GOTO SHOOT) will be executed. In this case the program sequence would branch to the instruction following the label SHOOT.

Shooting

It is usual procedure to execute a shooting routine when an enemy is spotted.

```
]SHOOT
]  0 - RADAR TO SHOT
]  GOTO LOOK
```

is an example of a simple shoot routine. Since a robot has been spotted by radar, a negative number is presently stored in the RADAR register. The enemy robot is that number (ignoring the negative sign) of meters away. In order to obtain a positive number of the distance, the program subtracts RADAR from 0. This new positive number is then stored in the SHOT register. Storing the number in the SHOT register causes the gun to fire a shell that has been set to explode at that distance, in the direction indicated by the contents of the AIM register.

Random Number Generation

The RANDOM register is used to generate random numbers. A few examples of random number routines are:

```
] 100 TO RANDOM
] RANDOM TO A
```

This routine stores 100 in the RANDOM register, which sets the limit for the generator. The generator then returns a random number from 0 to 99 and stores it in the RANDOM register. That value is then stored in A by the TO command. From then on each time the contents of the RANDOM register is stored in a register, the generator will return a different number. The limit of the generator will only change when a new value is stored in the RANDOM register by using the TO command.

```
] 513 TO RANDOM
] RANDOM - 256 TO B
```

This code stores a random number between -256 and 256 into the B register.

```
] B + 1 - A TO RANDOM
] RANDOM + A TO C
```

This routine stores a random number between A and B into the C register.

A SAMPLE ROBOT IN SOURCE CODE

```
]  250 TO RANDOM                ; INITIALIZE RANDOM NUMBER
]
]START
]  DAMAGE TO D                 ; SAVE CURRENT DAMAGE
]
]SCAN
]  IF DAMAGE # D GOTO MOVE     ; TEST : MOVE IF DAMAGED
]  AIM + 17 TO AIM              ; IF NOT, INCREMENT AIM
]
]SPOT
]  AIM TO RADAR                ; ALIGN RADAR TO AIM
]  IF RADAR > 0 GOTO SCAN      ; SCAN IF NO ENEMY FOUND
]  0 - RADAR TO SHOT           ; OR SHOOT SPOTTED ENEMY
]  GOTO SPOT                   ; IS ENEMY STILL THERE
]
]MOVE
]  RANDOM TO H
]  RANDOM TO V                 ;PICK A RANDOM PLACE TO GO
]
]MOVEX
]  H - X * 100 TO SPEEDX      ; TRAVEL TO NEW X LOCATION
]  IF H - X > 10 GOTO MOVEX   ; TEST X POSITION
]  IF H - X < -10 GOTO MOVEX; TEST X POSITION
]  0 TO SPEEDX                ; STOP HORIZONTAL MOVEMENT
]
]MOVEY
]  V - Y * 100 TO SPEEDY      ; TRAVEL TO NEW Y LOCATION
]  IF V - Y > 10 GOTO MOVEY   ; TEST Y POSITION
]  IF V - Y < -10 GOTO MOVEY; TEST Y POSITION
]  0 TO SPEEDY                ; STOP VERTICAL MOVEMENT
]  GOTO START                 ; START SCANNING AGAIN
```

WRITING AND EDITING SOURCE CODE

Robot programs are entered into the computer using a text editor.

The text editor may be entered by selecting option 3 from the Main Menu, or by selecting option 6 from the Assembler Menu.

Text-Editor Procedure

When you first enter the text editor, you will see a blank screen with some numbers at the bottom and a flashing square at the top. The numbers at the bottom show the length of the text, and the file name under which it is stored. The flashing square is called the cursor, and is the computer equivalent of a pen for writing characters. As you use the text-editor you will be operating in two modes; the add mode and the cursor mode. The add mode is used simply to add text at the cursor. The cursor mode is used to delete text at the cursor, move the cursor around in the text, adjust the position of the text on the screen, load source code files from the catalog, and save source code files to the catalog.

The blank screen indicates that the current text-editor file is empty. At this point there are two available options. One option is to begin writing a new source code, and the other option is to edit a robot that has already been stored.

To begin writing a new robot program, follow these steps:

1. Press

   CTRL-A

   to enter the add mode. The letter 'A' will appear in the lower RIGHT corner of the screen. You can now create a new source code file.

2. Press

   RETURN

   several times and then type in the sample robot program below. As you type, the characters will appear on the screen. The RETURN key is used to start a new line, and marks the new line with a ']' in the left margin. You may also use the left arrow key to backspace and correct mistakes.

```
];    SAMPLE ROBOT
]
]SCAN
]  AIM + 5 TO AIM              ;MOVE GUN
]  AIM TO RADAR                ;SEND RADAR PULSE
]
]LOOP
]  IF RADAR < 0 GOSUB FIRE    ;TEST RADAR
]  GOTO SCAN
]
]FIRE
]  0 - RADAR TO SHOT          ;FIRE THE GUN
]  ENDSUB
```

3. After typing the source code for the sample robot, press

   to exit from the add mode.

You have just created a new source code file, but the only place it exists is in the memory of the Apple. To save this new file on disk follow these steps:

1. Press

   The Source Code Catalog will now appear and the word 'SAVE' will appear on the left side of the screen.

2. To save the new robot program you just created you must give it a name. The name can be no longer than 7 characters and must not be the same as other robots on the disk. Let's call your robot 'SAMPLE'.

3. Type in the name 'SAMPLE' and press

4. Before saving the robot, the program will prompt you to confirm the command by pressing

The program file will now be written on the disk and you will be returned to the copy of the file which is still in the memory of the Apple.

To clear the memory and load a file into it which was previously saved on disk, follow these steps:

1. Press

   This will display the Source Code Catalog and the word 'LOAD' will appear on the left side of the screen.

2. Use the space bar to position the cursor next to the name of the desired robot and press

3. The file will now be loaded into the memory of the Apple, and the first portion of it will appear on the screen.

If you have followed the instructions to this point, you should have:

1. Created a new source code file in the memory of the text-editor.

2. Saved the file to disk.

3. Loaded a new file into memory.

The Cursor Mode

You are now ready to perform the second available option when the text-editor has been loaded, which is editing the source code file. When editing source code you will use the cursor mode to delete text at the cursor, move the cursor around in the text, adjust the position of the text on the screen, load source code files, and save source code files. These functions are described below:

1. Cursor Movement

The cursor can be moved to any location in the file by using the five keys on the right side of the keyboard.

A) The RETURN key moves the cursor up one line

B) The left and right arrow keys move the cursor left and right one character

C) The slash (/) key moves the cursor down one line

To move the cursor all the way in any direction on the screen, push the ESC key and then the direction key.

Once you have positioned the cursor where you want it, there are several options. Either exit to the add mode and write some text or stay in the cursor mode and use a cursor function.

2. Moving Text

There are also methods of moving the text itself, when in the cursor mode. The direction, in which the text moves, is set by pressing the '+' key (a forward direction) or the '-' key (a backward direction) prior to pressing the L, P, or A keys.

A) The L key will move the text up or down one line

B) The P key will move the text up or down one full page

C) The A key will set the text in continous scrolling motion.

You can move to the beginning or end of the text by pressing the ESC key first and then the '-' key or the '+' key respectively.

3. Deleting Text

This function deletes text from the screen and the memory of the Apple, but not from the disk. There are three methods of deleting text:

A) CTRL-D : Any character may be deleted by positioning the cursor over the character and pressing CTRL-D.

B) CTRL-G : Any line, or portion of a line, may be deleted by positioning the cursor over a character and pressing CTRL-G. This will delete the character and the rest of the line that follows it.

C) ESC CTRL-Z : All of the text, presently in the text-editor, may be deleted by pressing ESC and then pressing CTRL-Z. You will have to confirm the command by pressing the # (Shift-3) key. This protects against accidental erasures.

4. Block Operations

This function allows you to 'mark' a portion of the current source code, and then manipulate that 'block' to another place in the file. You must designate the beginning and the end of a block by placing block markers at those two points. To insert a block marker immediately after the cursor, press

and it will be represented on the screen by a flashing ')' sign. Only one block can exist in a file at any one time and any attempt to insert a block mark when a block has already been defined will result in this error message

BLOCK ALREADY MARKED

When a block is marked press

and the three block options will be displayed on the bottom of the screen :

(C)OPY

This is used to copy the marked block to other locations in the file. To copy a block, move the cursor to the location in the file where you want to insert the copy of the block. Press

and a copy of the original block will be inserted where the cursor is. This will not destroy the original block. The same block can be copied as many times as you wish.

(D)ELETE

This is used to delete the marked block. To delete a block press

and the block and its markers will be erased from the file.

(U)NMARK

This is used to remove the block markers from the text. To unmark a block press

and the markers will be erased and the text will not be affected. You may also remove markers with normal delete commands. Remember, these changes only exist in the memory and not on the disk.

5. Find Operations

To find all occurrences of a word or a phrase in a file, use the 'FIND' operation. Press

and the screen will prompt you with the following message

FIND:>

At this point, type in the word or phrase you wish to locate and press

The program will now locate the first occurrence of that word and will display it in the center of the screen. To find all subsequent occurrences press

and

You need not enter the word each time. The search will always begin at the current cursor location and search in the direction that the indicator in the lower left corner of the screen shows.

6. Printing Source Code Files

To print the source code on a printer, press

The screen will prompt you to type in the printer slot number. Once this has been done and the RETURN key has been pressed the text will print out.

7. Adding Text

Position the cursor to where you want to begin adding text and enter the add mode by pressing

You will now be able to add text as desrcibed earlier. To exit back to the cursor mode, press

8. Loading Files

To load a file into memory which was previously saved on disk, follow these steps:

1. Press

   CTRL-L

   This will display the Source Code Catalog and the word 'LOAD' will appear on the left side of the screen.

2. Use the space bar to position the cursor next to the name of the desired robot and press

   RETURN

3. The robot program will now be loaded into the memory of the Apple, and the first portion of it will appear on the screen.

9. Saving Files

To save files on disk, follow these steps:

1. Press

   The Source Code Catalog will now appear on the screen and the word 'SAVE' will appear on the left side of the screen.

2. To save the new text of the source code press the space bar until the cursor is next to the file name you wish to save the new source code in. If the file has no name, type one in. Press

3. The source code will now be saved on disk under the desired file name and you will return to the text-editor and a copy of the file which is still in the memory of the Apple.

10. Entering the Assembler

To save the current robot program and enter the assembler, press

This will save the text as it appears in the memory on to the disk and then exit the text-editor to the assembler. Before a robot can be assembled, it must have been given a name by being saved to disk.

SUMMARY OF EDITOR KEYS

Cursor Mode:

Moving keys

	+	Set forward direction
	-	Set backward direction
ESC	+	Move to end of text
ESC	-	Move to start of text
	A	Sets text in continuous scrolling motion
	RETURN	Move cursor one line up
ESC	RETURN	Move cursor to top of page
	/	Move cursor one line down
ESC	/	Move cursor to bottom of page
	<-	Move cursor one space left
ESC	<-	Move cursor to left end of line
	->	Move cursor one space right
ESC	->	Move cursor to right end of line
	P	Move text up or down one full page
	L	Move text up or down one line

Text deleting keys

CTRL-D	Delete the character at the cursor
CTRL-G	Delete the line at the cursor
ESC CTRL-Z	Delete the whole file

File handling keys

CTRL-L	Clear memory and load a source file
CTRL-S	Save text as a source file
CTRL-R	Saves current file and enters assembler

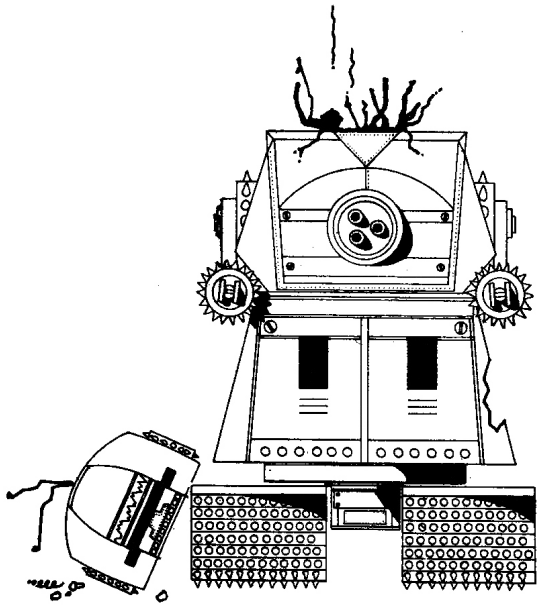
Control keys

ESC CTRL-Q	Exits to main menu
CTRL-F	Executes FIND operation
CTRL-P	Prints file in memory
CTRL-V	Places block marker at cursor
ESC-V	Displays Block options:
	C copies marked block
	D deletes marked block
	U removes block markers

Add Mode:

Text adding keys

CTRL-A	Start adding text
ESC ESC	Stop adding text
<-	Backspace, erases as it goes
->	Moves text to the right
RETURN	Acts as a carriage return



THE ASSEMBLER

The assembler translates source code programs into robot-understandable object code. It also checks for errors in the source code and displays a message if one is found.

The assembler can be entered from the Main Menu by selecting option 2, or from the editor by pressing CTRL-R. If the assembler is entered from the editor, a robot source program is loaded and ready to assemble.

If the assembler is entered from the Main Menu, the following choice will be displayed:

```
DO YOU WANT TO:

1. ASSEMBLE FROM SOURCE CODE OR
2. LOAD AN ASSEMBLED ROBOT?

   (PRESS 1 OR 2)
```

Press

to access the catalog of robots. Select a robot by using the space bar and RETURN key as described earlier (see Exhibit 2 on page 10). When a robot has been selected, the following question will appear on the screen:

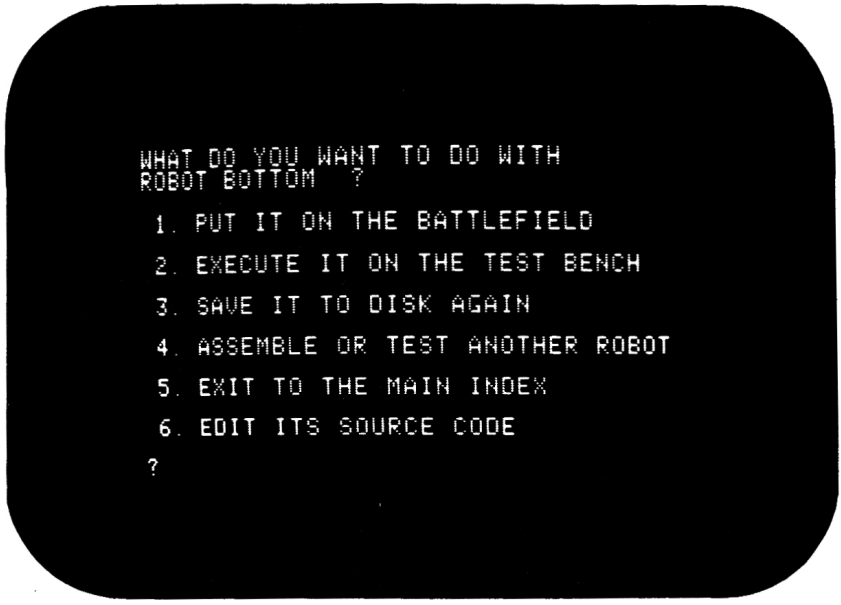
```
DO YOU WANT TO PRINT THE ASSEMBLY
( Y OR N ) ?
```

If you have a printer and want to print the assembly, press

and then enter the printer address. If you do not want the assembly printed out, press

The source code will be assembled and the corresponding object code will be displayed on the screen. Press RETURN to display the Assembler Menu.

Exhibit 5: The Assembler Menu



Option 1 - Exit to the battlefield with the assembled robot loaded and ready to challenge competitors.

Option 2 - Exit to the Robot Test Bench.

Option 3 - Save the object code to disk.

Option 4 - Load another robot source or object file.

Option 5 - Exit to the RobotWar Main Menu.

Option 6 - Exit to the text editor with the assemble robot's source code already loaded.

Assembly Errors

There are eight errors that the RobotWar assembler can detect. When the assembler detects an error it will display a message such as:

```
NO DATA FIELD IN LINE 27
     10 + TO C
          ^
```

The error message indicates the type of error, the program line number and the position in the line (^) where it occurred. Following are the possible error messages:

1. NO DATA FIELD - There is no register or number after a command.

2. UNKNOWN ITEM - You have tried to use a register or a label that is not defined.

3. LARGE NUMBER - You have tried to store a number greater than 1,024 or less than -1,024 into a register.

4. PROGRAM TOO LONG - Program is too big for the allotted program storage area. Programs have a maximum length of 256 object code instructions.

5. FATAL JUNK - You have included something that the computer cannot understand, like an illegal statement.

6. STORE IN NUMBER - You have tried to store a value in a number instead of a register.

7. RESERVED LABEL - You have tried to use a register name as a label.

8. NO PROGRAM CODE - There are no instructions in the program.

Object Code Exercise

The following pages list the object code's commands and registers and the translation of the sample robot's source code. Using the list and the two codes for the sample robot, compare and identify the source code and its object code translation. It will be very useful to understand the object code when learning to use the test bench in the next chapter.

LIST OF OBJECT CODE INSTRUCTIONS

INSTRUCTION	ACTION
,	Load accumulator with next data item
IF	Load accumulator with next data item
+	Add next data item to accumulator
-	Subtract next data item from accumulator
*	Multiply accumulator by next data item
/	Divide accumulator by next data item
=	Skips the next command unless the accumulator is equal to next data item
>	Skips the next command unless the accumulator is greater than next data item
<	Skips the next command unless the accumulator is less than next data item
#	Skips the next command if the accumulator is equal to next data item
TO	Store accumulator in next data item
GOTO	Branch to the address given
GOSUB	Gosub to the address given
ENDSUB	Return from a subroutine

ASSEMBLY OF ROBOT SAMPLE

```
CODE  BUILDING
====  ========

SCAN
0       ,       AIM
1       +       5
2       TO      AIM
3       ,       AIM
4       TO      RADAR
LOOP
5       IF      RADAR
6       <       0
7       GOSUB   FIRE
8       GOTO    SCAN
FIRE
9       ,       0
10      -       RADAR
11      TO      SHOT
12      ENDSUB

CODE  STATISTICS
====  ==========

140   LETTERS
13    INSTRUCTIONS
3     LABELS
2     REFERENCES
```

THE TEST BENCH

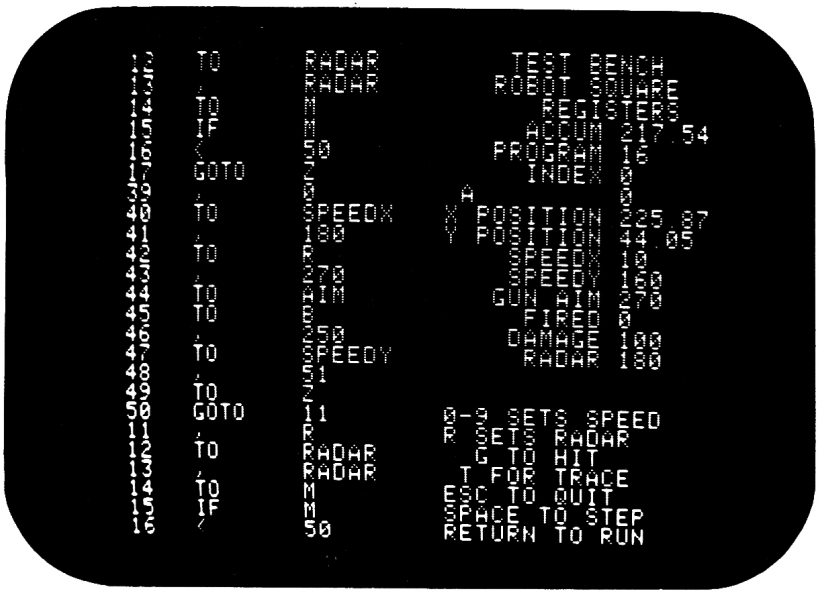
The test bench is a micro-computer simulator of a robot. With the test bench, you can monitor a robot's performance without actually putting it on the battlefield. This simulator will prove an important device, as you learn to debug robots, because it allows you to monitor the object code and the contents of the registers.

Load a robot into the test bench by selecting option 2 from the Assembler Menu or from the Main Menu.

Operating The Test Bench

As the test bench runs the program, each instruction (in object code) will appear on the left side of the screen as it is executed. On the right side of the screen are displayed the robot's position and register contents. Also shown are the instruction number being executed (program counter) and the accumulator.

Exhibit 6: The Test Bench



Controlling The Test Bench

The test bench can be interrupted by pressing the space bar. Press the space bar again to execute one more instruction. This can be useful when analyzing a program to see if it is acting as you had planned. Pressing RETURN will start the test bench running again. To change the speed of the test bench, press a number from 0 to 9.

Simulating Radar

Pressing the R key will cause the radar display to light up and the RADAR register will display a negative number to simulate an enemy robot in view. This will allow your program the opportunity to go into its "enemy spotted" routine.

Simulating Damage

Each time the G key is pressed, a random amount, up to 10%, will be subtracted from the DAMAGE register. This allows the program the opportunity to use its damage detection routine. The DAMAGE register will also indicate damage if the simulated robot crashes into a wall. The test bench will automatically stop when the DAMAGE register reaches 0.

Tracing Registers

The trace is used to check the contents of registers not normally displayed on the test bench. Press the T key to access the tracer. The test bench will stop, and the following question will be displayed:

NAME REGISTER TO TRACE?

Enter the name of the register you want to trace and press

RETURN

The test bench will continue, with the contents of the traced register displayed on the line above 'X POSITION'.

Exiting The Test Bench

To exit from the test bench, press

ESC

and you will return to the assembler menu.

STORING ROBOTS

There is a limited amount of space on the RobotWar disk to store robot files. However, robot files can be transferred to and from auxiliary storage disks.

Auxiliary storage disks are used only to store robot files. Robot files on auxiliary disks must be transferred back to the RobotWar program disk befor they can be tested, assembled, edited, or battled.

Initializing The Disk

Robot files can be saved only on auxiliary disks which have been initialized by the RobotWar program. To initialize a disk, select option 5 from the main menu. Insert a fresh disk when prompted and press RETURN. The initialization process will take about two minutes. When the process is complete, the disk will be initialized and the program will prompt you to remove the storage disk and re-enter the RobotWar disk.

Storing And Retrieving Source Code

To store a robot's source code file on an auxiliary storage disk, access the text-editor and load the desired file into the Apple's memory. Insert the initialized storage disk and press

CTRL-S

Type the name of the file you want to save and press

When the disk drive stops operating the file will be stored on the disk. Remove the storage disk and insert the RobotWar disk.

To transfer a source code file from an auxiliary disk back to the RobotWar disk: access the text-editor, insert the auxiliary disk and press

When the catalog is displayed use the standard selection process to load the desired file into memory. Then insert the RobotWar disk and save the file using a CTRL-S.

Storing And Retrieving Object Code

An object code file can be stored onto a storage disk only from the assembler menu. Insert the storage disk and press

When the menu re-appears on the screen, the object code will have been stored. Remove the storage disk and insert the RobotWar disk.

To transfer an object code file from a storage disk to the RobotWar disk: select option 2 from the main menu or option 4 from the assembler menu to display the following question:

DO YOU WANT TO:

1. ASSEMBLE FROM SOURCE CODE OR
2. LOAD AN ASSEMBLED ROBOT?

(PRESS 1 OR 2)

Then insert the storage disk and press

Use the standard selection process. When the assembler menu appears on the screen insert the RobotWar disk and select option 3 to save the file.

DELETING FILES

Any file can be deleted from a disk by using Applesoft Basic. Access the main menu and select option 6. Then insert the disk and type

CATALOG

and press

RETURN

This will display the names of the files stored on that disk. Source code files are identified by an 'S:' preceding the file name.

To delete a source code file, type

DELETE S:(file name)

Or, to delete an object code file, type

DELETE (file name)

and press

RETURN

The file will be deleted from the disk. To return to the RobotWar program insert the RobotWar disk, type

PR#6

and press

RETURN

ROBOTWAR KEY SUMMARY

The following pages describe the keys and their functions in the different sections of RobotWar.

From the Main Menu

1	Exit to the battlefield
2	Exit to the assembler
3	Exit to the text-editor
4	Turn battlefield sound on or off
5	Initialize an auxiliary storage disk
6	Exit to Applesoft Basic
7	Schedule a match
8	Run a scheduled match

From the Battlefield

RETURN	Start the battle
ESC	Stop the battle and return to main menu
Q	Turn the sound off
S	Turn the sound on

From the Assembler Menu

1	Exit to the battlefield
2	Exit to the test bench
3	Store the assembled robot on a disk
4	Exit to the assembler
5	Exit to the main menu
6	Exit to the text-editor

From the Assembler

Space Bar	Stop the assembler or move it one step
RETURN	Start the assembler operating again
0 - 9	Adjust the speed at which the assembler is scrolling

From the Test Bench

Space Bar	Stop the test bench or move it one step
RETURN	Start the test bench operating again
0 - 9	Adjust the speed at which the test bench is scrolling
R	Simulate radar
G	Simulate a shell hit
T	Trace a register
ESC	Exit to the assembler menu

From the Text-Editor

CTRL-A	Enter the add mode
ESC ESC	Enter the cursor mode

In the Add Mode

RETURN	Start a new line
<-	Backspace and erase one character
->	Add a space
Character	Add that character at the cursor
ESC ESC	Exit to the cursor mode

In the Cursor Mode

Cursor Moving:

RETURN	Move up one line
ESC RETURN	Move to top of page
<-	Move left one character
ESC <-	Move to the left end of the line
->	Move right one character
ESC ->	Move to the right end of the line
/	Move down one line
ESC /	Move to the bottom of the page
ESC +	Move to end of text
ESC -	Move to beginning of text

Text moving:

+	Set a forward direction
-	Set a backward direction
A	Start continuous scrolling motion in set direction
P	Move text one full page in the set direction
L	Move text one line in the set direction

Deleting text:

CTRL-D	Delete the character at the cursor
CTRL-G	Delete from cursor to the end of the line
ESC CTRL-Z	Delete entire file in memory

Control keys:

ESC CTRL-Q	Exit to main menu
CTRL-P	Print text in memory
CTRL-A	Exit to the Add Mode
CTRL-S	Save the file in memory to disk
CTRL-L	Load a file from disk (clears current file from memory)
CTRL-R	Save the file in memory to disk and exit to the assembler
CTRL-F	Start FIND operation
CTRL-V	Insert block markers
ESC-V	Display block functions: C -copies the marked block at the cursor D -delete the marked block U -remove the block markers

MORE FINE MUSE SOFTWARE FOR YOUR APPLE COMPUTER

WORD PROCESSING

Super-Text 40/80: The ultimate deluxe word processor for the Apple Computer featuring 80 column screen, Math Mode, and Split Screen. **$175.00**

Super-Text 40/56/70: The best features and best value in word processing, no extra hardware required! **$125.00**

BUSINESS

Form Letter Module: Send a personalized letter to everyone on your mailing list with Form Letter Module! Use with Super-Text for super efficiency. **$59.95**

Address Book: Store 700 names and addresses, then print envelopes and mailing labels! Use with Form Letter or alone. **$49.95**

Data-Plot: Create and include charts and graphs right in your reports! Four different kinds of charts available. **$59.95**

GAMES/EDUCATION

U-Draw II: You'll be amazed at what you can draw! Have hours of fun and learning. **$39.95**

Robotwar — The Best Selling Game That Teaches Programming! Program your own robot and let him loose on the battlefield! Learn to program and have hours of fun. **$39.95**

The Voice — The #1 Talking Disk for The Apple! Learning is fun when your Apple talks back and it can with The Voice! Easy to use and a favorite for kids of all ages. **$39.95**

Three Mile Island: Take charge of a nuclear reactor in Three Mile Island! **$39.95**

EDUCATIONAL

Elementary Math Edu-Disk: Math is easy when you learn with color pictures and demonstrations! Keep score and test your skills with Elementary Math. **$39.95**

Appilot II Edu-Disk: What could be more fun than lessons that talk, move, and interact with the student! Appilot II makes a game out of learning. **$99.95**

GAMES

ABM: Enjoy missile madness with ABM! Can you save the East Coast from enemy attack? **$24.95**

Castle Wolfenstein — The #1 Best Selling Game in America! There's nothing else like Castle Wolfenstein, the all time favorite arcade/adventure game! Can you escape with the Secret War Plans? **$29.95**

Firefly: Fire up your Apple with Firefly, the challenging maze game! Can you find your way through the flytrap? **$24.95**

Frazzle: The exciting outer space game! Alien beasties surround and attack your Frazzle ship: save yourself from total destruction! **$24.95**

International Gran Prix: The most popular racing game for the Apple! Race your car through all the courses at the International Gran Prix! **$29.95**

The Cube Solution: Master the cube with The Cube Solution, the enjoyable way to deal with the common cube headache. **$24.95**

The Best of Muse: A fantastic value — 10 great MUSE games on 1 disk! **$39.95**

ONE OF A KIND

Know Your Apple: Learn about your amazing Apple® computer with animated screen graphics, music, and voice. Know Your Apple is a must for every Apple owner! **$34.95**

MUSE Software • 347 N. Charles St. • Baltimore, MD • 21201
301-659-7212

Apple is a registered trademark of Apple Computer, Inc.

MUSE SOFTWARE™

347 N. CHARLES STREET
BALTIMORE, MD 21201
(301) 659-7212

PRINTED IN U.S.A.