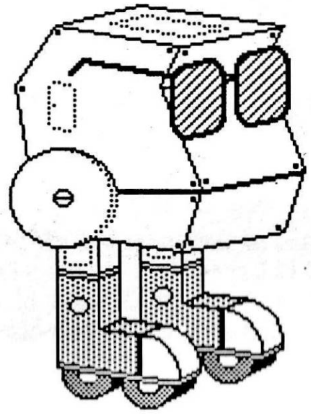
BRAINPOWER INC.
24009 Ventura Boulevard
Suite 250
Calabasas, California 91302

LEARNING ADVANTAGE EDITION

Ready when you are.



CHIPWITS™

by Doug Sharp and Mike Johnston Version 1.0

for the 64k Apple II Family

Acknowledgments

The development of software is a task that requires the participation of many people from many disciplines. The design and programming of ChipWits was done by Michael Johnston and Doug Sharp. Sherwin A. Steffin served as project manager. Joseph Koenka managed the quality assurance and testing phases of development with the help of David White, Eileen Spinner, and Adam Zalesny. ChipWits' documentation was written by Michael Cuneo. Additionally, BrainPower would like to thank all the ChipWit users who took the time to share with us the expertise they gained with the product.

BrainPower Inc.
24009 Ventura Boulevard
Suite 250
Calabasas, CA 91302
(818) 884-6911

Table of Contents

Table of Contents

Introduction

Robots are becoming a part of our everyday lives. Their presence has moved from science fiction to reality in only a few short years. Robots first made their appearance in steel mills and automobile factories. As they became more precise and intelligent in their operation, they made their way into new areas such as law enforcement and the exploration of dangerous environments. Now they are becoming available for the home in small but increasing numbers. The ChipWits system gives you an opportunity to experiment with programming a robot in a variety of missions. ChipWits also helps sharpen your logical reasoning and problem solving skills. As you develop these skills in your robots, you will develop them in yourself as well.

In summary, ChipWits is a robot-programming simulation and a tool for learning new problem solving skills.

THE MANUAL

This manual is designed to make it easy to use the ChipWits system. We begin by showing you around the system and letting you see a ChipWit in action. Next, you get a thorough lesson in ChipWit programming. This ends with a detailed analysis of a ChipWit. The final section of Chapter I provides details about the different ChipWit Environments (where missions take place) and about ChipWit mission statistics.

Chapter II shows how to use the advanced features of the ChipWit system to program more efficient and effective robots.

Appendix A offers four ChipWits for study and improvement. The manual ends with a glossary of ChipWit terms.

Throughout the manual, bold faced type is used in two ways. Usually, names are put in bold type to alert you to locate that thing on the Apple screen. Occasionally, bold type is used to refer to section titles in the manual.

Finding Your Way

The Table of Contents at the beginning of the manual will help you find where specific topics are discussed in the text. If you encounter terms unfamiliar to you, use the glossary at the back of the manual to understand how each is used in the ChipWits system.

HARDWARE REQUIREMENTS

To operate ChipWits, you need a 64K Apple II, II+ IIe, or IIc with a mouse, joystick, or Koala Pad™ connected. An Imagewriter or a printer connected with a Grappler™ graphic interface card allows you to print Workshop and Environments screens. **However, a printer is not required for using this system.**

INTRODUCTION

After you read this Chapter you will be able to:

1. Program ChipWits to prepare them for specific missions.

2. Enter any ChipWit into a mission and critically observe its performance.

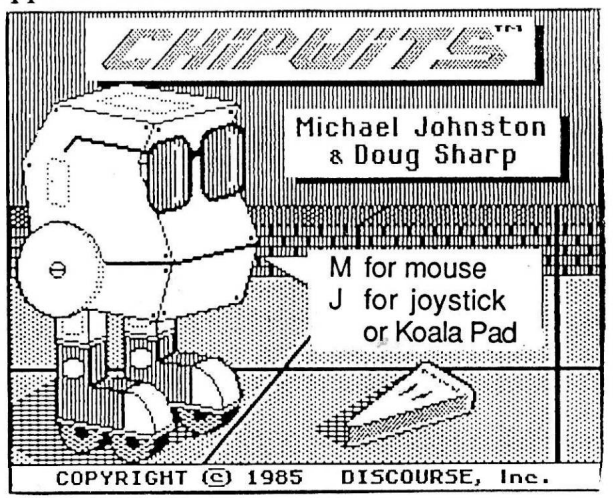
3. Modify a ChipWit, based on observing it on a mission, to improve its performance.

Start Up

Let's begin.

1. Insert the disk in the Apple. Turn the computer on.

2. The BrainPower identification screen will come up. After a few moments, the ChipWits opening screen appears. It looks like this:



Pull-Down Menus

The menu bar is the stripe across the top of the screen with the words **Options, Missions, Workshop,** and **Warehouse.** Below it, a large ChipWit requests you to press "J" if you are using a joystick or Koala Pad™, or "M" if you are using a mouse.

After the disk spins, you will see a new screen with an arrow cursor. The cursor's movement is controlled by either a mouse, joystick, or Koala Pad™. Position the arrow over the word **Options** on the menu bar. Now press your mouse button, joystick (button 0), or Koala Pad™ (button 0). A pull-down menu with the selections **Print, Sound Off, Statistics,** and **Step** appears. Let the button up, and the menu disappears.

To make a selection from a pull-down menu, simply click on the menu and continue holding the button down as you move the arrow over your choice. The choices become highlighted as you drag the arrow over them. When the selection you want is highlighted, release the button.

Pull down the Options now and select **Sound Off**. (Be careful not to select the first choice, **Print.** If your printer is not on, Print will cause the program to freeze.) Did you see Sound Off flicker after you selected it?

Since ChipWits wasn't making any sound, you will not hear anything different after you select Sound Off. But if you pull down the Options menu again, you'll see the Sound Off choice has changed to **Sound On**. This is a feature of pull-down menus: they change depending on what you select. Sound Off/Sound On is a toggle switch menu choice. That means it changes back and forth from one choice to another.

Another feature of pull-down menus is that they can mark choices you have made. Pull down the **Missions** menu. Do you see how **Greedville** is both underlined and checked? (We will see what these marks mean in a second.) For now, select the fourth choice on that menu, **ChipWit Caves**. Pull the menu down again. **ChipWit Caves** is underlined, although **Greedy** is still checked.

ChipWits Menu Bar

Now that you know how to make selections from pull-down menus, let's see what each of the choices on Chipwits' menu bar contains.

Warehouse Menu

The **Warehouse** is where ChipWits are stored. Click on its menu (located at the right side of the menu bar) and pull it down.

There is room for fourteen ChipWits in the Warehouse. The first two have been programmed already. They are named Greedy and C.W. Jr. Unprogrammed ChipWits are represented by numbers.

The ChipWit name or number that is underlined is the current ChipWit. This ChipWit will be used in the Workshop (where ChipWits are programmed) and the Environment (where ChipWits go on missions) until you select another ChipWit.

Workshop Menu

The Workshop is where ChipWits are named, programmed, and reprogrammed. Click on and pull down the **Workshop** Menu. The Cut, Copy, and Paste capabilities in the Workshop allow programming on one ChipWit to be transferred to another or shifted around inside the same ChipWit.

We will return to the Workshop soon.

Missions Menu

A mission is when a ChipWit explores an Environment. You evaluate and reprogram ChipWits based on how they do on missions. There are eight Environments for ChipWits to run missions in: Greedville, ChipWit Caves, Doom Rooms, Peace Paths, Memory Lanes, Octopus Garden, Mystery Matrix, and Boomtown. Each one offers the ChipWits (and their programmers!) different challenges, dangers, and goals.

Environment names in the menu can be underlined or checked in the menu. (If you selected ChipWit Caves earlier, it is now underlined. Greedville is checked, and the other six are neither underlined nor checked.) The underlined name represents the current Environment. This is the Environment in which the ChipWit missions will be conducted until you select another Environment. Checked names represent Environments for which the current ChipWit has been specifically designed.

The first two menu choices on the Missions menu are **Start Mission** and **Begin Series**. Start Mission puts the current ChipWit (the one underlined in the Warehouse Menu) in the current Environment (the one underlined in the Missions Menu) and begins a mission.

Begin Series runs an unending string of missions with the current ChipWit in the current Environment. Series gives you an accurate average score for your ChipWit if you let it run through enough missions.

Once a series begins, the menu choice Start Mission becomes End Mission and the choice Begin Series becomes Last Mission. You can terminate a series by selecting **Last Mission**. When the current mission ends, the series will be over. If you select **End Mission**, the series will immediately terminate. This is not the best way to end a series, however, since the score of the

abbreviated final mission will be averaged into the ChipWits performance statistics. (Statistics are kept automatically on each ChipWit. Details on ChipWit scoring and statistics follow.)

Options Menu

The Options menu has four selections: **Print, Sound Off/On, Statistics**, and **Step/No Step**.

Print prints out whatever is on the screen (provided you have an Imagewriter or a printer connected with a Grappler™ graphic interface card). The Print dialogue box requests you select your printer type and the slot it is plugged into. A click on the printer type selects it. A click on the slot number causes it to increase by one. **Cancel** removes the dialogue box. **OK** starts the printing.

Sound Off/On is a toggle switch selection that turns ChipWit noises off (if they are on) or on (if they are off).

Statistics displays the statistics for the current ChipWit in the current Environment. Statistics can be selected at any time--even during a mission. The program pauses, however, when the statistics are on the screen. Select Statistics. Notice that ChipWits keeps track of the number of missions the current ChipWit has run in the current Environment as well as the high score and average score of all missions it has run there. To remove the Statistics box from the screen, point the arrow at the letters **OK** in the box's bottom right corner and click.

ChipWits automatically updates the Statistics record each time a ChipWit runs a mission, until you reprogram the ChipWit. At that point you have a new ChipWit and a new Statistics file.

Step, the final choice, will be discussed in the next section when we watch the ChipWit in a mission.

A Trial Mission

Let's see what a ChipWit does in an Environment on a mission. If you have not already done so:

1. Insert the ChipWits disk in the disk drive and turn the computer on.

2. Wait for the BrainPower screen to come up. A few seconds later, a screen with a large ChipWit appears. Click the button on your mouse, joystick, or Koala Pad™.

Now you can begin a mission.

1. Pull down the **Warehouse** Menu. Is **Greedy** selected? If it isn't, select **Greedy**.

2. Pull down the **Missions** Menu. Is **Greedville** selected? If it isn't, select Greedville.

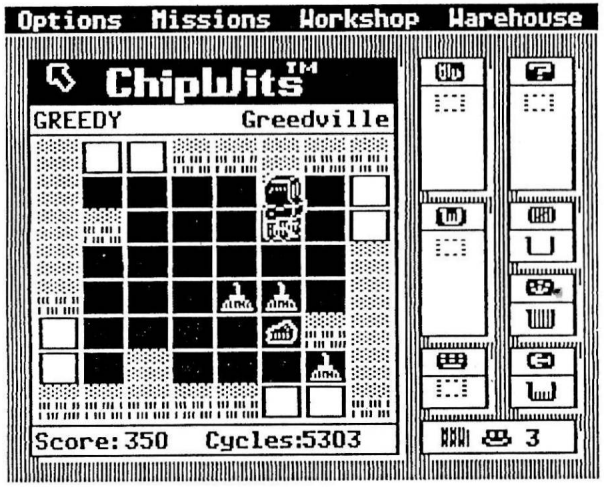
3. Pull down the **Missions** Menu again. This time, select **Start Mission.**

4. Before we begin our tour of the Environment, observe Greedy and his efforts in Greedville for as long as you like.

The next section explains the screen you are watching now. Leave the ChipWit running as you read. If you reselected the Sound On choice and all the beeping and roller-skating distracts you, pull down the Options menu and select Sound Off. If the mission ends, select Start Mission (from the Missions Menu) again.

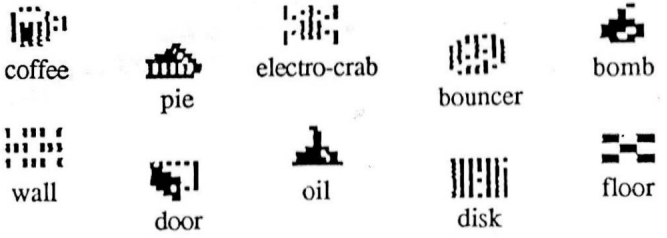
INSIDE AN ENVIRONMENT

The large window on the left displays the ChipWit on a mission in the Environment. At the top of the window, both the ChipWit and the Environment are identified. (You should be watching **Greedy** in **Greedville**.)



Each Environment consists of a series of rooms. The rooms are bounded by walls and a floor. Walls are constructed in the middle of rooms in some Environments. Doors are openings in the outer walls. When a ChipWit reaches a door and moves through it, a new room is automatically displayed.

Contained within the rooms are a variety of Things. ChipWit Things are oil cans, pie, coffee, electro-crabs, bouncers, bombs, and disks. Parts of the Environment (doors, walls, and the floor) are also considered Things by the robots.



While the ChipWit is on a mission, two running counts are visible in the Environment window. In the lower left corner of the window is the Score and in the lower right is the Cycles count. The ChipWit receives points added to its score whenever it acquires (eats) an oil can or disk, or zaps a bouncer or an electro-crab. The number of points it receives depends on which Environment it is in. (Details follow in the section titled **Environments**.)

Cycles represent the ChipWit's lifespan. At the start of a mission, the ChipWit is assigned a fixed number of cycles. The cycles are expended based on the ChipWits activities. When the cycles count reaches 0, the mission is over. (Details on cycle allotment and expenditure in each of the Environments follow in sections titled **Environments** and **Some Vital Statistics**.)

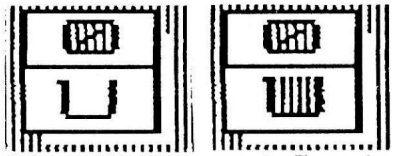
Status Meters

As you watch your ChipWits perform, you will want information concerning their health. This is provided in the four small meters found on the right of the Environment screen. The meters tell you how much damage your ChipWit has sustained, how much fuel your ChipWit has, what the last Keypress Argument was, and what the current value in the ChipWit's Range Finder is. Each is pictured and explained below.

Damage Meter

ChipWit damage is caused by the robot bumping into anything in the Environment or trying to pick up walls, bouncers, electro-crabs, or bombs. The Damage Meter (illustrated below) shows the amount of damage your ChipWit has sustained on its current mission. When the beaker under the icon is empty, it means the ChipWit has sustained no damage. When the beaker is full, it means the ChipWit has been completely damaged and the mission is over!

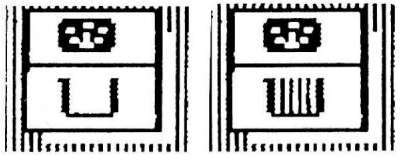
ChipWit damage is not repairable. One of the goals of ChipWit programming is avoiding damage. (Damage figures are detailed in the section **Some Vital Statistics**.)



The Damage Meter: No Damage & Completely Damaged

Fuel Meter

The ChipWit begins each mission with a full fuel tank. This is indicated by a full beaker under the Fuel Meter (illustrated below). Fuel is expended in two ways. Each action that the ChipWit makes requires fuel. (The amount of fuel each action requires is detailed in the section **Some Vital Statistics**.) The ChipWit also loses fuel when it is attacked by an electro-crab.



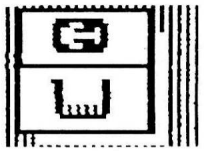
Fuel Meter Empty & Full

The ChipWit refuels by eating pie or drinking coffee. Another goal of ChipWit programming is to make sure the robot has enough fuel. If it does not find and acquire pie and coffee, it will run out of fuel and end the mission.

The Range Finder

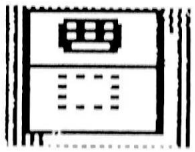
After any LOOK command, if the ChipWit has seen something (wall, door, floor, electro-crab, bouncer, bomb, pie, coffee, disk, or oil can), the object's distance from the ChipWit is registered in the beaker of the Range Finder. A full beaker means 7 floor tiles, an empty beaker means 0. Graduations in the beaker each represent two tiles.

Use of the Range Finder is discussed in Chapter II. The Range Finder looks like this:



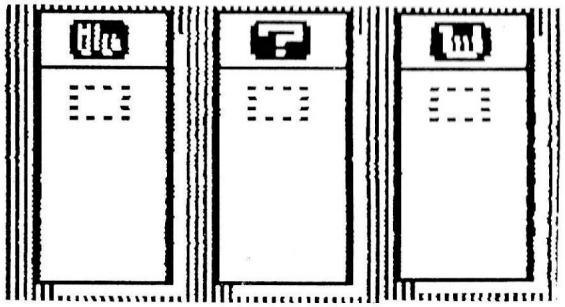
Keypress Icon

Keypresses are discussed at the beginning of Chapter II. The Keypress icon looks like this:



Memory Stacks

Above the Status Meters on the right of the Environment screen are the **Memory Stack Registers**. The three Registers show the top items in the Move, Number, and Thing Stacks. The roller skate above the Register at the top left labels that stack as the Move Stack, the question mark above the stack to its right labels it as the Thing Stack, and the beaker above the stack below the Move Stack labels it as the Number Stack.



Stacks are lists that you can request the ChipWit make of three types of items: moves (the ChipWit makes), Things (the ChipWit sees, touches, smells, zaps, or eats), and numbers (that the ChipWit can get from a multitude of sources). These lists (or stacks) are made and used by the ChipWit on a first item in, last item out basis. Displayed in the three on-screen registers are the top three items in each of the three memory stacks. (These are the last items that have been put into the stacks and the first, therefore, that will come out.) Each memory stack has room for 200 items.

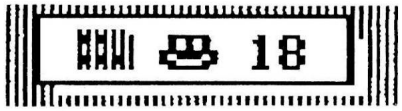
The use of stacks is discussed fully in Chapter II.

The Debug Monitor

The **Debug Monitor** is in the bottom right corner of the Environments screen.

ChipWit programs, as we will see, are made up of panels (groupings) of chips. The panels are identified by letters (with the exception of the Main Panel which is identified by a ChipWit face) and the chips on them are identified by numbers. Numbers and letters identifying the chips and panels that are currently executing are shown in the Debug Monitor while the ChipWit moves through its mission. This lets you determine where your ChipWit program needs reprogramming.

As you watch Greedy in Greedville, you see three pieces of information in the Debug Monitor.



The icon on the left represents a panel. It is followed by a ChipWit face to its right. This means the executing panel is the Main Panel. To the right of the ChipWit face is a constantly changing number. This is the chip on the Main Panel that is executing at that moment.

Since the action (both of the robot and the running chip identification) is fast, you may wish to slow it down to follow what is happening. To do this, pull down the Options menu and select **Step**. Now the robot will execute a chip only if button 0 of the joystick or Koala Pad™ or the button on the mouse is pressed. To resume full speed, return to the Options menu and select **No Step.** (This is the other half of the toggle selection Step/No Step.)

When we look at the Workshop in the next section, the concepts of chips and panels will become clear.

Summary

Each Environment has distinct goals for the ChipWit (the mission). In the easier Environments, the ChipWit is faced with minor dangers like running out of energy or being damaged by running into Things. In more difficult Environments, ChipWits face hostile Things: electro-crabs, bouncers, bombs, and lack of fuel sources.

The Status Meters and the Memory Registers give you information about your ChipWit's internal condition during the mission. The Debug Monitor lets you diagnose the problems in your ChipWit's program. You can slow the execution of the program down to your own pace (step), and see exactly how your programming has made the robot behave.

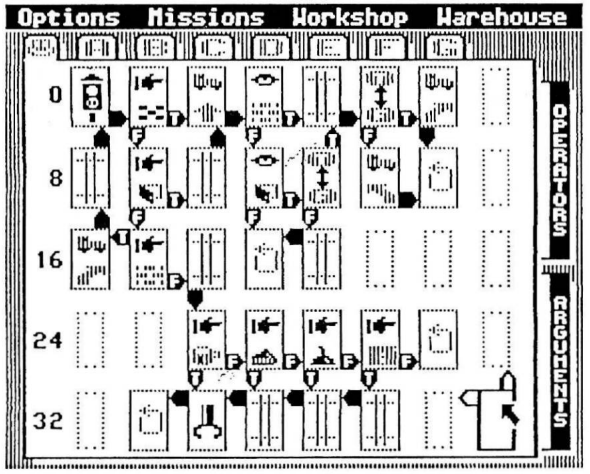
The next section teaches you how to program a ChipWit. This is the fun part-- what ChipWits is all about!

THE WORKSHOP

If you are reading this documentation through at the computer, you should still be watching Greedy scoot around in Greedville. We will now be leaving the Greedville Environment and entering the Workshop where Greedy and all other ChipWits are programmed and reprogrammed.

Into the Workshop!

Enter the Workshop by pulling down the Workshop Menu and selecting the first choice: **Enter**. Do this now. If you were running Greedy, you are now looking at Greedy's Main Panel. It looks like this:



If you are looking at the panel of another ChipWit (the name of the ChipWit you are viewing is underlined in the Warehouse menu), switch to Greedy by going to the Warehouse Menu and selecting Greedy. If you are looking at Greedy, but are seeing a blank panel, click on the ChipWit face in the tab to the left of the letter A.

A Workshop Tour

Before we learn how ChipWits are programmed, let's take a quick look at the programming tools you will use. These tools include: panels, chips, Operators, Arguments, and true/false wires. (You might want to select Sound On from the Options menu now. The Workshop is a pretty quiet place compared to an Environment during a mission.)

Panels

ChipWits are controlled by programs written in IBOL (Icon Based Operating Language). The programs are made up of chips set into panels. Each rectangle in the large window is a chip. The window itself is a panel.

There are two types of panels: the Main Panel and Sub-Panels. The Main Panel controls the overall flow of the program while the seven Sub-Panels execute only when called by the Main Panel.

You can determine which panel you are working on by the Main Panel/Sub-Panel Identification below the menu bar. There you see tabs containing the ChipWit face followed by the letters A through G. The face represents the Main Panel while the letters are the different Sub-Panels. You can tell which panel is being displayed by which tab is connected to the panel. Choose a panel for display merely by clicking on its letter (or the face for the Main Panel).

If you click a on letter when Greedy is in the Workshop, you will see an empty panel. This is because Greedy is programmed only on the Main Panel. Notice that all panels are identical in size and shape. Each has space for forty chips. Chip number 0 in each panel is a traffic light chip. (This is where each panel begins executing.)

Chips

All chips on a panel are assigned a number for the purpose of identifying them in the Debug Monitor. The numbering goes left to right, and top to bottom. Chips are numbered 0 (the stoplight) through 39 (the lower right corner). These are the numbers that are displayed in the Debug Monitor when a ChipWit is on a mission in an Environment.

The chips contain instructions to the robot. The instructions are comprised of Operators (verbs) and Arguments (objects) which we will discuss in a moment. The chips execute in a sequence determined by the robot builder. This sequence is set by the position of the chips in relation to one another and by true and false wires on the chips.

Operator/Arguments

There are two menus on the side of the Workshop screen. These are the **Operators** menu and the **Arguments** menu.

The Operators menu allows you to choose Operators for each chip. Operators are the equivalent of verbs: Look, Touch, Move, etc. This menu is a pull-across menu on the right side of the Workshop screen. Click on the word Operator there and it will appear as long as you hold the button down. To see how this menu works:

1. Click on the empty chip in the lower right hand corner of the screen. Its borders are heavily outlined showing it is the active chip (the one being programmed).

2. Pull out the Operators Menu, hold your button down, and position the point of the arrow on the icon for Look. It is a picture of an eye and looks like this:

3. Let go of the button. If the arrow was positioned correctly, the eye is now in the top of the active chip.

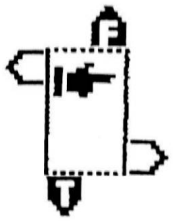
After you have selected the Operator LOOK (a verb), you must complete the chip with an Argument. Arguments are the objects of the Operators. The Arguments Menu, another pull-across menu, is below the Operators Menu. Items appear in the Arguments Menu only after an Operator is selected. The

Arguments in this menu are those available for the selected Operator. In some cases, Operators do not take an Argument. In those instances, the Arguments menu remains blank.

The Arguments menu works the same way as the Operators menu. Select items by clicking on them. Arguments fill in the bottom of the active chip when selected.

True/False Wires

Select a chip away from the corner (by clicking on it) and put both the FEEL Operator and an Argument in it. Notice that after you selected your chip, four wires appeared on the the sides of the chip. After you select an Operator, two of the wires fill in. One wire has the letter T in it and the other has the letter F. The letters stand for True and False. They look like this:



These wires control the flow of the program from one chip to another. Click on the true wire. If you have the sound on, you hear a beep.

Now click on an outline on another side of the chip. The true wire moves to that side. The wires are as important as the commands (Operators and Arguments), and you will be moving them around a lot.

Moving and Deleting

You can clear chips of Operators, Arguments, and true/false wires by clicking on them, holding the button down, and dragging them off the top or sides of the panel.

You can edit chips merely by clicking on them (to make them active) and selecting new Operators and Arguments.

PROGRAMMING A CHIPWIT

Now our tour of the Workshop is over. You know about all the tools of ChipWit programming and are ready to begin learning how to use them.

Programming Goals

The fun and the challenge of the ChipWit system is programming your ChipWits to master the Environments in which they are placed. What does "master" mean? Essentially, mastering an Environment means gaining the maximum number of points there. Points are gained or lost depending on what the ChipWit eats and zaps. Points are counted on the screen during every mission (remember seeing them running in the lower left when Greedy was in Greedville?). They are the gauge by which you measure the success of your robot building efforts.

COMMUNICATING WITH CHIPWITS

Giving instructions to a ChipWit (programming it) is like talking to a small child. If you were teaching either a ChipWit or a child to set the table, you would give it a clear sequence of instructions. You would watch what the child (or the robot)

does and give additional instructions based on their performance. You might begin:

Command Sequence	Resulting Actions
"Look in the cupboard for the plates."	The child finds the plates.
"Bring them to the table."	Child brings them.
"Now look for the glasses."	Child cannot find glasses.
"Look in the cupboard on the left."	Child now finds glasses.

Notice that this example of programming can be broken into three distinct components: telling the child what to do (The Instruction Set), telling it when to do it or giving it the commands in the right order (The Sequence of Instructions), and checking the resulting action to see if it was able to execute the command (Conditional Testing).

These are three basic elements of programming. Let's take a look at each to see how they work with the ChipWit tools: Operators, Arguments, and wires.

The Instruction Set

Each Operator represents an action you are commanding the ChipWit to perform. Operators are the same as verbs in imperative sentences. For an example, take the command: **[You are to] LOOK [for] PIE.**

When the sentence is spoken or written in English, the bracketed words are used to make the meaning clear. In an imperative (command) sentence, the subject, "You," is often left unsaid. Thus, we are used to hearing the sentence as: **LOOK [for] PIE.** The preposition **[for]** makes listening to the sentence

easier and more comfortable for people. ChipWits, being computer robots, do not require prepositions for communication. Thus, we can edit the sentence to two words: **LOOK PIE.**

In ChipWit (and programming) terminology, the verb **LOOK** is an Operator and the noun **PIE** is an Argument. The two combine to form a command or an Instruction Set.

Operators/Arguments

As you have already seen, IBOL, the ChipWit programming language, uses icons, or symbols, to represent Operators and Arguments. The symbol :

stands for the Operator **LOOK**. The symbol:

stands for the Argument **PIE.**

The following catalogue:

1. lists and illustrates each of the ChipWit Operators.

2. tells what each Operator commands the ChipWit to do, and

3. tells what Arguments each Operator uses (what can be the object of the command).

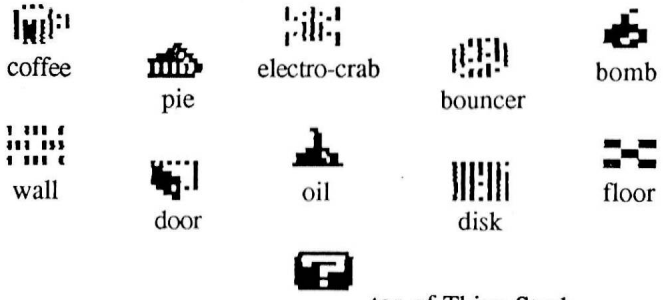
The Operator LOOK

LOOK commands the ChipWit to look directly ahead in a straight line for a specific Thing. (A ChipWit can see across a room.) Whatever Argument you use with this Operator is the

Thing the ChipWit will look for. If the Argument is seen, the program continues through the LOOK chip's true wire to the chip connected to that wire. If the Thing is not seen, the program continues through the LOOK chip's false wire to the chip connected to that wire.

Whenever LOOK is used and the ChipWit sees the Thing, the number of tiles between the ChipWit and the Thing is placed in the Range Finder. Chapter II gives some ideas about how to use this feature with the Number Stack.

The Operators for LOOK are all the Things that might be found in an Environment as well as the item at the top of the Thing Stack. They are:



The Operator SMELL

SMELL commands the ChipWit to smell an entire room for a specific Thing. Whatever Argument you use with this Operator is the Thing the ChipWit will smell for. If the Argument is smelled, the program continues through the SMELL chip's true wire to the chip connected to that wire. If the Argument is not smelled, the program continues through the SMELL chip's false wire.

The Operators for SMELL are all the Things that might be found in an Environment. For a listing of these Arguments, see LOOK above.

The Operator FEEL

FEEL commands the ChipWit to feel directly ahead one space (floor tile) for a specific Thing. Whatever Argument you use with this Operator is the Thing the ChipWit will feel for. If the Argument is felt, the program continues through the FEEL chip's true wire to the chip connected to that wire. If the Argument is not felt, the program continues through the FEEL chip's false wire.

The Operators for FEEL are all the Things that might be found in an Environment. For a listing of these Arguments, see LOOK above.

The Operator PICK UP

PICK UP commands the ChipWit to reach directly ahead one space (floor tile) for a Thing. PICK UP is used to acquire disks and oil for points, and pie and coffee for fuel. If the ChipWit attempts to PICK UP a bomb, it will be blown up. If it attempts to PICK UP a bouncer, electro-crab, or a wall, it will sustain damage.

There are no Arguments for the PICK UP Operator. The ChipWit will try to PICK UP whatever is on the tile directly in front of it. If nothing is there, it will reach just the same.

The Operator ZAP

ZAP commands the ChipWit to attack with its plasma beam anything directly ahead in the room. (A ChipWit plasma beam

can reach across a room. It will remove anything except walls and floors from an Environment.) ChipWit receives points only for ZAPping electro-crabs and bouncers. If it ZAPs a bomb, it will get itself blown up. If it ZAPs pie or coffee, it will not gain the fuel it would have had it picked it up. If it ZAPs disks or oil, it will not gain the points those Things are worth if they are picked up.

Like the PICK UP Operator, the ZAP Operator has no Arguments.

The Operator MOVE

MOVE commands the ChipWit to move either forward or backward one tile, or to rotate 45 degrees in either direction on the tile it occupies.

The Arguments for MOVE are four arrows indicating direction and rotation. They look like this.

MOVE has a fifth Argument that tells the ChipWit to execute the top move stored in the Move Stack. (Remember, stacks are lists that you can have the ChipWit make as it goes on a mission.) The Move Stack is discussed in Chapter II.

The Operator SUB-PANEL

SUB-PANEL commands the program to leave the Main-Panel and continue execution at the beginning of a specified Sub-Panel. Since Sub-Panels can be reached only from the Main

Panel, the SUB-PANEL Operator can only be used on the Main Panel.

The Arguments for SUB-PANEL are the letters A through G. They stand for the Sub-Panel requested.

SUB-PANEL is a command to the program rather than a behavioral command to the ChipWit

The Operator LOOP

LOOP commands the ChipWit program to go back to the beginning (the stoplight) of whatever panel it is executing and continue execution from there.

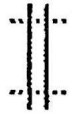
LOOP has no Arguments. It, too, is a command to the program rather than a behaviorial command to the ChipWit.

The Operator BOOMERANG

BOOMERANG commands the ChipWit program to go back to the Main Panel and continue execution with the chip after the Sub-Panel chip that sent the program away from the Main Panel. Since BOOMERANG returns the program to the Main Panel, it is only available for use on Sub-Panels.

BOOMERANG has no Arguments. Like the LOOP and the SUB-PANEL Operators, BOOMERANG is a command to the program, not the ChipWit.

The Operator JUNCTION



JUNCTION merely connects two chips with each other. It is used when the lay-out of your program necessitates a link between two chips that are one space or more apart. More than one JUNCTION chip can be strung together to cross large areas of a panel. JUNCTION has no Argument.

The Operator COIN FLIP

COIN FLIP generates a random choice for a branch from the chip's true or false wires.

The Operator SING

SING commands the ChipWit to sing a note. The note it sings is determined by a number (0 being the first pitch in the scale and 7 being the last, an octave higher).

The Arguments for SING are the numbers which determine the pitch it sings. They are the numbers 0 through 7 (as represented by the graded beakers), the Number Stack (which causes it to sing the top number there), the Fuel Meter (which causes it to sing the value there), the Range Finder (which causes it to sing the value there), and the Damage Meter (which causes it to sing the value there).

The Operator KEYPRESS

KEYPRESS checks to see if a specified key on the Apple keyboard is being pressed at the time the chip is executing. If the key is being pressed, the program continues through the KEYPRESS chip's true wire. If the key is not being pressed, the program continues through the false wire. KEYPRESS allows the programmer to step in and direct the flow of the program. KEYPRESS is discussed in Chapter II.

The Arguments for KEYPRESS are the letters of the alphabet and a blank square. The letters stand for specific keys you want the KEYPRESS to check for. The blank square will cause a true branch if no key is pressed and a false branch if any letter is pressed.

The Operator SAVE MOVE

SAVE MOVE commands the ChipWit to put a specified move (the Argument of the SAVE MOVE chip) into the Move Stack. This is the way the ChipWit makes the list of moves that constitutes the Move stack. Uses of the Move stack are discussed in Chapter II.

The Arguments for SAVE MOVE are the four directional move arrows.

The Operator COMPARE MOVE

COMPARE MOVE commands the ChipWit to examine the move on the top of the Move Stack and compare it to a specified move. If the moves are the same, the program branches through the COMPARE MOVE chip's true wire. If the moves are different,

the program branches through the COMPARE MOVE chip's false wire.

The Argument of the COMPARE MOVE chip represents the move you want the ChipWit to compare to the move at the top of the Move Stack. They are the four directional arrows and the empty square. The empty square represents the bottom of the Move Stack. If the Move Stack is empty, and you use this Argument, the chip will branch through its true wire.

The Operator SAVE THING

SAVE THING commands the ChipWit to put a specified Thing into the Thing Stack. This is the way the ChipWit makes the list of Things that constitutes the Thing Stack. Uses of the Thing Stack are discussed in Chapter II.

The Arguments for SAVE THING are the Things that are the Arguments for LOOK.

The Operator COMPARE THING

COMPARE THING commands the ChipWit to examine the Thing on the top of the Thing Stack and compare it to a specified Thing. If the Things are the same, the program branches through the COMPARE THING chip's true wire. If the Things are different, the program branches through the COMPARE THING chip's false wire.

The Argument of the COMPARE THING chip represents the Thing you want the ChipWit to compare to the item at the top of the Thing Stack. The bottom of the Thing stack is represented the same way as the bottom of the Move Stack.

The Operator SAVE NUMBER

SAVE NUMBER commands the ChipWit to put a value into the Number Stack. This is the way the ChipWit makes the list of values that constitutes the Number Stack. Uses of the Number Stack are discussed in Chapter II.

The Arguments for SAVE NUMBER are the numbers you want the robot to put into the stack. They are the numbers 0 through 7 (as represented by the beakers), the values from the Fuel and Damage Meters, and the current reading from the Range Finder.

The Operator COMPARE NUMBER: EQUAL?

COMPARE NUMBER: EQUAL? commands the ChipWit to examine the value on the top of the Number Stack and compare it to a specified value. If the values are the same (equal), the program branches through the COMPARE NUMBER: EQUAL? chip's true wire. If the values are different, the program branches through the COMPARE NUMBER: EQUAL? chip's false wire.

The Arguments represent the value you want the ChipWit to compare to the value at the top of the Number Stack. They are the values 0 through 7 (as represented by the beakers), the value from the Fuel and Damage Meters, the current reading from the Range Finder, and the bottom of the Number Stack (represented by the empty square).

The Operator COMPARE NUMBER: LESS THAN?

COMPARE NUMBER: LESS THAN? commands the ChipWit

to examine the value at the top of the Number Stack and compare it to a specified value. If the value in the Number Stack is less than the specified value, the program branches through the COMPARE NUMBER: LESS THAN? chip's true wire. If the value in the Number stack is greater than or equal to the specified value, the program branches through the COMPARE NUMBER: LESS THAN? chip's false wire.

The Argument of the COMPARE NUMBER: LESS THAN? chip represents the value you want the ChipWit to compare to the value at the top of the Number Stack. They are the values 0 through 7 (as represented by the beakers), the values from the Fuel and Damage Meters, and the current reading from the Range Finder.

The Operator INCREMENT

INCREMENT increases the value at the top of the Number Stack by one. Numbers increase from zero to seven and then wrap around (return) to zero. There must be a value in the Number Stack in order to INCREMENT.

The Operator DECREMENT

DECREMENT decreases the value at the top of the Number Stack by one. Numbers decrease from seven to zero and then wrap around (return) to seven. There must be a value in the Number Stack in order to DECREMENT.

The Operator POP

POP removes the top item from the specified stack.

The Arguments for POP are the Number, Thing, and Move

Stacks. They represent the stack you want the item removed from.

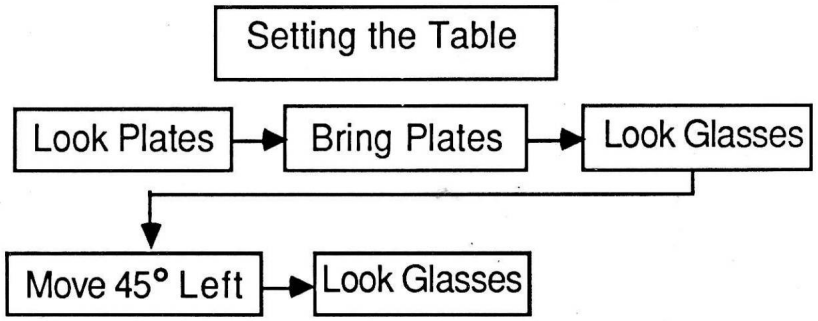
Instruction Sequence

You now know almost everything you need to tell a ChipWit what to do (to make up Instruction Sets for the robot). Let's see how we determine the order of our commands: The Sequence of Instruction.

Looking at the set of instructions for the child, you see that if our commands were programmed on chips, they would read like this:

1. LOOK PLATES
2. BRING PLATES
3. LOOK GLASSES
4. MOVE 45 degrees LEFT
5. LOOK GLASSES

Notice that the chips are programmed to perform in a specific order. ChipWit programming, like all computer programming, requires instructions to be placed in a sequence that allows the program to achieve its objectives. Programmers often use Flow Charts to help them figure out the best sequence (or flow) of commands. The example we have been looking at is represented in a simple flow chart below:



Each rectangle represents an Operator and Argument (one instruction set). The arrows point out the sequence of the instructions. As you develop your ChipWit programs, you may want to sketch out simple flow charts for yourself.

The Sequence of Instruction for ChipWits is determined by the true and false wires that are attached to almost every chip. These wires are attached to every chip that performs a Conditional Test (see next section). Conditional Testing determines, to a large degree, the actual flow of instruction in the ChipWit's program.

Conditional Testing

Conditional Testing is the third major component of programming ChipWits (the first two are Instruction Sets and Flow of Instruction). Conditional Testing is the process of checking the results of an Instruction Set and branching to a new Instruction Set based on those results.

There is an example of Condition Testing in the Instruction Set that we have been using. The child ran into a problem after step number 3:

1. LOOK PLATES
2. BRING PLATES
3. LOOK GLASSES
4. MOVE 45 DEGREES LEFT
5. LOOK GLASSES

After the child carried out Instruction 3, based on the results of that Instruction Set, we asked the child to change direction and look again.

What we were saying in instruction 3 was, **"[If you] LOOK [and find the] GLASSES, [then] BRING [the] GLASSES [to the table].** We provided an alternative in case the glasses were not seen. This alternative, instruction 4, was to **MOVE 45 DEGREES LEFT**, and (instruction 5) **LOOK GLASSES** again.

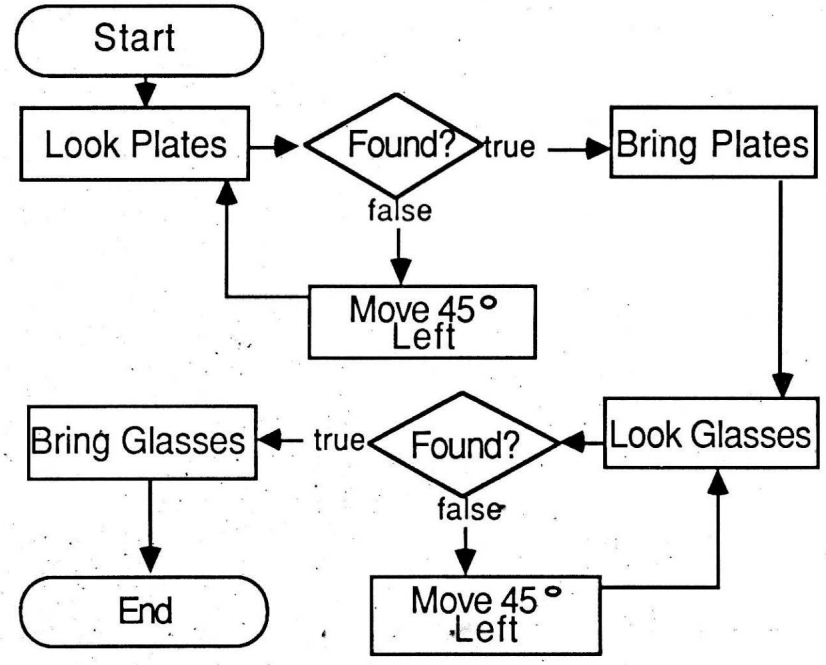
We can set up a general rule for Conditional Testing. That rule is:

IF [something is true or meets some condition you have set,] THEN DO [something]. IF [it is] FALSE, THEN DO [something different].

Programmers, who hate to use more words then they absolutely have to, have a shorter way of expressing the same rule:

IF (x)...THEN DO (y)...IF NOT (x)... DO (z).

A more detailed flow chart for the example we have been using (showing Conditional Testing), looks like this:



The diamond figure is the Decision Box. It represents the test determining if the condition asked for is true or false (if the command in the Instruction Set was executed). It determines what the next instruction set should be. In this case, since the test of LOOK GLASSES is false, the next instruction is to MOVE 45 DEGREES LEFT, followed by another instruction to LOOK for the GLASSES. If the result of the test had been true, the next instruction would be to bring the glasses.

The main difference between programming a ChipWit and instructing a child is that when you program the robots, you must anticipate the problems and situations they will face. When working with a child, you can jump in at any time and give new instructions. With the ChipWit, the Debug Monitor lets you see where your instructions need to be modified. To make the necessary changes, however, you must take the ChipWit out of the Environment and into the Workshop for re-programming. Flow charts, like the one above, will help you plan your program's Sequence of Instructions based on Conditional Testing.

Most chips test to determine whether specified conditions are being met. These chips have true and false wires. Notice how a MOVE FORWARD does not have true and false wires since the ChipWit will do exactly as you tell it (even if it moves into a bomb and blows up). Chips like MOVE FORWARD, ZAP, PICK UP, and SAVE MOVE have only one wire. They always continue out that wire.

SAVING A CHIPWIT

When you have been working on a ChipWit in the Workshop, YOU MUST SAVE IT EITHER BEFORE PUTTING ANOTHER CHIPWIT INTO THE WORKSHOP OR AN ENVIRONMENT OR TURNING THE COMPUTER OFF. Do this by selecting **Save** from the Workshop menu. A dialogue box that looks like this will appear.

Type the name you want to give your ChipWit in the rectangle. Click on each Environment for which you have designed the ChipWit specifically. (A second click on an Environment name removes an assignment made by mistake.) When you are finished, click on OK and your ChipWit will be saved to disk.

If you load another ChipWit (either into an Environment or the Workshop) or turn the computer off without saving changes made to the current ChipWit, the changes will be lost.

A CLOSE LOOK AT GREEDY

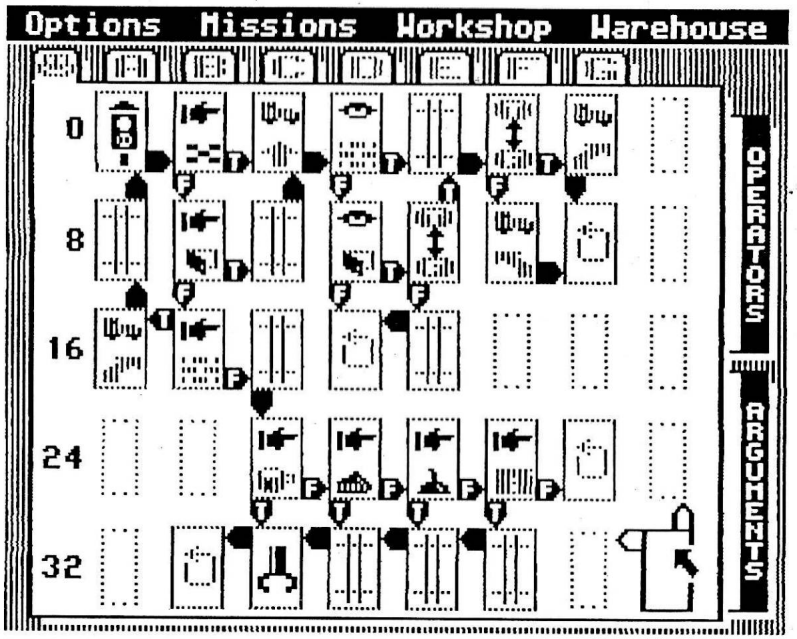
At this point, you are probably fired up and ready to build the ChipWit that ate New York! If you want to try just that, skip the next section and go straight to the section titled **SOME FINAL DETAILS.** Here you will find an explanation of several tools to make your time in the Workshop easier and a precise catalogue of what your ChipWits can expect in each Environment. What follows next is a chip by chip look at a

simple ChipWit: Greedy. Unless you are an experienced programmer, if you skip this part now you will want to return to it later.

Greedy: Life and Times

Greedy, a simple one panel ChipWit who was born and raised in Minneapolis, was destined by fate (actually, by a ChipWit builder) to spend his days in Greedville. Although you can place Greedy in any Environment, as we will see, he functions efficiently only in his adopted hometown, Greedville.

A printout of Greedy's Main panel is shown below so that, in the event you are reading this in the bathtub or an airplane, you will be able to see what we are referring to.



Greedy begins its exploration of Greedville by feeling for the floor (chip 1). If it doesn't feel the floor, it branches out its false wire on chip 1 and feels for a door (chip 9). If it does not feel a

door, it branches through the false wire (on 9) and feels for a wall (chip17). This sequence has determined that there is something in front of the robot (or else it would have felt the floor) and that it is not a fixed part of the Environment (a door or a wall). By the process of elimination, the robot builder has established that there is either oil, pie, coffee, or a disk in front of the ChipWit. (Greedville does not have electro-crabs, bombs, or bouncers.) Any of these four possible Things are desirable, and attainable through the PICK UP Operator.

But Greedy is not the most efficient ChipWit in the Warehouse. Rather than just grabbing, it goes through a sensing routine for coffee, pie, oil, and disk just as it did for floor, door and wall earlier. It wants to know what it is picking up! If a Conditional Test for one of these Things is positive (chips 26, 27, 28, and 29), the program branches through a true wire either to the PICK UP chip (34) or to a JUNCTION chip (35, 36, or 37) which connects to the PICK UP Chip. From the PICK UP Operator the program branches to a LOOP chip (33) and returns to the stoplight.

Notice that there is a LOOP chip attached to the false wire of the FEEL for DISK chip (30). There would be no instance in Greedville where this chip could be reached. But if Greedy were in Doom Rooms, it could be feeling an electro-crab. Thus, it would branch through the false wires of the complete identify Thing routine (chips 1, 9, 17, 26, 27, 28, and 29) without successfully finding a true wire.

We have now seen how Greedy reacts when it finds itself in front of a good Thing. Let's now see what it does when it finds itself in front of a wall, door, or an empty floor tile. We will see that Greedy's programmer has taken a casual attitude toward controlling the robots movement.

If Greedy senses a wall (chip 17), it rotates 45 degrees and LOOPS back to the stoplight. Regardless of whether it senses the floor or a door (via chips 1 or 9), Greedy goes to the same

movement sequence. (This is mandated by the JUNCTION chip 10 which funnels true wire responses from chip 9 to chip 2.)

This response consists of moving forward one tile (chip 2). Since Greedy is on a new space, it makes sure it isn't about to slam into a wall by looking for a wall. If it has moved in front of a wall, the robot runs the routine in chips 4, 5, 6, 13, and 14. This routine uses a COIN FLIP Operator to generate a random rotation movement. Greedy then loops and begins the panel again.

If Greedy does not see a wall, chip 3 branches to 11 and the robot looks for a door. If a door is seen, Greedy uses a COIN FLIP Operator to determine if it will loop to the beginning of the panel or run the routine used when a wall is sensed. Not only is the decision made randomly, but one of the choices of the random decision is a routine that is based on a random decision (chip 5).

An Analysis of Greedy

Greedy was programmed on the assumption that there were plenty of good Things to acquire in the Environment it would be operating in. This means its programmer did not concern himself with carefully controlling the movement of the ChipWit. Greedy feels for Things to acquire, and, if they are there, it never misses acquiring them. If it doesn't feel them, it moves randomly (usually under the control of COIN FLIP Operators: chips 5 and 12).

Improving Greedy

Assuming we are custom building Greedy for Greedville, we can edit out the routine it runs when it has determined there is a Thing in front of it. Chips 26, 27, 28, 29, 34, 35, 36, and 37 can be replaced by a PICK UP Operator in chip location 26.

Greedy's movement is a problem. Can you see how chips 3 and 11 cause the robot problems? Could the use of the Range Finder in conjunction with these chips improve Greedy? (See Chapter II for suggestions on Ranger Finder uses.)

Greedy moves in and out of rooms based on COIN FLIPS rather than whether the rooms have anything it may want. See if you can develop a routine for Greedy to (A.) realize when it is moving out of a room and (B.) smell the room it is leaving for disks. (Disks are worth the most points of all Things in Greedville as you will see when you read SOME FINAL DETAILS.) If there are disks remaining, have the robot turn around and find them.

As it stands now, Greedy is a good point producing ChipWit. Can you make it better and more elegant besides? The next section tells you a few more ChipWit details you'll need to understand the different Environments and it also shows some Workshop time savers.

There are four ChipWits in the Appendix in the back of the manual. C.W. Jr., another ChipWit, is on disk. Study these examples as we have done with Greedy. They all need improvement!

SOME FINAL DETAILS

This section has two parts. The first part shows you some editing features that ChipWits offers in the Workshop. These let you Cut, Copy, and Paste your work. The second section tells you about each of the eight ChipWit Environments: what your ChipWits should be prepared to find there and how much each Thing is worth. It also gives you specifics on ChipWit fuel, damage, and energy statistics.

Cut, Copy, Paste

Pull down the Workshop menu and select **Enter**.

The Cut, Copy, and Paste functions (from the Workshop menu) allow you to manipulate entire panels in the Workshop.

Selecting Cut will clear the panel on the screen and put that panel in memory. Simply select another panel by clicking on another letter (or the Main Panel identifier). When that panel comes on the screen, select Paste and the panel you Cut will be transferred to the new panel. You can load in another ChipWit from the Warehouse and paste the panel into that ChipWit as well. A Cut panel will remain in memory until another panel is either Cut or Copied. It may be Pasted in different locations as many times as desired.

Copy functions the same way as Cut does, except that the panel you Copy remains in place while also being copied into memory. This allows you to take an especially good panel you have developed for one ChipWit and place it in another as well. A Copied panel will remain in memory until another panel is either Cut or Copied. It may be Pasted in different locations as many times as desired.

When you leave the Workshop, any panel Cut or Copied in memory is lost.

Clear Panel

The sixth selection on the Workshop menu is **Clear Panel**. This does exactly that to the panel that is on screen at the moment. The panel is not saved into memory. It is gone forever. Clear Panel is a useful tool. So is a band-saw. Be careful.

Environments

There are eight ChipWits environments. All of them have pie and coffee for ChipWit refueling (no points are gained when these two foods are acquired). Each Environment has a different layout, number of rooms, population of Things, and a different set of point assignments for those Things. Here are the details.

Greedville

Geography and Characteristics: Four confusingly connected rooms filled randomly with only good Things. There are no dangers here for a ChipWit except running into and trying to pick up walls.

Inhabitants and their point values: Disks are worth 100 points and oil is worth 50 points.

Cycles allotted ChipWit upon entry: 6,000

ChipWit Caves

Geography and Characteristics: Eight connected rooms that spell out a word familiar to every ChipWit. The Caves are filled with dangerous electro-crabs which drain ChipWits' fuel. The crabs offer additional point opportunities for the robots, however.

Inhabitants and their point values: Disks are worth 100 points, oil is worth 50 points, and electro-crabs are worth 50 points.

Cycles allotted ChipWit upon entry: 10,000

Doom Rooms

Geography and Characteristics: How could there be any other number of rooms but thirteen in Doom Rooms? The Rooms are

filled with electro-crabs and bouncers-- more opportunities for points. But beware! Crabs drain fuel, and bouncers, when you run into them, inflict damage. There are no good Things (disk and oil) in the Rooms!

Inhabitants and their point values: Electro-crabs are worth 100 points and bouncers are worth 250 points.

Cycles allotted ChipWit upon entry: 10,000

Peace Paths

Geography and Characteristics: Peace Paths offers the ChipWit strategist a novel challenge. Your robot will be confronted by electro-crabs, but you must teach it to avoid them rather than ZAPping. You lose points by attacking your enemies! Point opportunities are disks and oil. The fifty-four rooms are not densely populated, so you might consider some sensing routines to make sure your robot doesn't tarry too long in barren fields.

Inhabitants and their point values: Disks are worth 150 points, oil is worth 20 points, and electro-crabs cost your ChipWit 30 points.

Cycles allotted ChipWit upon entry: 16,000

Memory Lanes

Geography and Characteristics: The ten room maze contains a regenerating disk at either end. Traveling back and forth allows you to garner this prize multiple times (the disk regenerates while you are at the other end of the maze). Along the way are both electro-crabs and bouncers as well as oil.

Inhabitants and their point values: Disks are worth 250, oil is worth 30, electro-crabs are worth 30, and bouncers are worth 60.

Cycles allotted ChipWit upon entry: 20,000

Octopus Garden

Geography and Characteristics: Eight hallways and forty-four rooms branch off of a central location. At the end of each hallway is a high point value Thing. Why would the Move Stack be helpful here?

Inhabitants and their point values: Disks are worth 250 points, electro-crabs are worth 10 points, and bouncers are worth 20 points.

Cycles allotted ChipWit upon entry: 20,000

Mystery Matrix

Geography and Characteristics: The Matrix consists of one hundred rooms, each with identically located doors. Discover the secret of the high-point pathway by having your ChipWit read the signposts. (Signposts? What signposts? I didn't know a ChipWit could read!)

Inhabitants and their point values: Disks are worth 250 points, oil is worth 150 points, electro-crabs are worth 20 points, and bouncers are worth 40 points.

Cycles allotted ChipWit upon entry: 30,000

Boomtown

Geography and Characteristics: The nine rooms in Boomtown are the most dangerous rooms a ChipWit will ever face. They are filled with bombs that blow ChipWits up if the robots so much as sneeze on them. You'll see a pattern to the way the bombs are laid out-- and you better teach your ChipWits to see it as well or else they'll never go far in Boomtown (except straight up in the air).

Inhabitants and their point values: Disks are worth 250 points and electro-crabs are worth 40 points.

Cycles allotted ChipWit upon entry: 12,000

Some Vital Statistics

We have talked rather vaguely about fuel consumption, cycle consumption, and damage. In general, of course, you want your ChipWit to stay away from the fuel draining electro-crabs and the damaging walls and bouncers. Here are specific numbers to indicate the exact nature of these dangers.

Fuel

A ChipWit begins each mission with 7,000 units of fuel. When it eats pie or coffee it gains 1,200 units of fuel. When it is attacked by an electro-crab, it loses 200 units of fuel. The amount of fuel consumed by each Operator is the third column in the chart below.

Cycles

ChipWits begin each mission with a set number of cycles. This number is determined by the Environment. Cycles are expended for each Operator executed. The amount expended for each Operator is the second column in the chart below.

Operator	Cycles	Fuel
Loop	1	1
Sub-Panel	1	1
Boomerang	1	1
Junction	0	0

Move	3	5
Pick up	1	5
Zap	1	7
Sing	2	2
Feel	4	4
Look	4	2
Smell	4	2
Coin Flip	3	1
Keypress	3	1
Compare Number: Equal?	2	1
Compare Number: Less Than?	2	1
Compare Thing: Equal?	2	1
Compare Move: Equal?	2	1
Save Number	1	1
Save Thing	1	1
Save Move	1	1
Pop	1	1
Increment	1	1
Decrement	1	1

Damage

A ChipWit begins each mission in perfect health. Damage to a ChipWit is not repairable. 1,400 Units of damage kills a ChipWit. Attempting to Pick up a wall inflicts 50 units of damage. Attempting to Pick Up an electro-crab or bouncer inflicts 150 units of damage. Bumping into an electro-crab or a bouncer inflicts 100 units of damage. Bumping into all other objects inflicts 50 units of damage. Detonating a bomb inflicts 1,400 units of damage.

OBJECTIVES

In Chapter I you learned how to operate your ChipWits system. You also learned the elements of programming with the Workshop. After completing this section you will be able to:

1. Use the Keypress for testing program routines within an environment.
2. Use Stacks to improve the efficiency of your ChipWit programs.
3. Increase the speed and efficiency of your ChipWit and your programming.

THE KEYPRESS

The Keypress Operator is represented by an icon of an Apple keyboard.

The Arguments for Keypress are the letters of the alphabet and a blank key.

When a chip with a Keypress executes, it instructs the program to check if a specified key (the letter you have chosen for the Argument) on the Apple keyboard is being pressed. If the key is being pressed at the time the chip executes, the program branches out the Keypress chip's true wire. If the wire is not being pressed, the program branches out the chip's false wire. The blank key Argument will cause the chip to branch through its true wire if no key is pressed. If any letter is pressed, the chip branches through its false wire.

By selecting Keypress and a letter from the Arguments Menu, you install a manually controlled command in your ChipWit. The Keypress Operator gives your ChipWit the benefit of your human intelligence.

Uses

In Chapter I, we broke ChipWit programming into three parts: Instruction Sets, Sequence of Instruction, and Conditional Testing. To help you master robot building, the Keypress can be used to separate the challenges of constructing Instruction Sets from those of determining the larger Sequences of Instruction.

For example, one of the challenges a ChipWit faces in any Environment is knowing when it is time to leave a room and then finding a door and exiting. You can substitute the routine in which the ChipWit determines if it should leave the room with a Keypress linked to a routine to find a door and exit. As you are watching your ChipWit on a mission, make the decision yourself about when the robot should leave the room. Activate the Keypress and send your ChipWit to the exit routine.

This allows you to perfect the exit routine. Once this is accomplished, go back and create a routine for determining when it is time to leave a room. Put this in the program replacing the Keypress.

After all routines to perform specific functions have been worked out (clearing a room of dangerous Things, clearing a room of good Things, finding doors, etc.), the Keypresses used to call them can be substituted by new routines that determine when they should be called. By removing the Keypress commands and substituting sensing routines, you will remove your guiding hand and let the ChipWit think for itself!

Each routine that you control with the Keypress can easily be constructed in a Sub-Panel. Place you Keypress chips on the Main Panel and connect the true wires to Sub-Panel chips.

When you are ready to replace the Keypress with an internally driven routine, put that routine on a Sub-Panel and replace the Keypress chip with a call to that Sub-Panel.

You now can practice making the ChipWit perform tasks, using manual controls in case your ChipWit is not sure when to perform them.

A Special Application

Appendix A in the back of this manual has a few ChipWits for study (and improvement). ChipWit #1 is called KeyBaby. It is a completely Keypress controlled ChipWit. By using the I-J-K-M diamond on the keyboard, you can control KeyBaby's movement. Pressing Z causes a Zap and P causes a Pick Up.

KeyBaby works effectively because the program is so small that you are never far from the Keypress you need. In larger programs with Sub-Panels, repeating Keypresses and the routines they call may be necessary.

KeyBaby was designed for two reasons. He is a great tool for the savvy ChipWit builder to explore ChipWit Environments. Such explorations yield information about the Environments that is helpful in building true, internally driven ChipWits. KeyBaby is also great for acquainting younger people with the ChipWits system.

MEMORY STACKS

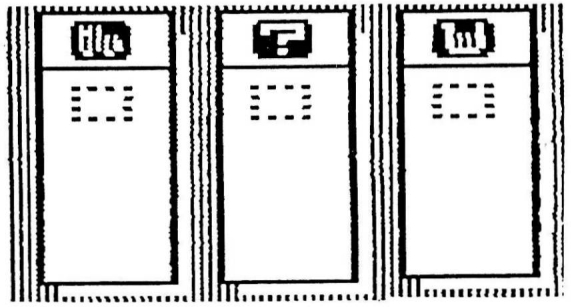
Memory Stacks were discussed briefly in Chapter I in the section on Operators and their Arguments. Stacks are storage areas in which ChipWits can keep lists of:

1. Moves the ChipWit makes.
2. Things your ChipWit encounters.
3. Numbers (from a variety of sources).

Items are stored in the Memory Stacks on a first in/last out basis. This means that the last item (move, number, or Thing) that you asked the ChipWit to remember will be at the

top of the stack. When you ask the ChipWit to save another item, that item becomes the top item on the stack and the former top item moves to the second slot.

The Memory Registers shows the top three items in each of the stacks. The item at the top is first in the stack, the item at the bottom is third. The registers for the stacks are illustrated below.



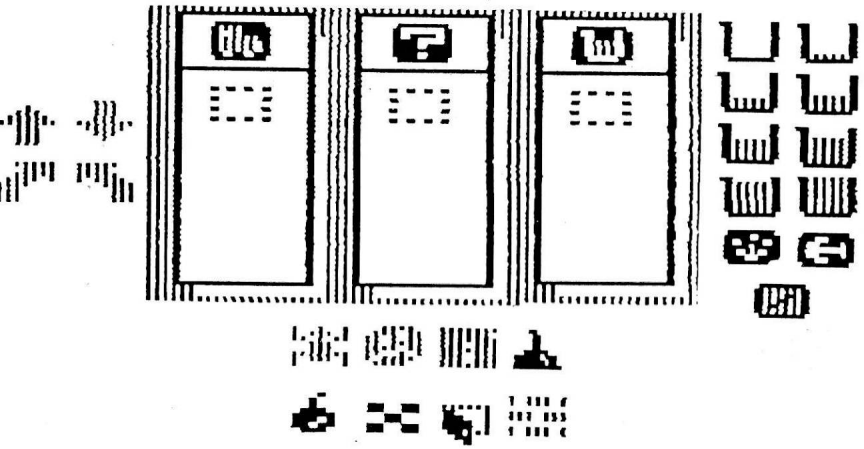
The ChipWit has access only to the top item of the stack. When you use the Operator COMPARE MOVE, for example, you are commanding the ChipWit to compare the Argument of the COMPARE MOVE chip with the move at the top of the Move Stack. Likewise, if you use the COMPARE NUMBER: EQUAL? Operator, you are commanding the ChipWit to compare the Argument of that chip with the number at the top of the Number Stack.

For the ChipWit to use items other than the top one in a stack, it must use the POP Operator to remove the top item from the stack. (The Argument that goes with the POP Operator tells the robot which stack to remove the top item from.) After POPping the stack, the item that was in the second slot becomes the first item. All other items move up one space as well.

Stacks can keep lists up to 200 items long. The 201st item pushes out the item that was put in the stack first.

Saving to Stacks

By using the three SAVE to stack Operators, you command the ChipWit to add items to the three stacks. Below is a chart which shows the NUMBER, MOVE, and THING stack icons and the Arguments that go with each. The Arguments represent the items that can be saved to that stack.



Stack Functions

What can the ChipWit (and you, the ChipWit programmer) do with the information that has been stored in the stacks? Where does the ChipWit get the information that is saved into the stacks? How do you use the information to build a better ChipWit once it is there? Read on!

The Thing Stack

The ChipWit can compare the top item in the Thing Stack to any one of the ten ChipWit Things (pie, coffee, oil, disks, electro-

crabs, bouncers, bombs, walls, doors, and floors) as well as to an empty square. (The empty square represents the bottom of the stack. If you choose COMPARE THING and select the empty square Argument you are asking the ChipWit to see if the stack is empty.) If the Argument of the COMPARE THING chip and the Thing at the top of the Thing Stack are the same, the program branches from the true wire. If they are different, the program branches from the false.

The Move Stack

The Move Stack functions in exactly the same way as the Thing Stack. If the top move in the Move Stack is the same as the Argument of the COMPARE MOVE chip, then the program branches from the chip's true wire. If they are different, it branches from the false wire. Again, the empty square Argument for COMPARE MOVE represents the bottom of the Move stack. If the stack is empty, the bottom of the stack will match the empty square.

The Number Stack

The Number Stack has two compare Operators: COMPARE NUMBER: EQUAL? and COMPARE NUMBER: LESS THAN? The former (COMPARE NUMBER: EQUAL?) works the same way as the compare Operators for the Move and the Thing Stack. If the match between the Argument and the value at the top of the Number Stack is identical, the branching is true. If the number in the top of the Number stack is not the same, branching is false.

COMPARE NUMBER: LESS THAN? also compares the top number in the Number Stack to the chip's Argument. If the Number in the Stack is less than the Argument, the branch is true. If it is more than or equal to the Argument, the branch is false.

Both Operators utilize the empty square Argument to check if the stack is empty.

The Stacks in Use

We now know what the stacks are capable of doing. Below are hints for each stack that will show you two things:

1. Where you can instruct your ChipWit to get items for its stacks.
2. Some uses for stacks.

The Thing Stack

There are many methods you can use to construct lists in the Thing Stack. You can have the ChipWit push into the stack anything it smells, sees, eats, zaps, or feels. Using the SMELL Operator and the Thing Stack, you can create an inventory of all Things in a room. With such an inventory, you can insure that the ChipWit does not leave any room until it has acquired all the high point items (usually disks).

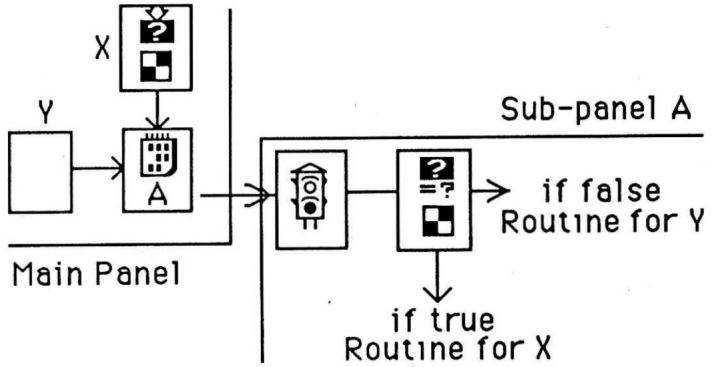
Using the Environments description at the end of the first chapter as a guide, have the ChipWit push all possible Things for that Environment into the stack. Program this routine on the Main Panel. Follow it with a routine telling the ChipWit to SMELL for the item at the top of the Thing Stack. If the match is positive (the item at the top of the Thing stack is SMELLed in the room), jump to a search and acquire Sub-Panel. If the Conditional Test is negative, Pop the item and SMELL for the new top item.

A variation on the method above can be used with the LOOK Operator. If the LOOK Operator on the Main Panel detects a Thing, push that Thing into the Thing Stack and jump to an

acquire Sub-Panel. There, LOOK for the Thing at the top of the Thing Stack. Move in a straight line until acquisition is possible. If the Thing is one that you want to ZAP, no ChipWit motion is required at all.

All stacks can be used as codes. This is a fairly sophisticated but effective way of using them. Select an Argument that is not present in the Environment or a Thing-- such as FLOOR-- that you will not be storing in the Thing Stack. This Argument will have a special meaning when found at the top of the Thing Stack.

A Sub-Panel chip on the Main Panel that is accessed by two separate chips can use the coded Thing to determine which chip led to the Sub-Panel. Once you have jumped to the Sub-Panel, check to see if that Argument is at the top of the stack. Based on the presence or absence of the coded Thing, you can sub-divide the Sub-Panel. The true wire of the COMPARE THING chip can lead to a routine that addresses one potential situation. The false wire on the COMPARE THING chip triggers a routine completely separate in purpose. The illustration below clarifies.



The Move Stack

The Move Stack is especially helpful in bringing your ChipWit back from long trips away from the center of an Environment or in Environments where travel back and forth in a pattern is required. (This is particularly true in the case of several Environments. Can you tell which ones?) As a ChipWit makes moves, push the opposite move into the Move Stack. When the time comes to retrace steps, simply select the Move Stack Argument for the Move Operator. Remember to Pop off each move as you use it!

ChipWit 4 in Appendix A uses the Move Stack this way. Octobuster, although fairly complex, is worth study to see how this is done.

The Move Stack, like the Thing Stack, can be used as a code to pass information to Sub-Panels. This will allow you to use the Sub-Panels for more than one application.

The Number Stack

The Number Stack is perhaps the most useful of the three memory stacks. It can be a great aid in conserving fuel units and cycles, as well as monitoring the health and well being of a ChipWit.

If the ChipWit detects a Thing through a LOOK Operator, the number of floor tiles between the robot and the Argument are registered in the Range Finder. Use the SAVE NUMBER Operator and the RANGE FINDER Argument to store the distance in the Number Stack. Now have the ChipWit COMPARE NUMBER: EQUAL? with the Argument one. True? Grab the Thing! False, move forward one tile, decrement the Number Stack by one, and loop back to the COMPARE NUMBER chip. Using the Number Stack in this situation-- as opposed to merely reaching and moving until the Thing was acquired-- allows you to program your ChipWit to operate

efficiently. (Can you figure out another way to use the Number Stack and the Range Finder to move toward seen objects?)

The Number Stack can be used to keep watch on the ChipWits health. The Damage Meter and the Fuel Meter produce values that can be pushed into the Number Stack. To build a ChipWit that eats only when it is hungry, monitor hunger by comparing the value from the Fuel Meter with a criterion you set (based, perhaps, on the availability of food in the specific Environment). When the meter reached a level you determine, send your ChipWit off to an acquire food Sub-Panel. In the mean time, let the robot concentrate on point producing acquisitions.

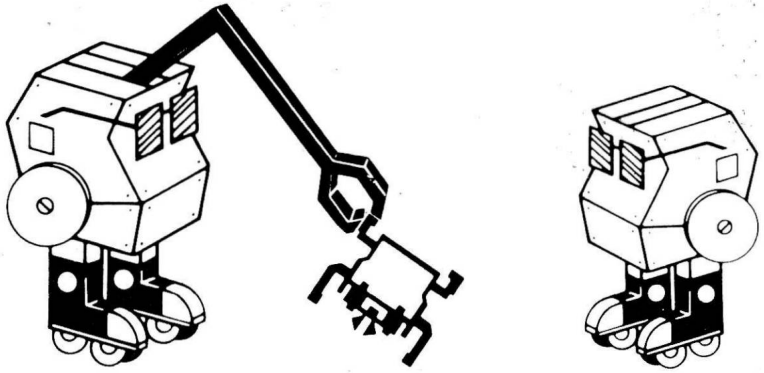
In the same way as you use the Number Stack to monitor ChipWit hunger, you can use it to monitor ChipWit damage. The Damage Meter, like the Fuel Meter, produces a value. This value can be pushed into the Number Stack and compared with criteria you set. If the ChipWit reaches the point where it must be careful not to sustain further damage, send it off to a Sub-Panel where a less aggressive movement and acquire routine will conserve its health.

FINAL WORDS

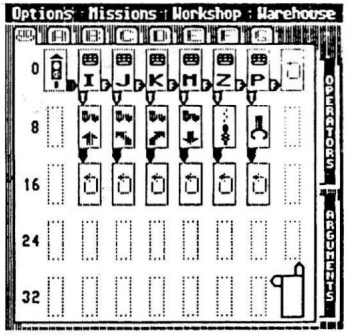
In this documentation, we have tried to supply you with the information you need to enjoy ChipWits. The ultimate value of the program will be determined by the learning that occurs after you have read and understood the documentation. We hope we have teased your interest enough for you to go on and discover many clever ways to use stacks, Sub-Panels, Keypresses and all the rest.

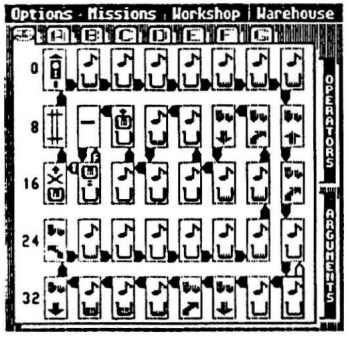
To help you along, we have included a Rogues' Gallery (Appendix A). This contains four ChipWits. Each needs

improvement. General suggestions are offered, but the analysis and implementation of the changes needed is up to you. After you have cleaned up the scoundrels, you'll be more than ready to build your own robots from scratch. Maybe you already are!



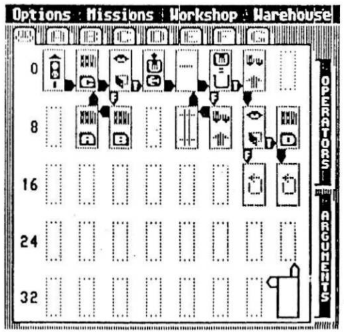
Keybaby: Keybaby, the darling of ChipWit builder Eric Madsen, is a great tool to use exploring Environments. It also allows very young children to play with ChipWits. Since the ultimate goal of all Keypresses is to be replaced by internally driven ChipWit Instruction Sets, see what you can train Keybaby to do for himself.

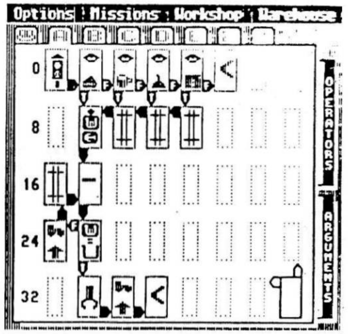


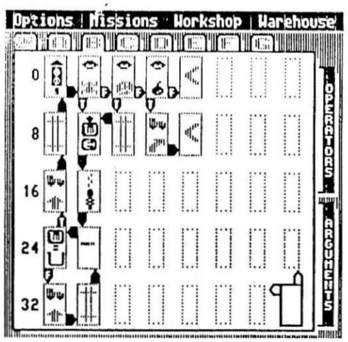


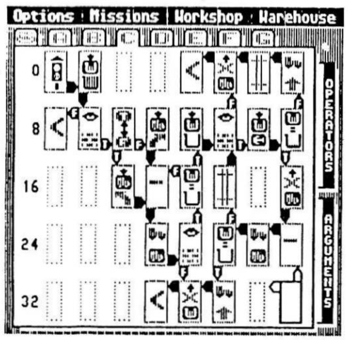
Carusoid. Carusoid, built by Leahcim Notsnhoj, is another ChipWit that will get young people interested in robot programming. Try having a beginner improving the singer bit by bit. Can you make him sing only in certain occasions (like before his dinner)? Remember, you can cut the song panel and place it on a Sub-Panel. There it can be used on special occasions. Perhaps you could teach him another song. Maybe just keeping him from destroying himself would be a good place to start. Basically, Carusoid is just for fun.

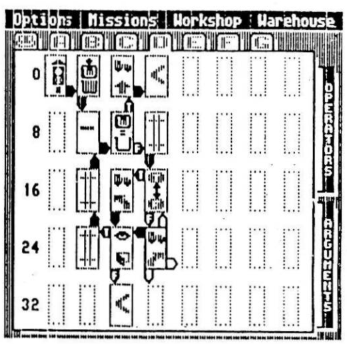
Kaye. Kaye, a darn good bread (point) winner, was built by Richard Phaneuf. Kaye is an aggressive ChipWit who runs in where she has conquered. Perhaps she is a bit reckless. Sometimes she gets herself tied up in circles. Watch her perform. Study the use of the Number Stack. Pretty good, eh?



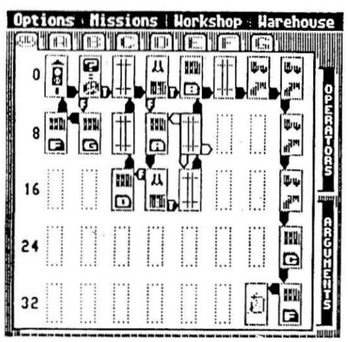




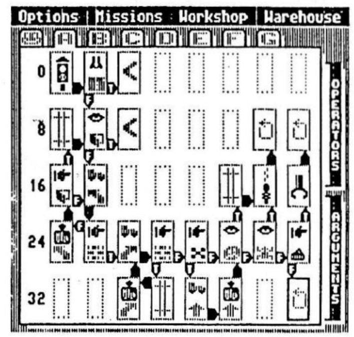


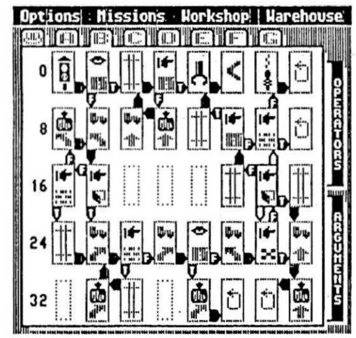


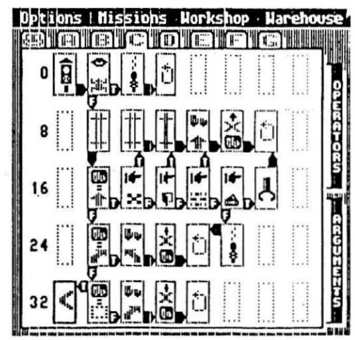
Octobuster's Main Panel

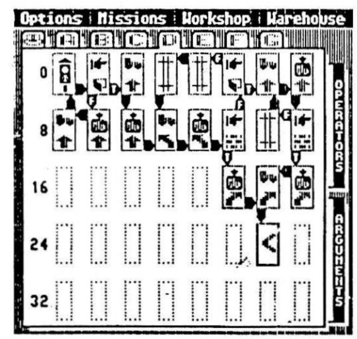


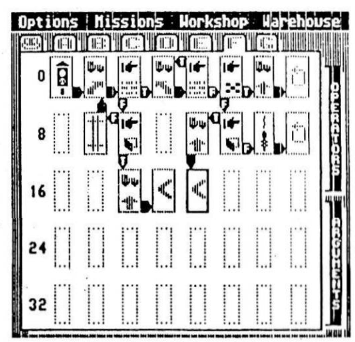
Octobuster. Octobuster was awarded the coveted grand prize in the Prix de Minneapolis in 1984. The humble author, none other than D. Sharp, Esquire, modestly points out that the use of the Move Stack in Octobuster is "...well, the finest example of ChipWit programming that you're every likely to see, frankly..., I mean really." Hmm. Do you thing Octobuster looks at his Fuel Meter often enough?

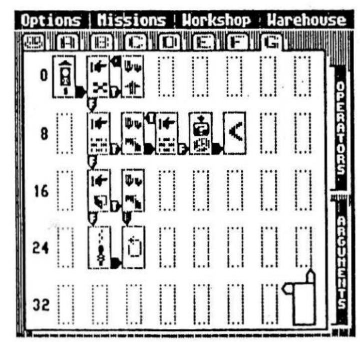












Argument - The object of an Operator.

Boomerang - The Operator that instructs the program to return from a Sub-panel to the Main panel.

Chip - The container of an individual Instruction Set (Operator and Argument). Chips are set in panels and connected to each other by true and false wires.

ChipWits - The robots you program.

Coin Flip - An Operator that performs a random choice between a True or a False branch.

Compare (Thing, Move, or Number) - An Operator which checks a Stack to determine if a condition (Things, Moves, Numbers) is present at the top of the stack.

Conditional Testing - Checking to see if a specified thing is present in an Environment or a stack. Depending on the presence of the thing, the ChipWit program branches through the testing chip's true or false wire.

Debug - An on-screen display providing information allowing the user to trace a ChipWit program's execution for the purpose of determining how well it is operating.

Decrement - An Operator that decreases the top value in the Number stack by one.

Environments - The rooms in which ChipWits carry out missions.

Flow Charts - A graphic format used by programmers to plan their programs.

Flow of Control - The Sequence of Instructions with in a program. The order in which the instructions are given to the robot.

IBOL (Icon Based Operating Language) - The picture defined set of commands which tells the ChipWit what to do.

Icon - A stylized picture which represents an instruction, action, Thing, or memory stack.

Increment - An instruction to increase a value in the number Stack by one.

Instruction Set - The combination of an Operator and an Argument which gives a command to a ChipWit. In some cases, an Operator alone (such as PICK UP) can make a complete Instruction Set.

Junction - A chip that has no other function than to connect two other chips.

KeyPress - Operator which checks whether a specific key has been pressed and branches True or False based on this.

Loop - The return-to-beginning instruction in a Panel.

Main Panel - The panel on which the primary Flow of Control is programmed. The Main panel calls the Sub-panels to execute.

Menu Bar - The strip across the top of the screen displaying the pull-down menus available to the user.

Mission Assignments - The objectives to be achieved within the different Environment by the ChipWit. Assignment vary based on the challenges of the Environment.

Operator - The action command telling the ChipWit what it is to do.

Options - A utility menu found on the Menu Bar.

Paste - An editing command allowing the user to transfer a previously Cut or Copied Panel to a new location within a ChipWit or to a new ChipWit.

Pop - Removes an the top item from a Stack.

Registers - On screen displays that show the top three items in each of the three stacks.

Save (Move, Number, or Thing) - An Operator which places (saves or pushes) a move, Thing, or number into a stack.

Stacks - Three different memory storage areas each holding up to 256 Things, numbers, or moves.

Stats Panel - A display showing the number of missions run by the current ChipWit in the current Environment, the average score for those missions, and the highest score from those missions.

Sub-Panel - A panel which is called from the Main panel. Sub-panels are identified by the letters a through G and can be repeatedly used in the program.

Things - Objects that a ChipWit encounters in an Environment (oil, disks, pie, coffee, electro-crabs, bouncers, bombs, walls, doors, and floor tiles).

Warehouse - The place where ChipWits are stored. Also, the menu which allows you to load ChipWits into an Environment or the Workshop.

Wires - (True and False) Connected to each chip. Direct Flow of Control based upon the results of Conditional Testing.

Workshop - The place where you build your ChipWits. Also, a menu that allows you to begin building or editing ChipWits.

Zap - The command for destroying all Things (except doors, walls, and the floor).

WARRANTY

'ower, Inc. warrants all of its products to be free of operating defects, whether caused by media or program error. Should you find an error which you believe to be caused by a failure of the or the program, you should return the disk for replacement along with a written description of oblem to **BrainPower, Inc., 24009 Ventura Blvd., Suite. 250, Calabasas, CA 91302**. Media failures ng less than ninety (90) days after purchase will be replaced at no charge. Those occuring after riod, or submitted without proof of purchase will have a service charge of **$10.00** plus shipping andling. Programming defects that are verified by **BrainPower** will be corrected and the m returned to you at no charge. This lifetime programming warranty shall remain in effect, so this product is being sold. Should **BrainPower** be unable to correct the error after a reasonable of time, you will be entitled to a full refund of the original purchase price.

event will BrainPower, Inc., its employees, or its authorized representatives be liable for quential or incidental damages arising from the purchase and/or use of this product, even if ower, or its authorized representatives have been advised of the possibility of such damages. states do not allow the exclusion of liability for incidental or consequential damages, so ove limitations and exclusions may not apply to you.

event you wish to consult with **BrainPower** regarding any question you may have, please call us 384-6911, and ask for **Customer Support.**

NOTES