UNIDEX 16

PROGRAMMING

CLASS

CHAPTER 3: THE EDIT MODE

Unidex 16 has a powerful editor which allows you to edit your programs with ease.

The edit mode offers two ways of editing a program - Screen Editing and Command Editing.

1. SCREEN EDITING

Screen editing allows you to place the cursor anywhere in the file (general display area) and change, add or delete single characters, thereby directly altering the text. This occurs when the [new-line] or [revise] mode is activated.

2. COMMAND EDITING

Command editing allows you to change many blocks of program via the soft key commands.

NOTE: The cursor (_) will be in the text (general display area) of the CRT when the edit mode's [new-line] or [revise] function is activated. Otherwise the cursor will be in the Command Entry Area only.

The block pointer (*) will be in the text (general display area) of the CRT, highlighting the current program block.

During [revise] or [new-line], both the cursor and the block pointer are in the same block of text. If, during [revise], when other edit functions (such as [find] or [copy]) are activated, the cursor will jump back to the line that the block pointer occupies as soon as the soft key edit function is executed.

When the [edit] soft key is selected, the following soft key choices will appear:

```
        <FILE>   new   last-edit   last-run   digitize
```

A. <FILE>

Pressing any softkey enclosed by < > will not cause any action to occur. It is only there to indicate that some action is required on your part. For example, <FILE> indicates that a file name may now be entered. Just type in the file name and then press soft key [.PP], [.CM] or [.TO]. (The [.PP], [.CM] and [.TO] commands indicate what sort of program (parts program, command file or tool file) is about to be edited. This will be described in section 3-3, Program Selection.)

If you are about to enter a new program, press the soft key [new] before entering the file name. You will see:

```
        <FILE>                                  digitize
```

As you enter the file name you will see:

```
          .PP       .CM       .TO
```

Select one and press <ENTER>. You will see:

```
      new-line   revise   find   delete   end   --ETC--
```

[New-line] will be highlighted if you are entering a new file.

B. LAST-EDIT

If the file you wish to revise is one that is the most recently edited file, the soft key selection [last-edit] and <ENTER> allows you to bring it up on the screen immediately for further editing.

C. LAST-RUN

If the file you wish to revise is one that is the most recently executed file, the soft key selection [last-run] and <ENTER> allows you to bring it up on the screen immediately for editing purposes.

D. DIGITIZE

Digitizing enables you to move the axes from point to point using the joystick, and enters the values into a parts program.

If you have the joystick option, press the [digitize] soft key to see:

```
                 inc.mode                        abs.mode
```

Select [inc.mode] to enter data in the incremental mode, where each new point is referenced to the last point, or select [abs.mode] to enter data in the absolute mode, where each new point is referenced to the home position.

Once [inc.mode] or [abs.mode] is selected, you will see:

```
 <FILE.PP>
```

This indicates that you may now enter the name of the file which will contain your digitizing information.

For more information on the joystick option, refer to chapter 13, Unidex 16 Options.

The next section (section 3-1), will explain entering a new file. Section 3-2 will discuss revising an existing file.

B. REVISE

In order to revise a statement, press [revise]. [Revise] is now highlighted, and you may move the cursor through the program via the arrow keys, making any necessary changes. Press [new-line] to quit the [revise] function and continue entering the new program. Whenever [new line] or [revise] is in effect, pressing it again will cause you to exit that function, since both [new-line] and [revise] toggle.

In the sample program, after pressing [revise], move the cursor up to the first block. Maneuver the cursor by using the arrow keys. Overwrite AAAAA with ZZZZZ.

C. FIND

The soft key [find] allows you to search for a single character, or a whole string.

NOTE: **Be Sure The Entry Is Enclosed In Quotation Marks Or You Will Receive An Error Message.**

Once you have typed in your entry, the soft keys will ask you:

```
                    thru     until     all
```

If you press [thru], Unidex 16 will find all entries up to and including a specific place in the program.

If [until] is selected, the search will continue up to but not including a specific place in the program.

[All], of course, causes Unidex 16 to search throughout the entire program for the selected entry.

Pressing either the [thru] or [until] selection will display:

```
<#-BLOCK>   start   end   <STRING>
```

At this point you may command Unidex 16 to search through or search until a certain block, string of characters, program start or program end. (If you choose <STRING> or <#-BLOCK>, enclose the string's characters or the block number in quotation marks. If you are entering a *number* of blocks to search, upward or downward, however, the number does not require quotation marks.)

If you've selected <#-BLOCK> or <STRING>, you are offered the choice of [upward] or [downward]. The direction you choose is in reference to the block which contains the block pointer.

In the case of multiple occurrences of the entry or string, the one closest to the block pointer is the one that will be found. To continue a search, just press <ENTER> if you are not in the [revise] mode. Otherwise, reenter the string, or more simply, press <RECALL LINE> in order to recall your [find] command without reentering it. Now just press <ENTER> in order to find the next occurrence of your entry.

The [all] selection is self-explanatory. Unidex 16 searches the entire program for your requested entry. If the entry or string occurs more than once within the program and you've selected [all], the first line on which the entry occurs within the program is the one Unidex 16 will find.

Using the sample program, press [find] and then type in "E". When asked, press [thru], then [end] and <ENTER>. The block pointer will jump to the last block.

D. DELETE

When the soft key [delete] is pressed, the following choices appear:

```
                    thru     until     all
```

Pressing [thru] or [until] will display:

```
          <#-BLOCK>   start   end   <STRING>
```

The program will be deleted from the block pointer to a given point. The point you select may be until or thru the start or end of the program, or until or thru a particular block or string. Again, if you choose <#-BLOCK> or <STRING>, you will next need to enter [upward] or [downward], after entering the character or characters in the delineating string or block. (These characters must be enclosed in quotation marks.)

The command [all] will delete the entire program upon pressing <ENTER>.

If [delete] is pressed, and then <ENTER>, instead of [thru], [until] or [all], the line on which the block pointer is located will be deleted.

Using the sample program, press [delete]. When asked, press [until], enter "C", press [upward] and <ENTER>. The last two blocks will be deleted. At this point, your program should consist of:

```
* ZZZZZ
* BBBBB
* CCCCC
```

and the status line will say "deleted".

E. CHANGE

You can replace one character or string with another by using the edit function [change]. When in the edit mode, press [--ETC--]. You will see:

```
            change  merge  move  retrieve  copy  --ETC--
```

Press soft key F1 for [change]. You will see displayed:

```
         <STRING>                                        to
```

Type in the entry or string to be changed, remembering to place quotation marks around it. As you enter the characters, <STRING> will disappear and only [to] will remain. Pressing [to], you will see:

```
  <STRING>       thru       until      all
```

Type in the replacement string and press [thru], [until] or [all].

As in previous examples, [thru] or [until] will give you:

```
            <#-BLOCK>    start    end     <STRING>
```

Depending on which of the above you select, your changes will occur [thru] or [until] the: start of the program, the end of the program, a given block or a given string. [All], of course, executes changes throughout the entire program.

<#-BLOCK> or <STRING> entries will give you a choice of [upward] or [downward].

If, after requesting a change, you simply press <ENTER>, the change will occur only on the block pointer line.

Once a change is made, the CRT displays the first screen of edit functions. To get the [change] soft key again, press [--ETC--]. In fact, pressing [--ETC--] will keep looping the three screens of edit functions around again and again.

NOTE: If you mistakenly ask for a soft key command, just step backward by use of the arrow key (<--), or press <SHIFT> and <CLEAR LINE> simultaneously.

Referring again to the sample program, press [--ETC--] to view the second screen of soft keys. Press [change] and you will see:

```
        <STRING>                                        to
```

Enter "Z" and press [to]. The display will show:

```
        <STRING>    thru    until    all
```

Type in "A", press [all] and <ENTER>. Your program should now consist of:

```
* AAAAA
* BBBBB
* CCCCC
```

and the status line will tell you the total number of changes that took place.

F. MERGE

[Merge] joins an existing file with the file being edited.

In order to merge one program file with another, just press soft key [merge]. You will see:

```
 <FILE>              char-sub        from-buff
```

The [char-sub] softkey will call for the Character Scribing Package, if your system contains this option. It is explained in detail in chapter 13, Unidex 16 Options. The [from-buff] soft key is explained on the next page.

To merge an entire file, simply type in the name of the file to be incorporated into the present file.

If you want the new file to be placed between lines 3 and 4, for example, place the cursor on line 3, press [merge], type in the correct file name and press <ENTER>.

Your soft key selection will then revert to the first page of edit functions again.

Using the sample program of this chapter *(SECOND)*, and the sample program of the previous chapter *(FIRST)*, we will merge two files together.

Press [revise] to move the cursor to the last line of the second program *(SECOND)*, if it's not already there. This should be the block which reads CCCCC.

Press [--ETC--] to view the second screen of soft keys. Press [merge] and type in the file name *FIRST*. Press [.PP] and

<ENTER>. The two programs will be merged. (The program will merge after the line that contains the cursor.)

In the above example, you merged two *complete* files. For an example of merging a *partial* file, do the following. First [end] and [update] this file. Now go into the example file called "FIRST". Go to the second line, and press [--ETC--] twice, and you will see:

```
        renumber   repeat   mrg-start   mrg-term   --ETC--
```

Press [mrg-start] and <ENTER> for the beginning of the portion of the file to be merged. Now go to the fifth line and press [--ETC--] and then [mrg-term] and <ENTER> to terminate the portion of the file to be merged. When you select [mrg-term], the portion to be merged ends on the line *above* the cursor. (When we merge this into another file, lines 2, 3 and 4 will be merged.)

Now [end] and [update] this file and edit the example file *SECOND* again, by pressing [edit], entering the filename, pressing [.PP] and <ENTER>.

Go to the second line of the program. Press [--ETC--] and then [merge]. Select [from-buff]. Press <ENTER>, and the lines delineated in the file called *FIRST* by the [mrg-start] and [mrg-term] commands, will be merged into the file called *SECOND*.

G. MOVE

[Move] relocates block(s) of data within a program. Press [move]. The display of soft keys will show:

```
                    thru        until        all
```

Moving a section of program consists of:

1. Placing data in buffer (extracting).

2. Placing data in another location (retrieving).

[All] will place the whole program in the buffer. You can get the information back by pressing [retrieve].

[Thru] and [until], as in previous examples, will yield:

```
        <#-BLOCK>   start   end   <STRING>
```

Again, depending on which of the above you select, the data to be moved will be [thru] or [until] the: start of the program, the end of the program, a given block or a given string. If <#- BLOCK> or <STRING> is chosen, [upward] and [downward] will be displayed. Choose according to the location of your block pointer.

If you only press [move] and <ENTER>, the line on which the block pointer is located will be the only one placed in the buffer.

Data that has been extracted can be recalled by pressing [retrieve].

NOTE: If further information is moved, the prior data in the buffer will be overwritten, and therefore lost.

H. RETRIEVE

Soft key function [retrieve] enables you to call back information that has been either moved or copied into the buffer, and deposit it anywhere in the program you like.

When you want to retrieve material and place it between lines 5 and 6, for example, place the cursor at line 5 and press [retrieve]. The information will be placed between lines 5 and 6.

Using the sample program *SECOND*, [find] "B". That will take the block pointer (and cursor if [revise] is activated) to the second block of the program. Now press [move] and then [thru]. You will see:

```
          <#-BLOCK>   start   end     <STRING>
```

Press [start] and <ENTER>. Both lines 1 and 2 of the program will be removed and placed into the buffer. Skip down to the last line of the program and press [--ETC--], [retrieve] and <ENTER>. Lines 1 and 2 will now be the last two lines of the program.

I. COPY

[Copy] is like [move] in that it places data to be copied into the buffer. However, the program lines are left in the existing position as well. It is duplicated in the buffer and will be copied elsewhere in the program when [retrieve] is pressed.

These soft key functions are similar to the [move] selections. Press [copy] and the soft keys display:

```
                    thru      until      all
```

[All] will copy the entire program. [Thru] and [until], as before, will display:

```
          <#-BLOCK>    start    end     <STRING>
```

Depending on which of the above you select, the data to be copied will be [thru] or [until] the: start of the program, the end of the program, a given block or a given string. If you choose

<#-BLOCK> or <STRING>, the choice of [upward] or [downward] will be given.

If you press [copy] and then <ENTER>, just the line on which the block pointer is located will be copied.

As in [move], if you copy more information into the buffer, the prior information will be overwritten.

By pressing soft key [--ETC--], you will see these editing choices:

```
                renumber      repeat
```

J. RENUMBER

You may number or renumber a program with the soft key [renumber], located on the third screen of edit functions.

When you press [renumber], the next group of soft keys will display:

```
                from#      inc#      no#
```

These will give you the following choices:

from# -allows you to select a number for the first block.

inc# - allows you to select a number by which the following numbers will be incremented.

no# - allows you to eliminate the line numbers.

For example, if you wish to number your sample program, press [-- ETC--] and [renumber]. The screen will display:

```
            from#      inc#      no#
```

You may just press <ENTER>, in which case you will have the default values for line numbering, where the first line number is 1, and the following line numbers increment by 1. Or, you can press [from#] and enter a number, such as 10, for example. Then press [inc#] and enter 5, for example. Press <ENTER>. Your program would display:

```
N10  (First block of program)
N15  (Second block of program)
N20  (Third block of program)
```

If you want to remove the line numbering, press [renumber], [no#] and <ENTER>. The line numbering will disappear.

If you want to change the line numbering, just press [renumber], press [from#] and enter a new first number. Then press [inc#] and enter a new increment.

K. REPEAT

The edit function soft key [repeat] allows you to repeat a block of information over and over again. Simply move the block pointer to the desired line, press [repeat] and <ENTER>. Each time <ENTER> is pressed, that block will be repeated.

Go to any line of your sample program and try the repeat function.

L. END

When you want to exit the edit mode, press [end]. The soft keys will then display the choices:

```
              abort          up-&-run    update
```

In order to store the file you have just entered, press [update]. Unidex 16 will do internal file managing to put it into the proper location in memory.

To eliminate the file you have just created, press [abort]. In this case, nothing will change in the internal memory.

In the case of a revision, [update] will store the latest version of your file, deleting the previous one. [Abort] will keep the previous file and delete the revision just made.

If you choose [up-&-run], Unidex 16 will update the file and go directly into the machine mode. Program execution will start from where the Block Pointer is located in the program when [end] and [update] was entered. The program will be scanned to set up variables, but *no lines that come before the block-pointer will be executed.* Therefore, you should be aware that variables will have a value of zero when the program begins execution at the desired block. You would probably want to make sure, therefore, that the block pointer is located *before* the program blocks that set the variables to specific values. (Program execution will be in the "single" mode.)

SECTION 3-2 REVISING AN EXISTING FILE

If you are revising an old file, press [edit]. Then enter the file name when you see:

```
<FILE>    new    last-edit    last-run    digitize
```

Then enter [.PP], [.CM] or [.TO] and press <ENTER>. Your file will appear for revisions. However, [new-line] will not be highlighted.

The edit functions for revising an existing file are:

- new line
- revise
- find
- delete
- change
- merge
- move
- retrieve
- copy
- renumber
- repeat

Note that the edit functions are the same for both entering a new file and revising an old one.

What is seen on the screen is a copy of the original file. If, after the revisions, you press [end] [update], *the original file is deleted and the revised copy replaces it*. On the other hand, if after the revisions, you press **[end] [abort]**, *the original file is kept and the revised copy is deleted.*

If you choose [up-&-run], Unidex 16 will update the file and go directly into the machine mode. The program will be scanned to set up variables, but no lines that come before the block-pointer will be executed. Therefore, you should be aware that variables will have a value of zero when the program begins execution at the desired block. You would probably want to make sure, therefore, that the block pointer is located *before* the program blocks that set the variables to specific values. (The program will be run in the single mode.)

NOTE: Since Unidex 16 is a multitask system, you may edit one program while another one is running. As a safety precaution, however, you cannot edit the program that is currently running. If you try to do so, Unidex 16 will complete the block it is currently executing and stop running the program. The file is now inactivated. To inactivate a file, you may also format the internal memory. For details, see chapter 4, section 4-1 F.

SECTION 3-3 PROGRAM SELECTION

At the beginning of this section, we mentioned that once a program name is called for, either old or new, the program selection of either [.PP], [.CM] or [.TO] must be entered (via the soft keys).

A. PARTS PROGRAM

The parts program, or [.PP], is your actual program, made up of motion commands (refer to chapter 10). An example of a parts program we will call *SAMPLE* is:

SAMPLE .PP

```
(REF,X,Y,Z)
G0 X75. Y-75. Z-136.
G4 F.5
```

```
G1 Z-10. F500.
T0102
G4 F5
G41
X16. Y16. F500.
Y30.
X45.
Y-30.
X-45.
X-16. Y16.
G2 L9. D180 C0 F500.
G2 L9. D0 C180 F500.
G40
G1 X16. Y-16. F500.
G1 X-16. Y-16. F500.
G1 Z10.
M30
```

B. TOOL FILE

The tool file contains tool diameters and offsets. The parts program specifies the tool number and the line in the tool file from which to get information on tool diameter and offsets. For example T0102 would call for Tool #1 using the offset information on line #2 of the tool file. Following is an example of a tool file we will call *SAMPLE*.

SAMPLE .TO

```
O01 D6. X5. Y3.
O02 D10. ; D is tool diameter
```

C. COMMAND FILE

The command file, or [.CM], contains all of the soft key commands Unidex 16 is to execute. For example, a command file may contain:

```
machine run FILE .PP auto
```

Now when the filename *FILE* is entered and <ENTER> is pressed, the *FILE* parts program will run in the [machine] [auto] mode.

A command file contains information that could have been entered via the softkeys when initiating a program run. It is only a convenience, since once this information is entered into a command file you need only enter the file name and press <ENTER> to initiate a program run. Following is a example of a command file we will call *SAMPLE .CM*:

```
machine run SAMPLE .PP SAMPLE .TO auto
```

The above command file will run the parts program *SAMPLE* in the auto mode, incorporating into it the tool file information contained in *SAMPLE .TO.*

An example of a more complex command file that we will call *COMBINE .CM*, is:

```
file delete COMBINE .PP
end
edit new COMBINE .PP
merge FIRST .PP
merge SECOND .PP
end update
machine run COMBINE .PP auto
```

When executed, the parts program *COMBINE* will be deleted, so that a new version of it can be created. Two files are merged in the file and it is then run in the auto mode. The next time the command file *COMBINE* is run, the former version will be deleted and yet another new one will be created.

How to create a parts program, tool file and a command file is detailed in chapter 9.

NOTE: To protect a file, refer to chapter 4, section 4-1 D.

NOTE: **Because you are working on a copy of a program when editing, be aware that the program will require twice its normal amount of memory.**

Any edit operation that causes you to exceed memory will give you an "out of memory" error.

CHAPTER 4: THE FILE MODE

SECTION 4-1 FILE MODE FUNCTIONS

When you enter the file mode by pressing the soft key [file], the following soft key selections will be displayed:

```
dir copy delete rename end --ETC--
```

A. DIRECTORY

If you choose the directory function ([dir]), you will see displayed:

```
/mms        /disk
```

Press the key which indicates the source of the directory you wish to view, either Unidex 16's internal memory ([/mms]) or the floppy disk ([/disk]). When you press <ENTER>, the first page of the directory will be displayed. Following the list of files in your directory is the amount of memory left and how much is available to you.

Press <NEXT PAGE> to see further listings within the directory and <PREV PAGE> to review the previous page. You may use <ROLL UP> and <ROLL DOWN> to move the directory up and down, one line at a time.

B. COPY

The soft key [copy] allows you to transfer data between port-A, port-B, the disk and Unidex 16 internal memory. One program or all files can be copied.

When you press [copy] you will see:

```
          <FILE>    all-file   /port-A   /port-B
```

1. <FILE>

<FILE> lets you know that a file name may be entered at this time, if that is what you wish to copy. Simply type in the file name and enter its type and location. An example would be:

```
FIRST
[.PP]
[/mms]
```

Unidex 16 will ask you, by means of the soft keys, to what location you wish to copy the file. This means the command entry area will display:

file copy FIRST .PP /mms to

The soft key command area will display:

```
          <FILE>            /port-A   /port-B
```

You can name a file (located in the internal memory or on a disk), or you can press port-A or port-B.

To name a file, you simply type in the file name, enter the program type and location. Example:

THIRD
[.PP]
[/mms] <ENTER>

When <ENTER> is pressed and the copy is made, you will see the message:

***** file copy done *****

If you choose a file name that already exists, Unidex 16 will tell you so. (You will not be able to overwrite it.)

You may try this with your sample program from chapter 2. Press [file] to enter the file mode. Then press [copy] and enter *FIRST* [.PP] [/mms]. At this point the command entry line will display:

file copy FIRST .PP /mms to

At the same time the soft key choices will be:

```
               <FILE>          /port-A   /port-B
```

To copy to another file at this point, enter the name *THIRD*, press [.PP], [/mms] and <ENTER>. The status line will display:

***** file copy done *****

To see *THIRD .PP* added to your directory, press [dir], [/mms] and <ENTER>.

To name port-A or port-B as the target, press the appropriate soft key. Again, when <ENTER> is pressed and the copy is made, you will see:

***** file copy done *****

If the file name you enter as the target already exists in the location specified, you will see:

***** target file exists already *****

As you can see, you cannot copy into an existing file.

If the file you name as a source does not exist in the location specified, you will see:

***** can't find source file *****

2. ALL-FILE

Pressing the [all-file] soft key indicates that all files are to be copied from one source to another. When [all-file] is pressed, the following soft keys will be displayed:

```
              /mms   /disk      /port-A   /port-B
```

Pressing [/mms] indicates that the files to be copied are stored in Unidex 16 memory. The following soft keys will be displayed at this point:

```
                     /disk        /port-A   /port-B
```

On the other hand, pressing [/disk] instead of [/mms] will display:

```
              /mms          /port-A   /port-B
```

You may now choose to what location the files will be copied, ie., Unidex 16 memory, disk, port-A or port-B.

When the copy is complete, you will receive this message on the CRT:

***** file copy done *****

NOTE: When copying all-files to memory from some source, check your [/mms] directory to make sure none of the files have the same name and type as those about to be sent. A duplicate file name with a duplicate program type will cause Unidex 16 to quit the copying function and send an error message.

3. PORT-A AND PORT-B

Port-A and port-B can be configured as RS-232 or RS-422 communication ports. Each can be connected to an external device such as a printer or a host computer. Certain rules must be followed, however. For details see section 4-2, File Transmission.

When copying from a port to the internal memory, you will be transferring programs from an external device to Unidex 16's memory. (For example, when downloading programs from a host computer to Unidex 16.) When files are transferred from Unidex 16's memory to a port, you are sending programs from the internal memory to an external device. (For example, when downloading programs from Unidex 16 to a printer.)

If the source from which you wish to copy is through port-A or port-B, you have the choice of copying the information to Unidex 16 memory or the disk.

Please refer to section 4-2 for details on file transmission.

C. DELETE

When in the file mode, the [delete] function will erase a file. Once [delete] is pressed, you will see displayed:

```
    <FILE>
```

This soft key display indicates that Unidex 16 is ready for the name, type and location of the file to be deleted. Enter:

SECOND
[.PP]
[/mms]

When you press <ENTER>, the *SECOND [.PP]* program in Unidex 16 memory will be erased and the soft key display will revert to the original file mode selections.

To check on this deletion, press [dir]. This will show you an updated version of your file directory.

D. RENAME

The soft key function [rename] is for one of three purposes:

1. Changing the name of a file
2. Protecting a file
3. Unprotecting a file

In order to change a file name, press soft key [rename]. The soft key display will show:

```
    <FILE>
```

At this point, enter the file name you wish to change. Example:

FIRST
[.PP]
[/mms]

Unidex 16's command entry area will display:

file rename FIRST .PP /mms to

The soft key command area will display:

```
<FILE>    protect     no-prot
```

You may now enter the name to which you wish to change. Example:

FIFTH
[.PP]
[/mms] <ENTER>

To check the directory in order to verify the name change, press [dir], [/mms] and <ENTER>.

The second purpose of [rename] is to protect a file. When a file is protected, it cannot be deleted or given another file name. It can, however, be edited.

In order to protect a file, press [rename]. Then enter the file to be protected, as well as file type and location. In the following example, pressing [rename] will display:

```
<FILE>
```

Enter:

FIFTH
[.PP]
[/mms]

The soft key display will offer:

<FILE>   protect   no-prot

Pressing the [protect] soft key and <ENTER> will protect your file. It will remain protected until you unprotect ([no-prot]) it.

To unprotect your file, simply press [rename] again. Enter file name, type and location. The soft keys will offer:

<FILE>   protect   no-prot

Enter [no-prot] to get your file out of protection.

E. END

Soft key [end] is self-explanatory. Press [end] and <ENTER> to quit the file mode and revert back to the original mode functions:

edit file machine parameter test

On the other hand, [--ETC--] will display two more file mode functions. They are:

format   verify

F. FORMAT

The soft key function [format] can be used for one of two purposes:

1. **Formatting a disk**
2. **Formatting Unidex 16 memory**

When you press [format], you will be given the choice of internal memory or a disk:

```
                    /mms        /disk
```

1. DISK

If you press [/disk] and <ENTER>, Unidex 16 will format your floppy disk. A new disk must be formatted before it is used.

CAUTION: [Format] [disk] will also destroy anything previously recorded on the disk.

2. MEMORY

Unidex 16 always reserves some user's memory as the stack, for jumps, variables and subroutines that may occur within the parts program. This reserved amount of memory cannot be used for any other purpose.

The amount of memory reserved for these functions is user-programmable within the parameter [.EEPROM] mode. (For details on setting parameters, see chapter 6, The Parameter Mode.)

Once a file is active through the [run] mode, it will remain active as long as a power down or <RESET> does not occur. The only way to inactivate the file and free the memory reserved for its execution is to format the internal memory. Press [format] and [/mms]. The status line message will ask:

```
            Warning?? Stop & Destroy Working Piece??
```

You will be given the choice:

```
                         yes                no
```

This protects you from inadvertently clearing the memory (stack). [Yes] will cause the parts program to quit running and cause Unidex 16 to quit the machine mode and inactivate the file that was currently running. If you check the internal memory directory, you will see the "A" (for Active) which is listed beside an active file is now gone. Formatting does not erase any files within the memory.

Refer to chapter 10 to find how much memory is needed for functions such as subroutines, repeat loops, etc.

G. VERIFY

In order to check one file's contents against another's, use the [verify] soft key command.

When you press [verify], the screen displays:

```
  <FILE>
```

you may now enter the source file. Example:

```
FIFTH
[.PP]
```

```
[/mms]
```

Unidex 16's command entry area will display:

```
file verify FIFTH .PP /mms against
```

You may now enter the target file. Example:

```
THIRD
[.PP]  <ENTER>
```

NOTE: The [/mms] or [/file] entry is unnecessary here, since the target file to be verified must be in the Unidex 16 memory.

If the two files are the same, you will see:

```
*** STATUS:  files match each other ***
```

If the two files do not match, you will see:

```
*** STATUS:  files do not match ***
```

You can try this by verifying your file *FIFTH* against file *THIRD*. Since one was copied from the other, you will see the message:

```
*** STATUS:  files match each other  ***
```

SECTION 4-2 FILE TRANSMISSION

TRANSMITTING FILE(S) FROM UNIDEX 16 TO PORT

A. TRANSMITTING ONE FILE FROM UNIDEX 16

When transmitting a file from Unidex 16 to port A or port B, Unidex 16 *automatically* places a percent sign (%) before the program (after the title). This is your begin-file character and is provided by Unidex 16.

The *end-of-file* character is user-programmable. This character is entered into the EEPROM (parameter 110). Then, when a file is transmitted, both of these characters are placed into the file by Unidex 16.

When transmitting a file from Unidex 16 to a port, follow this sequence:

[file]
[copy]
FILE NAME
FILE TYPE - [.PP], [.TO] or [.CM]
FILE LOCATION - [/mms] or [/disk]
[/port-A] or [/port-B]
< ENTER >

Using the parts program *SAMPLE* as an example, the file will be received by the port-A or port-B device as:

```
file name        :  @SAMPLE@
file type        :  .PP
length (bytes)   :  110
last edit date   :  08-12-1986  10:45
```

```
% G90 X1. Y2. F200.
Z-.2
G4 F.5
G91 G1 G70 X10. F200.
X-10.
Z2.
M30
(Programmable end-of-file character)
```

NOTE: The "@" symbol that is seen before and after the file name is automatically inserted by Unidex 16. This is true for the Begin-File (%) and programmable End-of-File character as well.

B. TRANSMITTING ALL FILES FROM UNIDEX 16

When all files are to be transmitted, follow this soft key sequence:

```
[file]
[copy]
[all-file]
[/mms] or [/disk]
[/port-A] or [/port-B]
```

Again, Unidex 16 will automatically enter the begin-file character (%) as well as the end-of-file (parameter 110) and end-all-files (parameter 111) characters that are set in the EEPROM.

TRANSMITTING FILE(S) FROM PORT TO UNIDEX 16

C. TRANSMITTING ONE FILE TO UNIDEX 16

When transmitting a file from port-A or port-B to Unidex 16, all of the characters necessary for transmission must be entered by the programmer. This means the begin-file and end-of-file characters entered by Unidex 16 when Unidex 16 is transmitting, must now be entered by the user. For example, to send one file to Unidex 16, follow this sequence:

[file]
[copy]
[/port-A] or [/port-B]
FILE NAME (required)
FILE TYPE (required)
[/mms] or [/file]

Unidex 16 expects to see the following from the host computer:

% (required)
|
one-file
|
end-of-file character (required) — This character must match the end-of-file character progam in Unidex 16's EEPROM parameter #110.

The file's name and type while in the host computer is not important to Unidex 16; however, the name and type of that file when in Unidex 16 memory (or on the disk) must be specified. Also, the begin-file (%) and programmable end-of-file characters *must be entered into the file by the user before transmission. The end-of-file character must match the one programmed into the Unidex 16 EEPROM (parameter 110).*

D. TRANSMITTING ALL FILES TO UNIDEX 16

When transmitting all files from a host computer through port A or port B to Unidex 16, all characters necessary for transmission must be entered by the user. To do so, follow this sequence:

[file]
[copy]
[all-file]
(from)
[/port-A] or [/port-B]
to
[/mms] or [/disk]

The begin-file (%) and programmable *end-of-file* character must have been inserted into each file before transmission to Unidex 16. The last file to be sent must have an *end-all-file* character after it's end-of-file character if transmission is to operate properly.

E. TIME-OUT

In order to prevent the system from "hanging-up" when files are being transmitted between a port and Unidex 16, Unidex 16 will terminate the "transmission active" state. The time that elapses before this Time-Out is user-programmable and is set in the EEPROM, parameter #161. The number entered is in seconds and may be to 65,535. *If zero is the value of parameter #161, the default value is 5 minutes.*

SAMPLE TRANSMISSION PROGRAM

The following is a parts program written on an IBM PC computer in the Basic Language. It illustrates the type of commands required for the proper transmission of a parts program.

```
10 OPEN "COM1:9600,N,8,1" AS #1
20 PRINT#1,"%";
30 PRINT#1,"; DRAWING NO. - 1333-1003"+CHR$(10);
40 PRINT#1,"G1 F100."+CHR$(10);
50 PRINT#1,"X10. Y12. Z12."+CHR$(10);
60 PRINT#1,"G4 F1."+CHR$(10);
70 PRINT#1,"X-10. Y-12. Z-12."+CHR$(10);
80 PRINT#1,"G4 F1."+CHR$(10);
90 PRINT#1,"M47"+CHR$(10);
100 PRINT#1,+CHR$(9);
110 STOP
120 END
```

EXPLANATION OF BASIC PROGRAM

Line 10 The first line establishes the baud rate (9600), parity bit (N for none), the bits per character (8) and the stop bit (1). These should be programmed here to match the parameter settings in your Unidex 16 EEPROM.

Line 20 The percent sign (%) is the start byte. This is fixed, not user programmable. It is important to remember the semicolon (;) at the end of the line, in order to eliminate the carriage return/line feed.

NOTE: Unidex 16 uses a line feed (ASCII Code 0AH) as an end-of-block internally. Therefore, when *inputting* data from port A or port B, be certain that there is only one line feed at the end of each block.

When *outputting* to port A or port B, however, in order to match a conventional terminal display or printer function, Unidex 16 outputs both line feed (0AH) and carriage-return (0DH) at the end of each block.

Line 30 to Line 90 The information enclosed in quotation marks will be sent to Unidex 16. The CHR$(10) provides the line feed (which is the end-of-block in Unidex 16 files).

Line 100 The end-of-file character is user-programmable. You must program it in the EEPROM, parameter #110. In this sample program, CHR$(9) signals end-of-file transmission.

CHAPTER 5: THE MACHINE MODE

SECTION 5-1 MACHINE MODE FUNCTIONS

When you press soft key [machine] and <ENTER>, the following options are available. You may:

1. Press <NEXT PAGE> to view subsequent pages on the CRT.
2. Use the Axis Select Switch and Home button to send all axes home.
3. Select a sub-mode by selecting one of the following:

```
mdi  jog  run  single/auto   end   --ETC--
```

A. JOG

[Jog] is the most basic of the machine modes. It allows you to manually position your workpiece by use of the arrow keys, one axis at a time.

If in a limit, [jog] the axis out by using the arrow keys.

You may use the Axis Select Switch and Home button in the jog mode to send any axis home.

NOTE: The arrow keys control the movement of axes labeled X, Y and Z. Axes labeled U, V and W are controlled by soft key commands +U, -U, +V, -V, +W and -W, listed among the jog mode's soft key commands. For X, Y and Z moves, the assignment of + or - to an arrow key can be selected through the EEPROM parameter #160. For details, see chapter 6.

Pressing [jog] displays the following display:

```
minimum  low  medium  high  top --ETC--
```

The soft keys [minimum], [low], [medium], [high] and [top] all refer to the speed at which the axes will move.

ENGLISH
(.0001 INCH MACHINE RESOLUTION)

	Distance In Inches	Feedrate In Counts/minute
[Minimum]	.0001	267.8
[Low]	.001	5,000
[Medium]	.01	50,000
[High]	.1	500,000
[Top]	1.0	1,250,000

METRIC
(1 MICRON MACHINE RESOLUTION)

	Distance In Millimeters	Feedrate In Counts/minute
[Minimum]	.001	267.8
[Low]	.01	5,000
[Medium]	.1	50,000
[High]	1.0	500,000
[Top]	10.0	1,250,000

At this point, select [low] speed and set the Manual Feedrate Override Switch to 50% until you are more familiar with your system.

Press [--ETC--] to see:

```
+U  -U  +V  -V  --ETC--
```

NOTE: The following (+U, +V and +W axes) require an additional indexing board.

1. +U, -U AXIS

Manual positioning of the U axis.

2. +V, -V

Manual positioning of the V axis. Press [--ETC--] to see:

```
+W  -W  --ETC--
```

3. +W, -W AXIS

Manual positioning of the W axis.

NOTE: Before jogging the U, V and W axes, select a speed through the [minimum], [low], [medium], [high] or [top] speed selection soft keys. No movement will occur without a speed selection.

Press [--ETC--] to see:

```
set-rtn  return  back-end  end  --ETC--
```

4. SET-RTN

[Set-rtn] enables you to set a temporary return point. You can move away from this point and Unidex 16 will remember the coordinates. When you press [return], Unidex 16 will send the axes back. In order to set [set-rtn], jog the axes to the desired position. Press [set-rtn]. If you move to a new position and set a new [set-rtn], the former setting will be overwritten.

5. RETURN

[Return] will cause the axes to move to the position established with the [set-rtn] command at whatever jog feedrate you have set.

6. BACK-END

[Back-end] is only seen when you quit running a parts program in the run mode in order to enter the jog mode.

When finished the jog mode, press [back-end] to return the axes to the positions they occupied when you quit the run mode.

The sequence of axes movement (which axis moves first) is set in the parameter [EEPROM] mode (refer to section 6-2) and operates the same as with [return]. All axes will move at the last jog feedrate.

7. END

[End] causes Unidex 16 to exit the jog mode, but unlike [back- end], will not move the axes back to their original positions if in the middle of a parts program. Press [--ETC--] to see:

```
s-stop s-cw/on s-ccw/on rpm-dn rpm-up --ETC--
```

NOTE: The following only apply to a system equipped with a MTI-16 board. These functions are designed for machine tool applications where spindle, clamp or coolant are frequently used. These functions are implemented using the S and/or M functions to activate the MTI-16 board.

8. S-STOP

The soft key function [s-stop] will stop the spindle by outputting an M5 command. This command is recognized by the MTI- 16 board, which then cancels the [s-cw/on] or [s-ccw/on] command.

9. S-CW/ON

The [s-cw/on] soft key turns the spindle on and causes it to rotate in a clockwise direction through the M3 command.

10. S-CCW/ON

The [s-ccw/on] command turns the spindle on and causes it to rotate in a counterclockwise direction through the M4 command.

11. RPM-DN

The soft key command [rpm-dn]* slows down the spindle speed through the M40, M41, M42, M43 and M44 commands.

12. RPM-UP

The soft key command [rpm-up]* speeds up the spindle speed through the M40, M41, M42, M43 and M44 commands. Press [--ETC--] to see the following screen of soft key commands.

```
        clamp-on  clamp-off cl1-on  cl2-on  cl-off  --ETC--
```

* For [rmp-up] and [rmp-dn], the speed change amount is programmed in the parameter [EEPROM] mode (see section 6-2).

13. CL1-ON

[Cl1-on] turns on the coolant labeled #1 through the M7 command.

14. CL2-ON

[Cl2-on] turns on the coolant labeled #2 through the M8 command.

15. CL-OFF

[Cl-off] turns off the coolant through the M9 command.

16. CLMP-ON

[Clmp-on] causes the clamp holding the spindle to close through the M10 command.

17. CLMP-OFF

[Clmp-off] causes the clamp holding the spindle to open through the M11 command.

B. MDI (MANUAL DATA INPUT)

[Mdi] is an immediate mode. That is, the commands entered are executed as soon as <ENTER> is pressed.

The mdi commands are retained in the buffer and will be executed every time <ENTER> is pressed. When a new command block is entered, the previous one is overwritten. This being the case, no jumps or subroutines are possible within the mdi mode.

[Mdi] is more powerful than [jog] because more than one axis can be moved at a time. You may jump to mdi while in the middle of running a parts program. You may select new statuses such as feedrate, spindle speed, G codes and M functions. The statuses selected while in mdi will override those of the parts program. At this point, your soft key selections are:

```
        <CMD>              back-end   end   --ETC--
```

Enter a block of commands and execute them in the mdi mode. For example:

G1 G91 X1.0 Y1.0 Z1.0 U1.0 V1.0 W1.0 F50.0 E50.0

Any axes not applying to your system will just be ignored.

1. BACK-END

[Back-end] will cause Unidex 16 to end mdi and resume execution where it had left off, if in the middle of a parts program. All statuses will revert back to their previous values (except M functions, spindle speed and T functions, which will retain their updated values). All axes will move back to the positions they occupied when the parts program was interrupted. The axes will return at the mdi feedrate.

The sequence in which the axes move back to their original positions is programmed in the parameter [EEPROM] mode (see section 6-2).

[Back-end] is only displayed when Unidex 16 jumps from the run mode to jog or mdi.

2. END

[End] will cause Unidex 16 to end mdi. However, the previous statuses will remain overridden by the mdi statuses.

Press [--ETC--] to see:

```
        s-stop  s-cw/on  s-ccw/on  rpm-dn  rpm-up  --ETC--
```

The soft key commands shown above serve the same purpose as they did in the jog mode.

C. RUN

The [run] soft key allows you to run a complete parts program. The screen will display:

```
   <FILE.PP>                  last-run              last-edit
```

You may press [last-run] to run the most recently executed file, or [last-edit] to run the most recently edited file. Or, you may enter the name of the parts program to be executed. The following will be displayed:

```
        <AUX-FIL>  .G0ovr5%  .G0ovr25%  auto  dry-run
```

The <AUX-FIL> indicates that if a tool file and/or auxiliary parts program accompanies your parts program, it may be entered now. If both are to be included, the tool file must be entered *before* the auxiliary parts program.

Tool files are explained in chapter 9.

The auxiliary parts program will be scanned to prepare for variables, entry points and subroutines, as is the main parts program. It doesn't matter which parts program defines variables, entry points or subroutines, because the two programs will be treated as one in this respect. Do not duplicate definitions in the two programs, however, because that will cause errors just as it does in a single program.

The main parts program and the auxiliary parts program are not treated as one continuous program when executing. The only way to get from the main program to the auxiliary program or from the auxiliary program to the main program, is by using jumps to entry points or subroutine calls and returns. For example, the first block of the auxiliary program will *not* be executed after the last block of the main program unless the last block of the main program is a jump to an entry point defined to be the first block of the auxiliary program. The auxiliary parts program is useful for conserving memory, because commonly used routines will not need to be duplicated in several programs.

If, instead of running a program, you want a dry-run of your parts program, you must utilize the functions .G0ovr5%, .G0ovr25% and dry-run. (These will be explained in the following subsection.)

Pressing <ENTER> will display:

```
mdi jog run auto/single end --ETC--
```

The soft key [auto] allows you to choose if you will let the program run in the *single mode* or change it to the *auto mode*. Choose auto or single and <ENTER> to execute your parts program.

If in [auto], the entire program will run, but if in [single], only the first block will run. To run subsequent blocks, <CYCLE START> must be pressed.

<CYCLE START> is also required if the program is run initially in the [single] mode and is then switched to [auto] (or vice versa). After the mode is switched, you need to press <CYCLE START> to continue the program.

In both cases, pressing <SHIFT> <CYCLE START> causes Unidex 16 to execute from the beginning of the program.

The soft key [end], of course, causes Unidex 16 to quit the machine mode and revert back to the original display, but the program will keep running.

Pressing [--ETC--] displays the following soft key selections:

```
            .G0ovr5%  .G0ovr25%    dry-run   --ETC--
```

a. .G0ovr5%

Pressing the soft key [.G0ovr5%] indicates to Unidex 16 that you want your program dry-run to be executed at 5% of the rapid traverse speed, and not at the programmed feedrate. This applies when the dry-run feedrate is inhibited. (G0 indicates rapid traverse speed.)

b. .G0ovr25%

The same as G0ovr5% except this soft key command indicates to Unidex 16 that you require the program to be executed at 25% of the rapid traverse speed.

c. DRY RUN

Pressing [dry-run] will display the following choices:

```
                      clr-set      set       clear
```

1. Set

[Set] allows you to set all of the functions to be inhibited during a dry run. For example, if you press [set] and then enter Z and X, those two axes will remain stationary while the parts program is running. Example:

[set]
XZ <ENTER>

You can inhibit all functions by entering [set] and then entering [all]. Typically, when all functions are set, the dry run is utilized as a syntax check.

NOTE: When a syntax check results in an error, check the page which displays the parts program. The block pointer will be located either on the block which contains the error, or on the one above it.

2. Clr-set

[Clr-set] allows you to clear all inhibited functions and set new ones. For example, if X, Y, Z and F are all inhibited and you wish to clear all of them except F, press [clr-set], enter F and press <ENTER>. This clears all previous settings and sets F again.

3. Clear

[Clear] allows you to clear restricted functions. You may enter those to be cleared individually, or clear all of them by pressing the soft key [all] when it is displayed.

It will be helpful at this point to enter a sample program (edit mode) and run it (machine mode). Before running the program, make sure the axes are at the positive end of travel. If your home position is at the negative end of the axis, reverse all moves within the program from positive to negative.

(REF,X,Y); or appropriate axes

```
G90 G1 X0.5 Y0.5 F50.0; Reverse the direction of all moves
        if required
X1.5
X1.0 Y0
X0
M2
```

After entering this program in the edit mode, run it in the machine, single mode. If it runs well, run it again in the auto mode. Axes will move in the shape of a parallelogram.

D. SINGLE/AUTO MODE

[Single/auto] is not an individual mode in itself. It is an extension of the run mode.

The [single/auto] soft key toggles between the single and the auto mode. The one seen in the status line area of the page is active, while the one in the soft key command area is not. To toggle between modes, press the soft key [single/auto].

Pressing [single/auto] displays the following soft keys.

```
            .G0ovr5%    .G0ovr25%       dry-run
```

NOTE: To run a program directly from the edit mode, use the edit softkey [up-&-run]. It is explained in chapter 3, The Edit Mode.

SECTION 5-2 MACHINE MODE OPTIONS

The machine mode provides several means of adjusting parts program execution, all available from the front panel. They are:

1. Manual Spindle Speed Override
2. Manual Feedrate Override
3. Axis Select and Home
4. Block delete
5. Optional stop

A. MANUAL SPINDLE SPEED OVERRIDE

Spindle speed adjustment overrides the programmed spindle speed. Therefore, when running a program, you may override the programmed spindle speed by turning the spindle speed knob. The values range from 0 to 200. The values represent:

0	-	0% of programmed spindle speed
100	-	100% of programmed spindle speed
200	-	200% of programmed spindle speed

In many cases the highest or the lowest spindle speed is limited by the type of spindle used.

B. MANUAL FEEDRATE OVERRIDE

MFO overrides the programmed feedrate. When running a program, you may override the programmed feedrate by turning the feedrate knob. The values range from 0 to 200 and as with MSO, represent values ranging from 0 to 200%.

C. AXIS SELECT AND HOME

The Axis Select Switch and Home button work together and apply to most machine modes. If you want an axis to be sent home, simply select the appropriate axis on the Axis Select Switch and then press the Home button.

Whether the machine is running in mdi, jog or single, Unidex 16 will finish the block it is executing and then send the axis home. (If it is in the auto mode, however, Unidex 16 will ignore this function in order to protect the workpiece.) After sending the axis home, Unidex 16 will await further commands from you.

D. BLOCK DELETE

The block delete switch gives you the option, when running a parts program, of skipping certain blocks of program.

When editing the parts program, precede these optional blocks with a slash (/). When running this program in the machine mode, if the Block Delete Switch is on, these blocks will be omitted. If the switch is off, they will not be omitted. Example:

```
G1 X2. Y3.
/G1 X1. Y1.
```

In the above example, the second block will be skipped if the Block Delete Switch is on.

Do not use Block Delete (/) in a block containing ICRC, jumps, subroutines or defined entry points. Use conditional jumps instead (see section 12-4 B).

E. OPTIONAL STOP

During a machine run, the Optional Stop Switch gives you the option of stopping program execution whenever an M1 is encountered within the parts program.

Simply enter an M1 (optional stop) rather than an M2 (unconditional stop) when editing the parts program, if you wish to use the optional stop. Then, when M1 is encountered, if the Optional Stop Switch is on, the program will stop running. If it is off, the program will continue to run.

SECTION 5-3 MACHINE MODE DISPLAYS

In the machine mode, there are several pages of displays. Each gives you pertinent information concerning statuses and parameters. These pages are:

1. **Tracking display**
2. **Position display**
3. **Parts program**
4. **G/H, F/E, S, T**
5. **Axis information**
6. **Miscellaneous**
7. **User-programmable message**
8. **Custom display page**

In the machine mode, these pages can be viewed by pressing <NEXT PAGE>. The pages are described in the following subsections. The name of the page being viewed will be highlighted in the lower portion of the CRT.

A. TRACKING DISPLAY

The *tracking display* shows all six axes and the current position of each. As the axes move, the values on the page change as well. The tracking display page is depicted in figure 5-1.

```
X  2.3410      U  3.4891
Y -1.9339      V  2.3912
Z  2.5782      W -1.2855
```

FIGURE 5-1: TRACKING DISPLAY

The example shown in figure 5-1 is utilizing all six axes.

B. POSITION DISPLAY

The *position display* shows the "program position", the "machine position", the "position to be", the "current position" and the "distance to go".

The "program position" shows the positions entered in the parts program in inches or millimeters. These are the coordinates of the workpiece contour that the Unidex 16 is programmed to follow at the end of the current block. The position is referring to the user's reference zero (floating home).

The "machine position" shows the physical position of the axes with respect to the home position, in machine steps, at the end of the current block.

The "position to be" is the position to which the axes are going to be moved, with respect to the user's reference zero. It differs from "program position" by the amount of tool offset or ICRC currently employed.

The "current position" and the "distance to go" should, at any given moment, add up to the "position to be". The "current position" is the current axis real-time position.

The "distance to go" is the distance remaining to be finished in the current block of parts program.

The position display is depicted in figure 5-2.

	program position	machine position	position to be	current position	distance to go
<X>	0.200	1.800	0.200	0.200	0.000
<Y>	0.200	10.500	0.200	0.200	0.000
<Z>	-3.000	1.500	-3.000	-3.000	0.000
<U>					
<V>					
<W>					

FIGURE 5-2: POSITION DISPLAY

Figure 10-4 of chapter 10 further illustrates the position display.

C. PARTS PROGRAM

The *parts program display* shows your parts program as it is being executed.

The block currently being executed will be highlighted (figure 5- 3).

```
*     G23 F200
* (DVAR,AA)
*     AA=2
```

FIGURE 5-3: PARTS PROGRAM DISPLAY

As can be seen in the above figure, the second block of commands is currently being executed, since it is the one that is highlighted.

D. G/H, F/E, S, T

This display allows you to view current:

1. **G/H code**
2. **Feedrate**
3. **Spindle speed**
4. **Tool information**

The H and E codes are counterparts for the G and F codes. G and F apply to the X, Y and Z axes; H and E apply to the U, V and W axes.

The statuses that are in effect are the ones that are highlighted on the screen. These statuses may be entered through the parts program or mdi.

The G/H,F/E,S,T page is shown in figure 5-4.

```
G & H        G00/G01/G02/G03  G23/G24   H00/H01/H02/H03  H23/H24
function  :  G17/G18/G19      G70/G71   H17/H18/H19      H70/H71
             G08/G09          G90/G91   H08/H09          H90/H91
             G40/G41/G42      G98/G99
CanCycle:    G76/G77/G78/G81/G82/G83/G84/G85
Feedrate  :  F =    0.0                 E =    0.0
S function:  S =    cw/ccw
T function:  tool # =                   diameter =
             offset =  X        Y        Z
             U         V        W
```

FIGURE 5-4: G/H, F/E, S, T DISPLAY

An explanation of the codes follows:

G00 - Rapid traverse (top speed)
G01 - Linear interpolation
G02 - Circular movement, CW
G03 - Circular movement, CCW
G08 - Gradual acceleration
G09 - Gradual acceleration/deceleration
G17 - X and Y axes on the same plane
G18 - Z and X axes on the same plane
G19 - Y and Z axes on the same plane
G23 - Corner rounding
G24 - Non-corner rounding
G40 - Cancel G41 and G42
G41 - Intersectional cutter radius compensation (left)
G42 - Intersectional cutter radius compensation (right)
G70 - Axes travel in inches
G71 - Axes travel in millimeters
G76 to G78 - Canned cycles
G81 to G85 - Canned cycles
G90 - Absolute mode
G91 - Incremental mode
G98 - Temporary return *
G99 - Return to original position *
F - Feedrate
S function - Spindle speed, CW and CCW
T function - Tool number, diameter and offsets for each axis

* Applies to canned cycles G81 and G85 only

E. AXIS INFORMATION

The *axis information display* (figure 5-5) shows information concerning:

1. **Assigned axes**
2. **Mirrored axes**
3. **Dry run**
4. **Scaling function and factors**
5. **Synchronization**

Whichever statuses are in effect are highlighted. An explanation for each status is as follows:

1. ASSIGNED AXES

Axes pertaining to your system are highlighted. This information is entered through the parameter mode.

2. MIRRORED AXES

It may shorten the length of your program to mirror some axes' movements. Which axes are mirrored is displayed on the axis information screen. This information is entered through the parts program, using command (MIR). (See section 8-2.)

```
  Exist  : ---X--- ---Y--- ---Z--- ---U--- ---V--- ---W---
Mirror : no/yes no/yes no/yes no/yes no/yes no/yes
D run  : no/yes no/yes no/yes no/yes no/yes no/yes
S fact :
S func : no/yes
Sync   : no/yes
U int  : INT4- INT3- INT2-   INT1-   CIRQ-
```

FIGURE 5-5: AXIS INFORMATION DISPLAY

3. DRY RUN

When executing a dry run, some axes (or all) may be inhibited. When an axis is inhibited, it is displayed on the axis information page, but no movement occurs. If inhibited, the "yes" portion will be highlighted. Dry run information is entered through the machine mode.

4. SCALING

It may be convenient to run a program on a larger or a smaller scale. If so, this information would be entered in the parts program or through mdi, using the command (SCF). (See section 10-2.) The scaling function on the page will display a highlighted "yes". The scaling factor will display the number by which the axes movement will be scaled. Scaling can range from 0.00001 to 99.99999. This value is entered through the parts program also.

5. SYNCHRONIZATION

If axes from group XYZ are to be synchronized with axes from UVW, you will need Synchronization. The axes will then begin and end at exactly the same time. Synchronization must be entered through the parts program or mdi. Command (SYNC,1) will turn it on and (SYNC,0) will turn it off.

If synchronization is on, the axis information page will display a highlighted "yes" beside "Sync". Synchronization only applies if your system is utilizing axes from both groups.

NOTE: If "yes/no" appears only one time beside a given parameter, it means that the selection applies to all axes.

6. INTERRUPT

You may program an interrupt in your parts program. Which interrupt source (INT4, INT3, INT2, INT1, or CIRQ) as well as which interrupt option you choose, is shown on the axis information display.

For details on interrupts, see section 10-2.

F. MISCELLANEOUS

The *miscellaneous display* (figure 5-6) pertains to individual programs. It gives information on:

1. I/O FORMAT

The I/O bus format (8 bits) gives you the choice of utilizing either BCD or binary when transmitting or receiving information over the bus. The bytes selection lets you choose a 1, 2, 3 or 4 byte format when transmitting information, as well.

This information is entered through the parts program with the command (IOFT). See the *Unidex 16 Hardware Manual* for more information on the I/O format.

2. CUTTER COMPENSATION

Cutter compensation compensates for the radius of the tool.

This information is entered through the parts program or mdi by use of the command (CCP). (See section 10-3.)

Figure 5-6 shows the Miscellaneous page.

```
        $ I/O  format      :  BIN/BCD
               bytes       :  1/2/3/4
   Cutter compensation   :
```

FIGURE 5-6: MISCELLANEOUS PAGE

G. USER-PROGRAMMABLE MESSAGE

Only if the parts program has a message will MSG become one of the pages. It will then be seen displayed on the CRT.

To add a message to your parts program, refer to section 10-2.

H. DISPLAY PAGE

The last machine mode status screen is available for customer-defined displays. This is page 8 and is only seen if programmed with a display (DISP) command in a parts program. (For details on the display command, refer to section 10-2.)

This page does not always come up automatically, as does the Message Command (explained in section 10-2 also). If programmed with a DISP command, however, it can be programmed to come up automatically. When this page is programmed, use <NEXT PAGE> to access it, as you would the other status screens.

The location of the information written to this screen is defined by programming the row and column of the first character's position, starting from the left of the screen. The data will overwrite previous data, up to the length of the new data. If the previous data was longer in length, the excess characters will not be overwritten and will remain on the screen. To avoid this situation, be sure to clear the proper number of spaces, as explained in section 10-2.

Rows may range from 1 to 12 and columns from 1 to 60. If data is longer than the line length of the screen, the data will "wrap around" to the next line on the screen.

There are commands to clear the display screen, deactivate it, or bring it up automatically. Refer to section 10-2 for details.

SECTION 5-4 HANDLING LIMITS

As mentioned in section 5-1 A, if an axis runs into a limit, use the arrow keys (for the X, Y and Z axes) or the softkeys (for the U, V and W axes) to jog it out of the limit.

CHAPTER 6: THE PARAMETER MODE

Pressing the soft key [parameter] yields two choices:

```
            .EEPROM      clock          end
```

SECTION 6-1 CLOCK

Choosing [clock] allows you to read or set the day-month-year-hour-minute-second. To read the date and time, press [clock] and <ENTER>.

To set a new date and time, press [clock] and the enter the date and time in the following format:

dd-mm-yy-hh-mm-ss <ENTER>
as, 11-06-87-10-35-45 <ENTER>

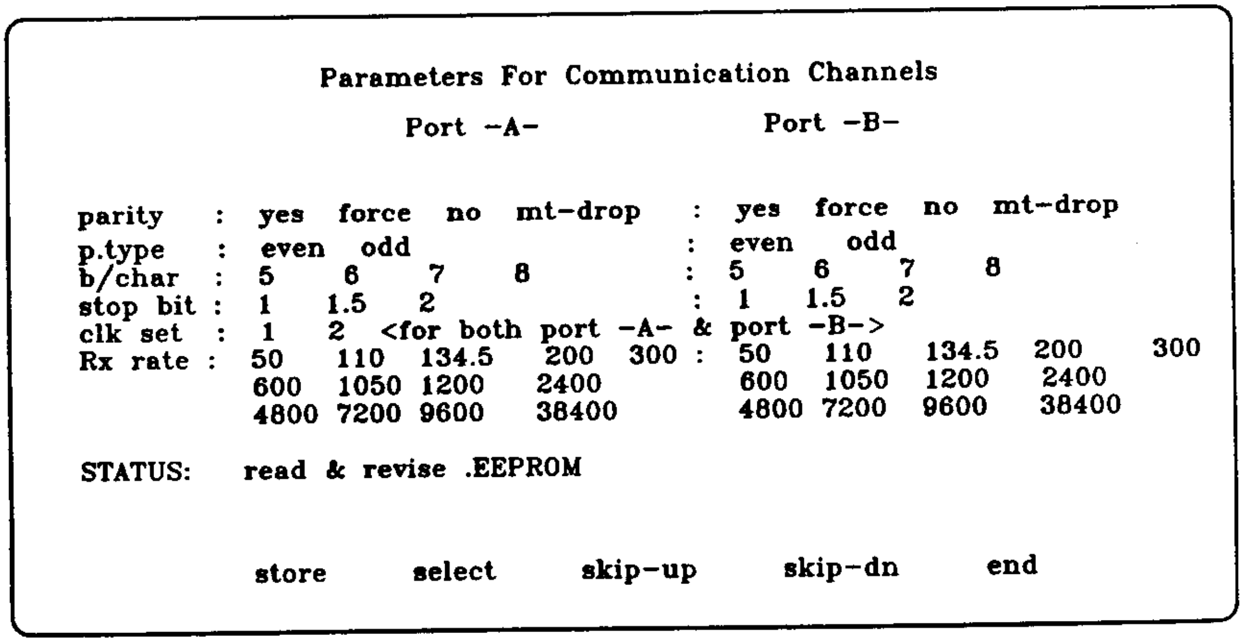
To exit the clock mode, press the soft key [end] and then press <ENTER>.

NOTE: System <RESET> does not reset the date and time. The clock is reset when Unidex 16 is shut down unless the Unidex 16 Real-Time Clock has been installed. (See section 13-3 for more details on the Unidex 16 Real-Time Clock option.)

SECTION 6-2 EEPROM

Selecting [parameter] again, press [.EEPROM] and <ENTER>. The parameters available to Unidex 16 will be displayed. The first page shows all the parameters for communication through port-A and port-B. The highlighted parameter is the one that is set. Figure 6-1 illustrates what the first page of parameters displayed on the screen should look like.

Flashing sections represent the parameters which are in effect. To change the status, [skip-up] or [skip-down], or use the arrow keys, to reach the appropriate line. Press [select] to jump forward to each new selection. Press [store] to set selection as the currently active parameter. (Press <NEXT PAGE> if you wish to view the next page of parameters.)



```
                    Parameters For Communication Channels
                              Port -A-               Port -B-

parity    :  yes  force  no   mt-drop   :  yes  force  no   mt-drop
p.type    :  even  odd                  :  even  odd
b/char    :  5    6     7     8         :  5    6     7     8
stop bit  :  1    1.5   2               :  1    1.5   2
clk set   :  1    2   <for both port -A- & port -B->
Rx rate   :  50   110  134.5   200  300 :  50   110   134.5   200     300
             600  1050 1200   2400         600  1050  1200    2400
             4800 7200 9600   38400        4800 7200  9600    38400

STATUS:    read & revise .EEPROM


           store      select      skip-up      skip-dn      end
```

FIGURE 6-1: FIRST EEPROM PAGE

EEPROM PAGE

Parity - Parity bit off or on
Parity type - Odd or even parity
Bits/character - 5, 6, 7 or 8 bits per character
Stop bit - Stop bit 1, 1.5 or 2 bits
Clockset - Each clock setting gives a different set of baud rates. Clock set 1 gives baud rates shown in figure 6-1, while Clock set 2 gives:

75 110 134.5 150 300
600 1200 2000 2400
4800 1800 9600 19200

Rx - Receiving baud rate *
Tx - Transmitting baud rate *

*Tx and Rx may be different values

Referring to the following pages, some information is entered as binary numbers and some is entered as decimal numbers. The decimal numbers fall into different ranges. The "TYPE" column gives this information as A, B, C and D:

TYPE:

A - BINARY NUMBERS (0 or 1)
B - DECIMAL NUMBERS RANGING FROM 0 - 255
C - DECIMAL NUMBERS RANGING FROM 0 - 65,535
D - DECIMAL NUMBERS RANGING FROM 0 - 16,000,000

NOTE: The bits of the parameter are designated as 0 - 7, reading right to left:

7 6 5 4 3 2 1 0 (Position of bits)
0 0 0 0 0 0 0 0 (Parameter bits)

NOTE: The 100 group of parameters applies to general information. The 200, 300 and 500 group apply to machine and axis control. The 400 group applies to MST functions.

PAGE 1

PARA#	EXPLANATION	TYPE
100	End address of user's RAM for basic system (65535 for 16K of user's RAM; add 32768 for each additional 32K of user's RAM)	D
101	Time after which CRT goes blank (in minutes) after the last key is depressed	C
102	Soft key attributes 7 6 5 4 3 2 1 0 0 0 0 0 0 0 0 0 (Not Underlined) 7 6 5 4 3 2 1 0 0 0 0 0 0 0 1 0 (Underlined) 7 6 5 4 3 2 1 0 1 1 0 0 0 0 0 0 (Reverse Video)	A
110	ASCII character which designates the end of file transmission	A
111	ASCII character which designates the end of *all-file* transmission	A

PARA#	EXPLANATION	TYPE
112	Status line message duration 255 = Infinite 0 to 254 = Time range in seconds	B
120	Reserved	
121	Joystick or Parallel Keyboard Option 00000000 - Parallel Keyboard option 00000001 - Joystick option	A
122	Reserved	
130	Drive format (1-9) (See section 13-2 for further information on the disk drive)	B
131	Reserved	
132	Reserved	
140	Reserved	
141	Reserved	
142	Reserved	
150	A/C power fail 00 = Disable function 01 = Display information when fail is detected	B

PARA#	EXPLANATION	TYPE
	02 = Display information when fail is detected, and force to single mode	
151	Check amplifier fault 00 = Disable function 01 = Display information when fail is detected 02 = Display information when fail is detected, and force to single mode	B
152	Check D/A overrun or Halt 00 = Disable function 01 = Display information when fail is detected 02 = Display information when fail is detected, and force to single mode	B
160	Select + or - move for jog mode (X, Y and Z axes only) bits: 7 6 5 4 3 2 1 0 X Y Z * * * * * 0 - X "right" arrow is + Y "in" arrow in is + Z "up" arrow is + 1 - X "left" arrow is + Y "out" arrow is + Z "down" arrow is + An arrow key will cause an axis to move either + or -. This parameter determines the direction when an arrow key is pressed.	A
161	Set RS-232 Time-Out. Number ranges from 0 - 65,535 0 = Default setting of 5 minutes 1 - 65,535 = Time-Out in seconds	C
162	Reserved	

PAGE 2

PARA#	EXPLANATION	TYPE
200	Metric constant and resolution for the X axis - G71. To calculate this value: 100000 x (programming resolution/machine resolution)	D
201	English constant and resolution for the X axis - G70. To calculate this value: 100000 x (programming resolution/machine resolution)	D
202	Metric constant and resolution for the Y axis - G71. Use formula as in parameter 200	D
210	English constant and resolution for the Y axis - G70. Use formula as in parameter 201	D
211	Metric constant and resolution for the Z axis - G71. Use formula as in parameter 200	D
212	English constant and resolution for the Z axis - G70. Use formula as in parameter 201	D
220	Metric constant and resolution for the U axis - G71. Use formula as in parameter 200	D
221	English constant and resolution for the U axis - G70. Use formula as in parameter 201	D
222	Metric constant and resolution for the V axis - G71. Use formula as in parameter 200	D

PARA#	EXPLANATION	TYPE
230	English constant and resolution for the V axis - G70. Use formula as in parameter 201	D
231	Metric constant and resolution for the W axis - G71. Use formula as in parameter 200	D
232	English constant and resolution for the W axis - G70. Use formula as in parameter 201	D
240	Fastest feedrate for X, Y and Z axes in counts/minute	D
241	Fastest feedrate for U, V and W axes in counts/minute	D
242	Axes which exist in system (set bit 7 for X, bit 6 for Y, etc.): 7 6 5 4 3 2 1 0 X Y Z U V W * * 1 - Active 0 - Inactive * - Don't care	A
250	Default to acceleration (G8/H8)/ accel/decel (G9/H9). The G8/H8 or G9/H9 set in the parameter mode will override the parts program. 7 6 5 4 3 2 1 0 G8 G9 H8 H9 * * * * 1 - Active 0 - Inactive * - Don't care If the acceleration/accel-decel is set in the EEPROM, it effects the entire program. If it is entered in the parts program it effects only	A

PARA#	EXPLANATION	TYPE
	the block in which it is located. If feedrate threshold is exceeded, acceleration/accel-decel is automatically turned ON.	
251	For X, Y and Z axes—time constant for acceleration /deceleration function parameter = time constant (milliseconds)/32.768 (Time constant from 65 to 8356 gives parameter range from 2 to 255)	B
252	For U, V and W group—time constant for acceleration/deceleration function parameter = time constant (milliseconds)/32.768 (Time constant from 65 to 8356 gives parameter range from 2 to 255)	B
260	Skip limit 7 6 5 4 3 2 1 0 X Y Z U V W * * 1 = Check limit 0 = Don't check limit * = Don't care	A
261	Stack size, utilized in running parts program (reserved for RPT, CLS, DFS, DFLS, variables and entry points)	D
262	Parts program defaults 7 6 5 4 3 2 1 0 A * * * * B C * * = Don't care A : Tracking display default 0 = Display in machine steps	A

1 = Display in program steps
B : XYZ axes default
0 = Default to G70
1 = Default to G71
C : UVW axes default
0 = Default to H70
1 = Default to H71

PAGE 3

PARA#	EXPLANATION	TYPE
300	The axis that is set to move first in sequence of axes movement (for [back-end] mode) 7 6 5 4 3 2 1 0 X Y Z U V W * * 1 = Set to move 0 = Set not to move * = Don't care	A
301	The axis that is set to move second	A
302	The axis that is set to move third	A
310	The axis that is set to move fourth	A
311	The axis that is set to move fifth	A
312	The axis that is set to move sixth	A
320	Home direction 7 6 5 4 3 2 1 0 X Y Z U V W * * 1 = Positive	A

PARA#	EXPLANATION	TYPE
	0 = Negative * = Don't care	
321	Home feedrate for X axis (counts/minute from 229 to 8,000,000)	D
322	Home feedrate for Y axis	D
330	Home feedrate for Z axis	D
331	Home feedrate for U axis	D
332	Home feedrate for V axis	D
340	Home feedrate for W axis	D
341	Home offset for X axis (counts from 0 to 8,388,607)	D
342	Home offset for Y axis	D
350	Home offset for Z axis	D
351	Home offset for U axis	D
352	Home offset for V axis	D
360	Home offset for W axis	D
361	Reserved	
362	Reserved	

PAGE 4

PARA#	EXPLANATION	TYPE
400	M function: Milliseconds from data latch until strobe	C
401	M function: Milliseconds to debounce acknowledge/strobe length	C
402	S function: Milliseconds from data latch until strobe	C
410	S function: Milliseconds to debounce acknowledge/strobe length	C
411	T function: Milliseconds from data latch until strobe	C
412	T function: Milliseconds to debounce acknowledge/strobe length	B
420	Spindle type: * *1. ONCE-PER-REV :* No output on S function bus. Uses M functions to perform Spindle tasks.	B

2. *TWO'S COMPLEMENT:*

S COMMAND	OUTPUT SCALE	MSB LSB
S 32767	+ Full scale	0111 1111 1111 1111
S 32766	+ Full scale -LSB	0111 1111 1111 1110
S 00001	+ LSB	0000 0000 0000 0001
S 00000	Zero scale	0000 0000 0000 0000
S-00001	- LSB	1111 1111 1111 1111
S-32767	- Full scale +LSB	1000 0000 0000 0001

S-32768	- Full scale	1000 0000 0000 0000

3. UNIPOLAR BINARY:

S COMMAND	OUTPUT SCALE	MSB LSB
S 65535	+ Full scale	1111 1111 1111 1111
S 65534	+ Full scale -LSB	1111 1111 1111 1110
S 00000	Zero scale	0000 0000 0000 0000

4. COMPLEMENT BINARY:

S COMMAND	OUTPUT SCALE	MSB LSB
S 65535	+ Full scale	0000 0000 0000 0000
S 65534	+ Full scale -LSB	0000 0000 0000 0001
S 00000	Zero scale	1111 1111 1111 1111

5. OFFSET BINARY:

S COMMAND	OUTPUT SCALE	MSB LSB
S 32767	+ Full scale	1111 1111 1111 1111
S 32766	+ Full scale -LSB	1111 1111 1111 1110
S 00001	+ LSB	1000 0000 0000 0001
S 00000	Zero scale	1000 0000 0000 0000
S-00001	- LSB	0111 1111 1111 1111
S-32766	- Full scale +LSB	0000 0000 0000 0001
S-32767	- Full scale	0000 0000 0000 0000

6. BIPOLAR COMPLEMENT:

S COMMAND	OUTPUT SCALE	MSB LSB
S 32767	+ Full scale	0000 0000 0000 0000

S 32766	+ Full scale -LSB	0000 0000 0000 0001
S 00001	+ LSB	0111 1111 1111 1110
S 00000	Zero scale	0111 1111 1111 1111
S-00001	- LSB	1000 0000 0000 0000
S-32766	- Full scale +LSB	1111 1111 1111 1110
S-32767	- Full scale	1111 1111 1111 1111

7. UNIPOLAR BCD:

S COMMAND	OUTPUT SCALE	MSB LSB
S 09999	+ Full scale	1001 1001 1001 1001
S 09998	+ Full scale -LSB	1001 1001 1001 1000
S 00001	+ LSB	0000 0000 0000 0001
S 00000	Zero scale	0000 0000 0000 0000

8. BIPOLAR BCD:

S COMMAND	OUTPUT SCALE	MSB LSB
S 07999	+ Full scale	0111 1001 1001 1001
S 07998	+ Full scale -LSB	0111 1001 1001 1000
S 00001	+ LSB	0000 0000 0000 0001
S 00000	Zeró scale	0000 0000 0000 0000
S-00001	- LSB	1000 0000 0000 0001
S-07998	+ Full scale +LSB	1111 1001 1001 1000
S-07999	- Full scale	1111 1001 1001 1001

NOTE: For 12 bit resolution on all unipolar binary formats, use right most 12 of 16 bits.

For 12 bit resolution on all bipolar formats, use left most bit (MSB) as sign bit, and right most 11 of 16 bits for magnitude.

For 3 digit resolution with sign on bipolar BCD format, use left most bit (MSB) as sign bit, and right most 12 of 16 bits for 3 digits.

- * Manual Spindle Speed Override applies to all types of spindles.
- * Once-per-Rev spindle type uses M functions to perform spindle tasks. The other types listed use the S function bus. Select the spindle type which corresponds to your D/A pattern.

PARA#	EXPLANATION	TYPE
421	Spindle speed: Increment or decrement speed for jog or mdi's rpm-up or rpm-dn by this number. Applies to all typesofspindles (range is from 1 to 255)	B
422	Spindle high-gear range maximum rpms (for Once-per-Rev spindle types) Indicates the positive high limit for other types (range is from 0 to 65535)	C
430	Spindle high-gear range minimum rpms (for Once-per-Rev spindle types) Positive low limit for all other types	C
431	Spindle low-gear range maximum rpms (for Once-per-Rev spindle types) Does not apply to Unipolar and Complement Binary types Negative high limit for all other types	C
432	Spindle low-gear range minimum rpms (for Once-per-Rev spindle types) Does not apply to Unipolar and Complement Binary types Negative low limit for all other types	C

440 Spindle ratio: Pertains to Once-per-Rev types. Indicates teeth ratio: (low range gearing/high range gearing) x 1024 C

441 Dead band (1 - 10%). Indicates speed changer accuracy for the Once-per-Rev type B

442 For M0, M1, M2, M30 and M47 - functional control B

00 = Perform software function, output data to MST bus with Type A handshake as described in the following note.
01 = Completely skip the codeword, no output to MST bus
02 = Perform software function, output data to MST bus with Type B handshake
03 = Perform software function, output data to MST bus with Type C handshake
04 = Perform software function, no output to MST bus
05 = No software function performed. Output code to MST bus with Type A handshake

06 = No software function performed. Output code to MST bus with Type B handshake

07 = No software function performed. Output code to MST bus with Type C handshake

NOTE: For *handshake type*, see Note on next page.

450 Normal M functions - functional control B

00 = Perform software function (if exists), output data to MST bus with Type A handshake.
01 = Completely skip the codeword
02 = Perform software function (if exists), output data to MST bus with Type B handshake

	03 = Perform software function (if exists), output data to MST bus with Type C handshake	
	NOTE: All M-functions pertaining to once-per-rev spindle types (M3-M5, M40-M44) will always wait for an acknowledge signal as long as type code is not set to 1 (skip)	
451	Normal S functions - functional control (does not apply to once-per-rev spindle types) Same as parameter #450	B
452	For T functions 00 = Perform software function, ie., tool length and tool radius compensation. Output data to MST bus with Type A handshake T functions follow same format as parameter #442	
460	Reserved	
461	Reserved	
462	Reserved	

NOTE: MST output handshake may be one of three types, (explained in detail in the *Unidex 16 Hardware Manual*. This is unrelated to the "type" column of this chapter, which refers to type of data to be entered). Briefly:

TYPE A: Output code to bus, wait forever for acknowledge

TYPE B: Output code to bus, no acknowledge

TYPE C: Output code to bus, wait for timed acknowledge

PAGE 5

PARA#	EXPLANATION	TYPE
500	Reserved	
501	Reserved	
502	Reserved	
510	Feedrate threshold for automatic Acel/Decel ON (X,Y,Z axes) in counts per minute	D
511	Feedrate threshold for automatic Acel/Decel ON (U,V,W axes) in counts per minute	D
520	Program new name for X axis. New name *must* consist of two characters (upper case alphabetic), entered as 01-26 (A-Z respectively)	B
521	Program new name for Y axis. New name *must* consist of two characters (upper case alphabetic), entered as 01-26 (A-Z respectively)	B
522	Program new name for Z axis. New name *must* consist of two characters (upper case alphabetic), entered as 01-26 (A-Z respectively)	B
530	Program new name for U axis. New name *must* consist of two characters (upper case alphabetic), entered as 01-26 (A-Z respectively)	B

531 Program new name for V axis. New name *must* consist of two characters (upper case alphabetic), entered as 01-26 (A-Z respectively) B

532 Program new name for W axis. New name *must* consist of two characters (upper case alphabetic), entered as 01-26 (A-Z respectively) B

540 The bits of this parameter have an effect on various unrelated functions. A

Bit 0 - If the rightmost bit of parameter 540 is set to 1, trailing zeroes and the decimal point will be truncated during message output. For example, 7.2300000 will be output as 7.23, and 12.000000 will be output as 12 with no decimal point.

Bits 1 & 2 - When a MSG command causes variable data to be input through port-A and/or port-B with a command such as (MSG,<A,VAR1,VAR2>,...text...), the characters are echoed back to the host. If you do not want port-A or port-B to echo back incoming characters to the host after this data has been input, set bits 1 and 2 to 1. If bit 1 is set to 1, port-B will not echo incoming characters. If bit 2 is set to 1, port-A will not echo incoming characters.

Bit 3 - If new axes names are entered in parameters 520 - 532, they appear on the machine display screens. To have the new names apply to program entry as well, set bit 3 of parameter 540 to 1.

Bit 4 - Execution of a command file called "Autoexecute". If bit 4 is set to a 1, and the autoexecution file exists, it will run upon reset or power-up when <ENTER> is pressed.

541 Enables errors and other statuses to be output through the RS-232 ports or I/O channel to the host computer. Output may be ASCII text as seen on the Status Line of the CRT, or as ASCII "code nnn", where the code represents the condition being reported. (The error codes and their associated text are listed in the *Unidex 16 Hardware Manual*. A

00000001 - If bit 0 is set to a 1, send "code nnn" to port-B.

00000010 - If bit 1 is set to a 1, send "code nnn" to port-A.

00000100 - If bit 2 is set to a 1, the MSG command message will be sent to port-B

00001000 - If bit 3 is set to a 1, the MSG command message will be sent to port-A

1000nnnn - If bit 7 is set to a one, the hexadecimal representation of the error code will be sent to the I/O address $700. The status of the first 4 bits is irrelevant.

Also available through this parameter is a shorthand form of the Message Command (MSG) called the Command with Service Request (CMDS) command. *(See chapter 10 for details on these commands.)* The port is chosen by setting bits in this parameter. **Bit 4** is set to a 1 to choose port-B. **Bit 5** is set to a 1 to choose port-A. The CMDS command is different from the CMD or MSG command in that it waits for a Service Request character to be received before going to the next block. **Bit 6** determines if the input from the port will be ASCII (set to 0) or Binary (set to 1).

542 If the Command (CMDS) command is sent to a host computer through port-A and/or port-B, a Service Request is expected by Unidex 16 before proceeding to the next block. The Service Request character is specified by parameter 542, which is set to ones and zeroes to represent the desired character. A

550 Password protection for the edit mode D

1 - 99999999 - Designates the password that will be required to enter the edit mode

0 - No password will be required to enter the edit mode

551 Password protection for the file mode D

1 - 99999999 - Designates the password that will be required to enter the file mode

0 - No password will be required to enter the file mode

552 Password protection for the machine mode D

1 - 99999999 - Designates the password that will be required to enter the machine mode

0 - No password will be required to enter the machine mode

560 Password protection for the parameter mode D

1 - 99999999 - Designates the password that will be required to enter the parameter mode

0 - No password will be required to enter the parameter mode

561 Password protection for the test mode D

1 - 99999999 - Designates the password that will be required to enter the test mode

0 - No password will be required to enter the test mode

562 Password protection for the system (cannot run a command file without the designated password if this parameter is enabled) D

1 - 99999999 - Designates the password that will be required to execute a command file

0 - No password will be required to execute a command file

TYPE:

A - BINARY NUMBERS (0 or 1)
B - DECIMAL NUMBERS RANGING FROM 0 - 255
C - DECIMAL NUMBERS RANGING FROM 0 - 65,535
D - DECIMAL NUMBERS RANGING FROM 0 - 16,000,000

SECTION 6-3 INHIBIT/ENABLE KEY

When the Inhibit/Enable key is switched to inhibit, no information may be entered or stored in the parameter mode, although you may still read data from the CRT.

If you try to store data, the STATUS line shows:

***** STATUS: .EEPROM Inhibited, can't store data *****

Switch to the enable function in order to store parameter information.

NOTE: Remember to press <RESET> after changing parameter values to reinitialize Unidex 16.

CHAPTER 7: THE TEST MODE

When Unidex 16 is turned on it automatically does a self-test, testing the Battery, RAM, ROM and EEPROM.

If the CRT displays Checksum Error for RAM when Unidex 16 is turned on, enter the test mode. Press [.SV/RAM] to see which program is causing this error. The status line will display the program name and type. Example:

***** CHECKSUM ERROR: CIRCLE .PP *****

If you want to test Unidex 16 while running a parts program, enter the single mode and wait for all movement to stop before entering the test mode. Otherwise, Unidex 16 will finish the current block of program, set itself to the single mode and then do the test function.

SECTION 7-1 TESTS

Pressing the soft key [test] displays:

```
              .RAM   .SV/RAM   .ROM   .EEPROM   end
```

A. .RAM

When you press [.RAM] and <ENTER>, the status line will warn:

***** STATUS: Warning!! Destroy all /mms files?? *****

The soft key command area will display:

```
                 no                  yes
```

If you press [yes] the user memory will be checked and cleared. All programs stored in the Unidex 16 internal memory will be erased.

If you press [no] the user memory will not be checked and cleared. The soft key command area will revert to the first test mode display.

B. .SV/RAM

Pressing [.SV/RAM] yields:

```
                            end
```

If you press <ENTER> at this point, the user memory will be checked but the internal memory will not be erased (no programs will be lost).

As mentioned previously, if a Checksum Error was displayed on the CRT upon power-up, enter the test mode, press [.SV/RAM] and <ENTER>. The status line will let you know which program has a problem and requires editing. It will display the program name and type.

Pressing [end] will take you to the system level of softkeys again.

C. .ROM (.EPROM)

Pressing [.ROM] and <ENTER> displays:

```
                              end
```

A checksum test will be performed on the ROM. Any problems will be relayed to you via the STATUS line. If this occurs, call Aerotech, Customer Service Department (see chapter 14, Service and Repair for phone number and address).

[End] will take you back to the original screen of system softkeys.

D. .EEPROM

Pressing [.EEPROM] and <ENTER> tests the .EEPROM. Any problems will be relayed via the STATUS line. If this occurs, call Aerotech, Customer Service Department (see chapter 14, Service and Repair for phone number and address).

CHAPTER 8: THE PASSWORD MODE

The Unidex 16 system may be protected from unauthorized operation by the use of passwords. There are passwords available for each mode of operation. Each mode's password is set up in the parameter mode. The following list gives the parameter number for each mode.

MODE	PARAMETER
Edit	550
File	551
Machine	552
Parameter	560
Test	561
System	562

Any of these parameters may be left at a value of 0 if password protection is unnecessary. A valid password is any decimal number from 1 to 99999999. You may choose the same password or a different one for each mode. The "System" password will prevent the execution of a command file.

The soft key [password] comes up at the system level display of soft keys:

```
          edit  file  machine  parameter  test  password
```

If any mode is protected by a password, you must press [password] now to access that mode. You will see:

```
                    log-off    log-on    end
```

Press [log-on] and <ENTER>. The status line will read:

STATUS : ENTER PASSWORD :

Enter the password now. You will see an asterisk (*) each time a character is entered. When you are finished, press [end] and <ENTER>.

You may now enter the mode of operation as usual. When you are finished, re-enter the password mode and press [log-off]. *This clears the user password register so that no protected modes may be accessed. If you neglect to do so, an unauthorized person may access a normally protected mode.*

If you try to enter a protected mode without entering the correct password, the status line will display:

STATUS : current password not valid for this function

If you make a mistake when entering password data, press zero eight times (which enables you to start with fresh data), or press <ENTER> and start over again.

CHAPTER 10: PROGRAMMING LANGUAGE

The programming language that is built into all basic Unidex 16 systems is composed of three parts:

1. **A subset of RS-274-D code words**
2. **A subset of RS-447 commands**
3. **Aerotech developed extensions**

The syntax of the new programming language falls into two categories:

TYPE 1: Machine program data in RS-274-D-like format.

TYPE 2: Machine set-up, initialization or operation parameters in RS-447-like format.

Both types 1 and 2 conform to the Electronic Industries Association (EIA) standards and can be readily transferred to other NC machines with minor modifications.

In the following text, the term "codeword" will be used to indicate type 1 (RS-274-D) format.

The term "command" will be used to indicate type 2 (RS-447) format.

NOTE: When discussing codes and commands, the term "modal" is encountered. When a type 1 code or type 2 command is modal, it is set until another code or command from the same group is issued to reset it. If a second command does not reset it, it will stay set for the duration of the program.

Default commands are those that exist upon power up. They may be changed, but are in effect once the system is reset or powered up again.

SECTION 10-1 TYPE 1 CODEWORDS - RS-274-D

The basic type 1 codewords follow the RS-274-D format. In general, all Unidex 16 motions and outputs are programmed in this format.

The motion codewords are:

Primary - X Y Z I J K L C D G F
Secondary - U V W P Q R O A B H E

The primary codewords are those which apply to the X, Y, Z axes.

The secondary codewords are those which apply to the U, V, W axes. These additional three axes can be added to Unidex 16 as an option.

The output codewords are:

M S T

The program can begin with a (%) sign followed by a title line. Anything written between that symbol and the first <ENTER> is ignored by Unidex 16. Therefore, a program name or a comment may be inserted in this block.

The following subsections will define the various Type 1 codewords.

NOTE: The Type 1 (RS-274-D) format requires a space between each command (for instance G2 G17 I1.).

A. N CODEWORDS

N codewords are used as line numbers. They are not necessary in the programming or running of Unidex 16, but may be useful when editing a program. If needed, enter them as described in chapter 3, The Edit Mode.

If block delete slashes (/) are included in a program they may be preceded by line numbers, but must be entered before any other commands when used. For example, /N100 X1. or N100/ X1.

(Unidex 16 has a softkey command [renumber] which will automatically number your program blocks for you.)

B. G CODEWORDS

The G codewords are also known as Preparatory Functions since they set up the mode of motion control before an axis movement occurs.

A summary of the G codes used when programming Unidex 16 is as follows:

G0/H0 Point to point positioning at rapid traverse rate. The rapid traverse rate is determined by the parameter setting (see chapter 6). This command should only be given when the tool is clear of the workpiece and not in a position to do any cutting or drilling.

G1/H1 Linear interpolation produces a straight line in which the feedrate is taken as the vectorial velocity of all axes combined. All axes will start and stop simultaneously. If you want both groups of axes (X,Y,Z) and (U,V,W) to move together, you must use the SYNC command (see section 10-2).

A command block of :

```
G1 X4.0 Y3.0 F50.0
```

will produce a straight line from the initial position to the X+4, Y+3 coordinate position, at a feedrate of 50 inches (or millimeters) per minute.

G2/H2 Two dimensional arc, clockwise interpolation.

G3/H3 Two dimensional arc, counterclockwise interpolation.

NOTE: **BOTH AXES INVOLVED IN THE CIRCULAR CONTOUR MUST HAVE THE SAME RESOLUTION.**

To indicate which two axes are to be involved in the circular interpolation, enter a G17, G18 or G19 command. If you do not enter one of these, Unidex 16 uses the default codeword G17 (X and Y axes). For details see G17/H17, G18/H18 and G19/H19 codewords in this section.

For example, programming:

```
G91 G2 X6.0 Y0 I3.0 J0
G2 X-6.0 Y0 I-3.0 J0
```

will produce a circle with a radius of 3 inches (or millimeters). Since G17, G18 or G19 was not entered, Unidex 16 will default to G17, therefore cutting on the X,Y plane.

NOTE: G0/H0, G1/H1, G2/H2 and G3/H3 all belong to the same group of modal codewords, i.e., one will stay in effect until canceled and replaced by another from this same group. G1/H1 is the default code of this group.

G4 Dwell. This command should be followed immediately by an F codeword (Fnnn.n) which programs the dwell duration in seconds (ranging from 100mS to 999.9 seconds). Dwell programming code must occupy a block of its own. If the Fnnn.n entry is in decimal form, the dwell will be in seconds. If in integer form, it will be in tenths of seconds. The following example will produce a dwell of .5 second:

```
G4 F5
```

G8/H8 Turn on acceleration function. This codeword will cause a gradual acceleration before the desired speed is attained. Only the block which contains this word is effected. However, if the acceleration parameter in the EEPROM is set (see chapter 6) the function will be performed automatically for every block.

G9/H9 Turn on both accel/decel functions. This codeword will cause a gradual acceleration before the desired speed is attained and a gradual deceleration before the end of motion. Only the block which contains this word is effected. However, if the accel/decel parameter in the EEPROM is set, the function will be performed automatically for every block. When feedrate threshold is exceeded, accel/decel will automatically turn on. See section 6-2 for these parameter settings.

G17/H17 Combines X and Y axes (G17) or U and V axes (H17) as on the same plane for the execution of circular interpolation. The Z (or W) axis can make a linear interpolation at the same time. Helical interpolation (spiral) is done by doing the circular and linear moves simultaneously.

G18/H18 Combines Z and X axes (G18) or W and U axes (H18) as on the same plane.

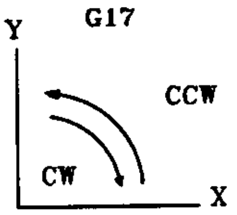
G19/H19 Combines Y and Z (G19) or V and W axes (H19) as on the same plane.

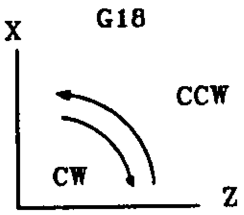
NOTE: The G17/H17, G18/H18 and G19/H19 codes determine which two axes will be paired together for circular interpolation. Each pair of axes can be programmed to make a circular move. Give a G2 or G3 codeword for circular interpolation (CW or CCW). Then give the G17, G18 or G19 to specify which two axes are to be involved in the move. The following diagram illustrates the direction of rotation on various planes.

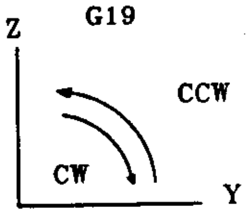
G17/H17, G18/H18 and G19/H19 form a modal group (one stays set until canceled by another).

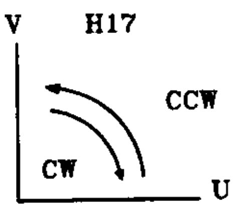
G17/H17 is the default mode upon power-up.

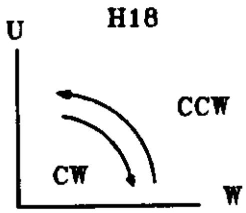
Both axes involved in a circular contour must have the *same resolution.*

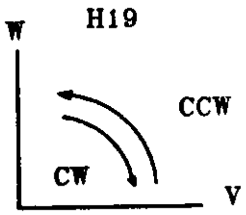












NOTE: The motion of the tool should be viewed from the positive (looking negative) perpendicular axis.

G23/H23 Corner-rounding mode

G24/H24 Cancel corner-rounding mode

NOTE: G23 is the command code that selects corner rounding whereby the motors get their next block of commands while they are still completing execution of the previous block. Normally, under G24, as the motors near completion of a command, they slow down appreciably. The corner rounding command, G23, keeps the motors running at a near constant velocity as they start to execute the next command, just before they have completed the previous one. Although there is a slight loss of accuracy using corner rounding, movements are performed more rapidly since the motors do not have to decelerate to zero speed before executing the next command. The result is a smooth, slightly rounded corner or "fillet" in the sections where successive cuts meet. This is especially useful for circular contouring. G24 is the codeword for normal positioning of the motors whereby they complete their previous commands and come to a stop before starting to execute commands in the next block. The greatest possible accuracy is achieved with normal positioning of the drives.

G23/H23 and G24/H24 form a modal group of codewords.
G24/H24 is the default mode at power-up.

G40 Cancel cutter compensation/offset. Turn off cutter radius compensation.

G41 Cutter compensation - left. Cutter is offset to the left side of the workpiece. Left is relative to the direction in which the cutter is moving.

G42 Cutter compensation - right. Cutter is offset to the right side of the workpiece. Right is relative to the direction in which the cutter is moving.

NOTE 1 G40, G41 and G42 form a modal group of codewords. G40 is the default mode at power-up.

NOTE 2: For more on ICRC, refer to section 10-3.

G70/H70 English programming. All of the dimensional and offset codewords represent measurement in inches. The decimal point is assumed to be 4 spaces to the left. Entering "X1" indicates a move of 0.0001 inch to Unidex 16. (Since the decimal point was not entered, it is assumed to be 4 places to the left.) Entering "X1.0" indicates a move of 1 inch to Unidex 16. (Since the decimal point was entered, its placement is not assumed by Unidex 16.)

The feedrate entries are limited to 5 digits (Fnnnn.n). Four of the digits are assumed to be before the decimal point and one after. The largest number you may enter as a feedrate in English programming is 9999.9.

In the following example, the left hand column contains programmed feedrates. The right hand column contains the feedrates as interpreted by Unidex 16.

PROGRAMMED FEEDRATE	ACTUAL FEEDRATE
F1	F0000.1
F1.0	F1
F12345	F1234.5

NOTE: Programming resolution for G70 is .0001.

G71/H71 Metric programming. All of the dimensional and offset codewords represent measurement in millimeters. The decimal point is assumed to be 3 spaces to the left. Entering "X1" indicates a move of 0.001 millimeter to Unidex 16. (Since the decimal point was not entered, it is assumed to be 3 spaces to the left.) Entering "X1.0" indicates a move of 1 millimeter to Unidex 16. (Since the decimal point was entered, its placement is not assumed by Unidex 16.)

The feedrate entries are limited to 5 digits (Fnnnnn). All five digits are assumed to be before the decimal point. The largest number you can program as the feedrate in metric programming is 99999. Following is a list of programmed vs. actual feedrates:

PROGRAMMED FEEDRATE	**ACTUAL FEEDRATE**
F1	F1.0
F1.0	F1.0
F12345	F12345
F1234.5	F1235 (rounds off the 5)

NOTE: Programming resolution for G71 is .001.

G70/H70 and G71/H71 form a modal group. The default mode is selected by a parameter setting, and depends on the type of ballscrew installed.

CANNED CYCLE COMMANDS

The following canned cycles are intended for machine tool applications, and work in the G17 (X/Y plane) mode only. For more on canned cycles, refer to chapter 11.

G76 Circular peck drilling canned cycle. Drills a series of holes in a circle. Radius of the circle, angle of the location of the first hole, number of holes and drilling depth are all user-programmable.

G77 Circular pocket milling canned cycle.

G78 Rectilinear pocket milling canned cycle.

NOTE: The G76, G77 and G78 canned cycles do not form a modal group. Each from this group cancel automatically after execution.

G80 Cancel canned cycle G81 to G85.

G81 Peck drilling canned cycle.

G82 Peck drilling with spindle stop canned cycle.

G83 Incremental peck drilling canned cycle.

G84 Peck drilling with spindle reverse canned cycle.

G85 Peck drilling with slow return canned cycle.

NOTE: G80 to G85 form a modal group of codewords. In addition, once any canned cycle is activated, G0, G1, G2 and G3 codewords will be disabled. G0, G1, G2 and G3 will not resume functions until after G80 cancels the canned cycle.

G80 is the default mode at power-up.

G98 Return to initial point during canned cycle G81 to G85.

G99 Return to temporary return (R) point during canned cycle.

NOTE: G98 and G99 are modal codewords. They are associated only with canned cycles G81 to G85.

G98 is the default mode at power-up.

G90/H90 Absolute programming. All data input is in the form of absolute dimensions. All dimensions are with respect to a floating zero, which is defined by the G92/H92 codeword or the default X0, Y0, Z0 position, which is machine "home".

In the absolute mode (G90), an axis moves to the locations indicated by the axis commands. For example, if you wanted to move the tool along the X axis to the +3.0000" position and it was presently located at the -3.0000" position, just program X3.000. The absolute position registers will keep track of the motor position.

G91/H91 Incremental programming. All data input is in the form of incremental dimensions. In the incremental mode (G91), the axes move the distance indicated, relative to the present position. For example, if you wish to move the tool along the X axis from the -3.000" position to the +3.0000" position, you would have to program X6.0000.

G90/H90 and G91/H91 belong to the same group of modal codewords. The default mode at power-up is G91/H91.

G92/H92 Preload axis relative coordinate registers to desired values. The G92 (for X,Y,Z axes) or H92 (for U,V,W axes) codeword should be followed immediately by the axis and value to be loaded. These codewords should occupy a block of their own. An example would be:

```
G92 X0 Y0 Z0
```

This will define the current XYZ relative coordinate to be 0, 0, 0. Therefore, a floating coordinate (floating home) is established in this example. All axes moves will now be in reference to this position.

C. X, Y, Z AND U, V, W CODEWORDS

These are dimensional codewords. They describe the axis to be moved and the distance (or location) desired. When absolute programming is used (G90/H90), these words program the location to which axes are to be moved. When incremental programming is used (G91/H91), these words program the distance by which the axes will be incremented with the move.

The highest number that may be programmed for an axes value is + 8388607 and may be specified in inches (G70) or millimeters (G71). (Refer to the G codes G70/H70 and G71/H71 of this section.)

1. You may also program an axis to a name other than X, Y, Z, U, V or W. The new name *MUST* consist of two characters, both being any upper case alphabetic character (A - Z).

Axis names are X, Y, Z, U, V and W by default. The user may enter new names in the EEPROM parameters #520 - #532. These must be entered as decimal numbers. (The upper case letters A - Z correspond to decimal numbers

01 - 26.) Any other number is invalid and will blank out that position on the display. (Which can be used to clear the display if the axis does not exist in the system.) After reset, the new axis names will be valid and will be displayed in the machine mode. To make the new axis name apply to program entry as well, bit 3 (from right) of parameter #540 must be set to a 1.

All program commands may use the new axis names in place of X, Y, Z, U, V and W. (However, a program may also use X, Y, Z, U, V and W, regardless of what the appropriate axis has been named. It is now the programmer's choice.) An example of changing an axis name is as follows:

Change label of X axis to "AB" by entering 00102 into parameter #520, and then press [store].

The left-most zero is ignored. The "A" is represented by 01, and the "B" is represented by 02.

(Set bit 3 of parameter #540 to a 1 to make the new label apply to program entry as well.)

D. I, J, K AND P, Q, R CODEWORDS

These are interpolation parameters parallel to the X,Y,Z (or U,V,W) axes respectively. The value may be positive or negative. Only two axes from the X,Y,Z group or two axes from the U,V,W group may be programmed in a circular interpolation at one time using G17/H17, G18/H18 or G19/H19. The third axis may make a linear move at the same time.

In circular interpolation, I,J,K,P,Q or R program the center of the arc (radius). These codes program the offset vector, which is the signed incremental distance from the beginning of the arc to the arc center, whether Unidex 16 is in the absolute (G90/H90) or incremental (G91/H91) mode (see figure 10-1). Figure 10-1 may be programmed as:

G2 X2.0 Y-1.0 I1.0 J-.1

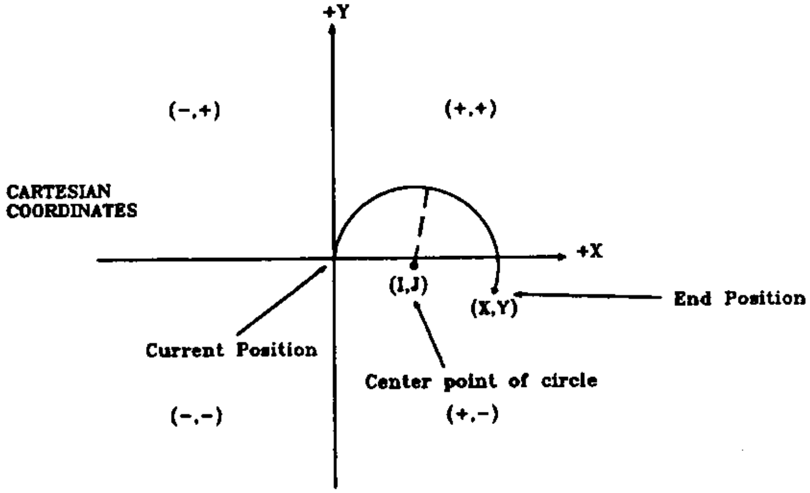


FIGURE 10-1: CIRCULAR INTERPOLATION

The offsets for each axis are:

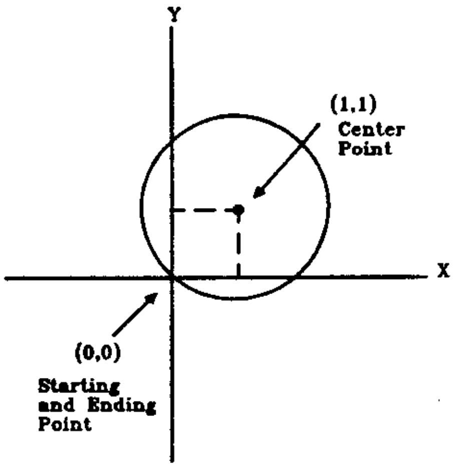
AXIS	OFFSET
X	I
Y	J
Z	K
U	P
V	Q
W	R

Unidex 16 can be programmed to do a complete circle in one command block. Only direction, center point and axes plane must be specified. For example:

G2 G17 I1. J1.

The above codewords specify a CW circle (G2) on the XY plane (G17). The ending point is assumed to be 0 (the same as the starting point). The center point is established by the offsets of I and J. In the sample command above, I1., J1. specifies the center point as the incremental distance of (1., 1.) from the starting point (0,0).

The following contour would result from the above command:



An example of a CCW circular contour on the ZX plane would be:

G3 G18 K2.5 I-2.5

An example of a CCW circular contour on the VW plane would be:

H3 H19 Q-2. R-2.

E. F/E CODEWORDS

F/E defines the feedrate in inches (G70) or millimeters (G71) per minute. F/E programs the vectorial feedrate of a move. This applies to linear, circular and helical moves.

F is for the X, Y and Z axes and E is for the U, V and W axes.

In English programming (G70), feedrate entries are limited to 5 digits. The largest number that can be programmed as a feedrate in English programming is 9999.9. The entries are seen by Unidex 16 as Fnnnn.n, with four digits to the left of the decimal point and one to the right. In the following example, the left hand column contains programmed entries. The right hand column contains what Unidex 16 interprets those feedrates to be.

PROGRAMMED FEEDRATE	ACTUAL FEEDRATE
F1	F.1 (sees entry as 00001)
F12345	F1234.5
F1.0	F1.0
F50	F5.0 (sees entry as 00050)

In metric programming (G71), feedrate entries are also limited to 5 digits. The largest number that can be programmed as a feedrate in metric programming is 99999. The entries are seen by Unidex 16 as Fnnnnn, with all five digits to the left of the decimal point (none to the right). In the following example, the left hand column contains programmed entries. The right hand column contains what Unidex 16 interprets those feedrates to be.

PROGRAMMED FEEDRATE	ACTUAL FEEDRATE
F1	F1 (sees entry as 00001)
F12345	F12345
F1234.5	F1235 (rounds off the 5)
F1.0	F1
F50	F50 (sees entry as 00050)

Whenever programmed feedrate exceeds 110% of the achievable rate, controls stop and error information is displayed.

F. L, C, D AND O, A, B CODEWORDS

For the X,Y,Z axes, to define the starting point of the circle, define L as the radius and define C as the angle of the starting point of the circle, measured from the X-positive (G17), Z- positive (G18) or Y-positive (G19) direction. Define D as the angle of the ending point of the circle, measured from the same direction. Both the C and D codewords are in degrees. If the minutes and seconds of a degree are required, enter them as fractions. Example:

112+2/60+16/360

The angle can also be entered as a decimal. For example:

112.2435

For the U,V,W axes, to define the starting point of the circle, define O as the radius and define A as the angle of the starting point of the circle, measured from the U-positive (H17), W- positive (H18) or V-positive (H19) direction. Define B as the angle of the ending point of the circle, measured from the same direction. Both A and B codewords are in degrees.

See figure 10-2 for an example of polar coordinate programming.

G2 L1.0 C140 D45 F50.0

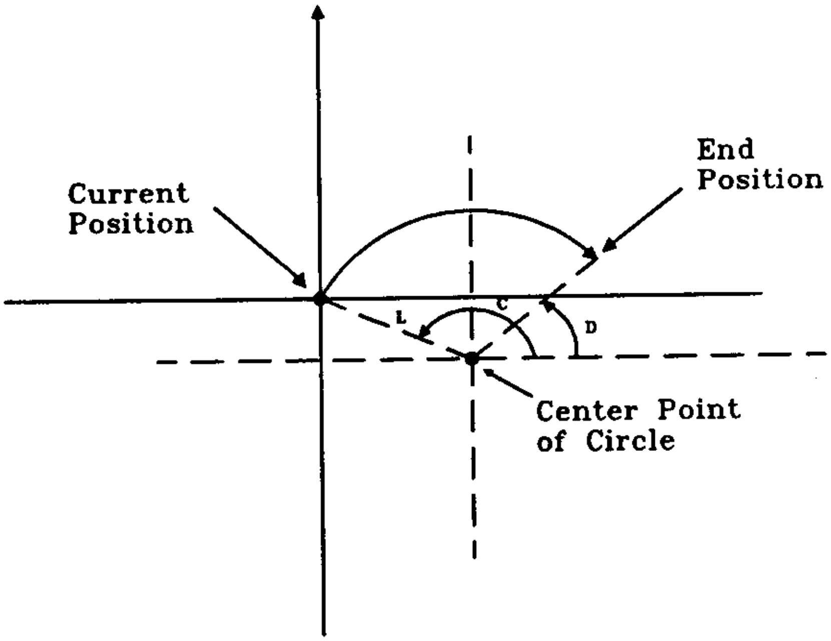


FIGURE 10-2: POLAR COORDINATE PROGRAMMING

In figure 10-2, an arc of 95 degrees width is shown. L1.0 specifies the radius, C140 the angle of the starting point and D45 the angle of the ending point.

G. S CODEWORDS

The spindle speed function programs the spindle speed in RPMs for once-per-rev spindle types. For details on how to program the other 7 spindle types, and the maximum and minimum speeds allowed, refer to chapter 6, section 6-2, the 400 parameter group.

S codewords, when executed, produce output on the M or S output bus, depending on the spindle type.

For different spindle types and the parameters that apply to each type, refer to chapter 6, The Parameter Mode, (parameter 400 group).

H. T CODEWORDS

The tool table is saved in a separate file. If the Aerotech Tool Changer is used, then the following apply:

T00	Removes tool offset.
T01 to T25	Insert tool with corresponding number. No offset.
T01nn to T25nn	Insert tool with corresponding number. The nn specifies offset number in the tool file. For example, T0113 would be tool #1 with offset #13 in tool file.
T26	Initiate tool changer operation.
T27	Return any tool.
T28	Reset carousel positions.
T31	Reconfigure carousel.
T41 to T65	Carousel rotates to selected tool (40 plus the tool number).
T99	Set current tool offset and diameter to 0.
T99nn	Set current tool offset and diameter to the one specified in offset nn (Onn) in the tool file.

I. M CODEWORDS

M codes are also called M function outputs. When executed by Unidex 16, the number following the letter M will be output onto the M,S,T function bus. Data is output as a 2 digit code, and the M-strobe signal will go active after the data appears. Then Unidex 16 will wait for and debounce an outside acknowledge signal. If this handshake signal comes back within a specific time, Unidex 16 will proceed and execute the following program. All signals are TTL level and all these signals are on the J4 connector of the CRT board. Parameters exist for the strobe length and length of time Unidex 16 will debounce the signal. Other parameters exist to select whether the function can be partially or totally selected (refer to chapter 6 for details).

There are special M codes that carry special functions. They are:

M0

M1

M2

M30

M47

How these special M codes are handled is decided in parameter #442. It can be set to perform its software function and output data to the bus, be ignored completely, perform its software function and output no data, or perform no software function but output data to the bus with or without a handshake. See chapter 6, parameter #442 for details.

M0 Program stop. After completion of all other words in the block, M0 will be output if enabled by parameter. Press <CYCLE START> to continue the program.

M1 Optional (planned) stop. When M01 is output, program execution will terminate, if the Optional Stop switch on the front panel is on. Unidex 16 resumes execution once the switch is turned off and <CYCLE START> is pressed.

M2 End of program. After completion of all other commands, M02 will be output. <CYCLE START> will restart the program.

M30 End of data. Execution will stop when M30 is encountered. <CYCLE START> will restart the program.

M47 Return to program start. When M47 is output, program execution will continue from the start of the program.

J. BLOCK DELETE

The block delete switch on the front panel gives you the option, when running a parts program, of skipping certain blocks of program.

When editing the parts program, precede these optional blocks with a slash (/). When running this program in the machine mode, if the Block Delete Switch is on, these blocks will be omitted. If the switch is off, they will not be omitted. Example:

```
G1 X2. Y3.
/G1 X1. Y1.
```

In the above example, the second block will be skipped if the Block Delete Switch is on.

K. USER COMMENTS

Comments may be included after program blocks to explain or clarify commands. This greatly improves the documentation of a program.

Place a semicolon (;) after the command block and then enter your comments. Any alphanumeric character or special symbol, except a percent (%) sign , can be used. Any entry listed after the semicolon is ignored by Unidex 16 when running the program.

SECTION 10-2 TYPE 2 COMMANDS - RS-447 FORMAT

Since both type 1 and type 2 codes are included within the same program, parentheses are used to differentiate between the two. The "(" is used to signify "control in", and the ")" is used to signify "control out". Enclosing your type 2 commands in parentheses enables Unidex 16 to distinguish between type 1 codes and type 2 commands within the parts program.

NOTE 1: Any RS-447 (type 2) command must occupy its own block within a parts program.

NOTE 2: In type 2 commands, the command and its options are separated by commas, not spaces.

The type 2 commands are:

A. Go home, REF or HOME
B. Mirror image, MIR
C. Jump commands, JUMP
D. Define entry point, DENT
E. Repeat loop, RPT

F. Call subroutine, CLS
G. Define subroutine, DFS
H. Define Library subroutine, DFLS
I. Abort subroutine, ABTS
J. Scaling factor, SCF
K. Turn on or off scaling, SCO
L. Display message on CRT, MSG
M. Message to Interfacing Equipment, CMD & CMDS
N. Display Page, DISP
O. Write to Datafile, DATA
P. Six axes synchronized contouring, SYNC
Q. Programmable accel/decel time constant, ADTC
R. Interrupt, INT or CIRQ
S. UMFO
T. UMSO
U. Safe Zone, ZONE
V. Cutter compensation, CCP

For information on all advanced functions, refer to chapter 12, *VARIABLES AND MATH PACKAGE.*

A. GO HOME, REF or HOME

Two commands send any or all axes to the home position, REF and HOME. Example:

(REF,X,Z)

This command example would send X and Z axes home simultaneously at a feedrate set in the EEPROM (parameter mode). This feedrate can be adjusted by using the Manual Feedrate Override (MFO).

The axis will first move to the end of travel and then move out until the marker is found. Then it will offset a distance (Home Offset – if set as explained in chapter 6, The Parameter Mode) to reach the home position. At this point, the coordinate registers will be cleared to zero.

The HOME command is identical to REF. For example:

```
(HOME,X,Y,Z,U,V,W)
```

B. MIRROR IMAGE, MIR

Mirror Image is available on Unidex 16 for all six axes. To program Mirror Image, enter X1, Y1, Z1, U1, V1 or Z1 after MIR. Example:

```
(MIR,X1,Y1)
```

This will turn on the mirror image for both the X and Y axes. Any axis not specified by this command will not be mirrored.

To turn MIR off, program the axis being mirrored followed by a zero. Example:

```
(MIR,Y0)
```

This will turn off the Mirror Image for the Y axis. All other axes are not effected. MIR is a modal command.

Mirror, as well as the scaling function and factor (discussed later), operate in both the incremental as well as absolute mode. However, when in the *absolute* mode, it is the user's responsibility to remember that these functions are always in reference to the *software home* (established by G92).

Also, when enabling an Interrupt (discussed later in this chapter) or a User-programmable Softkey (see User-Programmable Softkey in chapter 13, Unidex 16 Options), the program may (depending on interrupt type) skip the rest of the program block, remember *current machine position* and go to a subroutine or entry point. The user must re-establish the software home before mirror or scaling is reenabled in such cases, because when Unidex 16 returns from the subroutine or entry point, it remembers the *current machine position*. Therefore, the mirror and scaling are no longer in reference to the prior software home.

C. JUMP COMMAND, JUMP

The jump command (JUMP,nnnn) or (JUMP,nnnn,condition) will cause program execution to jump to and continue from a specific line (DENT,nnnn).

The place to which to jump is defined by a Define Entry command (DENT). The Jump and Define Entry commands must match. Example:

```
JUMP,AA11 (Jump to entry point AA11)
DENT,AA11 (Defined as entry point AA11)
```

The entry point name may be from 2 to 4 characters in length. The first two characters following JUMP are letters A-Z. The second two are alphanumeric, A-Z or 0-9.

The (JUMP,nnnn) command should occupy its own block of program.

An example of the (JUMP,nnnn,condition) command is:

```
(JUMP,AA12,$XAP.GT.1000)
```

This causes program execution to jump to entry point AA12 when the value of the current X axis absolute coordinate is greater than 1000.

The defined entry point for a jump can be replaced by a variable. Instead of programming (JUMP,nnnn), (JUMP,#VARn) can be entered. The entry point can then be defined by the variable. Example:

```
VAR1 = ENT1
(JUMP,#VAR1)
(DENT,ENT1)
```

This is the same as:

```
(JUMP,ENT1)
(DENT,ENT1)
```

NOTE: DO NOT DUPLICATE NAMES WHEN DEFINING JUMPS AND SUBROUTINES.

The # sign before the variable lets Unidex 16 know that the variable defines the entry point and is not simply the name of the entry point.

D. DEFINE ENTRY POINT, DENT

This command defines an entry point (DENT,nnnn) for a jump command. The entry point name may be from 2 to 4 characters in length. In (DENT,nnnn) the first two characters are letters A-Z. The second two characters are alphanumeric, A-Z or 0-9. An example of a define entry command would be (DENT,AA11).

This string should be a unique identification that is not used anywhere else in the program.

The (DENT,nnnn) command should occupy its own block of program.

E. REPEAT LOOP, RPT

The following example shows a repeat loop:

```
(RPT,8
G1 X1000 Y1000 F100
Z-1500
Z1500
X-1000 Y-1000
)
```

The loop is placed between parentheses. It will be repeated 8 times, which is defined in the first line. The first line should contain only RPT and the repeat count number; no other code should be used. The last line should have the ending parenthesis only.

Repeat loops can also be nested together. Example:

```
(RPT,10
X1000
(RPT,10
Z-1000
Z1000
)
)
```

A repeat function needs 12 bytes of stack, which is taken from the user-programmable memory. Every time you use this function it requires 12 bytes of memory. It will be put back when the function is completed. The above example of one RPT function nested inside another would use 24 bytes of stack.

The number of repeats may be programmed as a variable. See section 10-7 for details on this and other advanced programming.

F. CALL SUBROUTINE, CLS

Rather than program a sequence of commands several times, define the sequence as a subroutine. Each time this sequence is required within the program, call the subroutine with a (CLS,nnnn) command. Example:

(CLS,AB12)

The name of the subroutine is indicated by the nnnn label. The subroutine name may be from 2 to 4 characters in length. It should be defined uniquely by a Define Subroutine or Define Library Subroutine command. The first two characters are letters from A-Z. The second two are alphanumeric characters.

A subroutine may be called with parameters. An example would be (CLS,nnnn,PARA1,PARA2). Following the name nnnn in this example are two parameters. They can be actual values or variables. The real value or the value located at the variable's address is passed into the subroutine. These variables must be defined by DVAR (define variable) or DSGV (define system global variable) if the subroutine is being called from the main program. If one subroutine is calling another subroutine, these variables must be defined by the calling subroutines's DFS command or by DSGV. If the "GLOB" option is used, the variable may be defined by DVAR also. (See chapter 12 for details on variables.) An example of a Call Subroutine with parameters is:

(CLS,AA23,1245,VAR1)

When calling a subroutine (CLS,nnnn), a variable can be substituted for the name of the subroutine. Example:

```
VAR1=ENT1
(CLS,#VAR1)
(DFS,ENT1
.
.
.
)
```

This is the same as:

```
(CLS,ENT1)
(DFS,ENT1
.
.
.
)
```

In the above example, #VAR1 tells Unidex 16 that the subroutine is defined by a variable. This method of defining subroutines allows for more flexibility of programming, since the subroutine being called can vary with the value of VAR1.

For details on advanced programming with the use of variables, see chapter 12.

G. DEFINE SUBROUTINE, DFS

Every subroutine must be explicitly defined by a (DFS,nnnn command. Example:

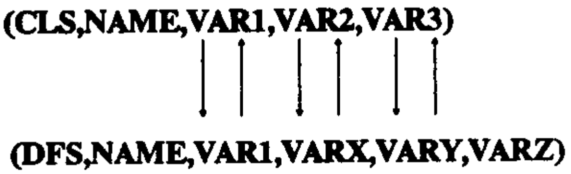
```
(DFS,AB12
```

The (DFS) name may be from 2 to 4 characters in length. The first two characters of nnnn are letters from A-Z. The second two are alphanumeric characters. This command should occupy a block of its own. Example:

```
(DFS,TAP1
S500 M3
Z-1000 F200
M5
G4 G0.2
M4
Z1000
)
```

The DFS command may contain variables, but not actual values. Variables listed after DFS are variables which are unique to the subroutine. (See chapter 12 for an explanation of Global and Local variables with subroutines.)

The variables listed after a DFS command can assume the same data as variables in the calling routine (the main program or another subroutine), or can be unique to the subroutine. For example:



```
(CLS,NAME,VAR1,VAR2,VAR3)

(DFS,NAME,VAR1,VARX,VARY,VARZ)
```

In the above example, the data of variables VAR1, VAR2 and VAR3 are passed to variables VAR1, VARX and VARY. Variable VARZ will equal zero.

CAUTION: Variables named by DFS are distinct from those named by CLS, even if the names are the same. See chapter 12.

Subroutines may be nested together also. Each DFS requires 12 bytes of stack.

Any named variables included in the program require 12 bytes of stack also. DSGV variables require only 8.

H. DEFINE LIBRARY SUBROUTINE, DFLS

A library subroutine is the same as a normal subroutine, except a library subroutine restores machine states. Therefore, a program which calls a Library Subroutine will not have its modal information altered after the subroutine is executed. Example:

```
(DFLS,nnnn
```

The (DFLS) name may be from 2 to 4 characters in length. The first two characters of nnnn are letters from A-Z. The second two are alphanumeric characters. These characters cannot be defined anywhere else in the program. This command should occupy its own block of program.

The modal codes and commands which will be reinstated after a library subroutine are:

1. All modal G codes
2. Feedrate (F and E codes)
3. Scaling function and factor
4. Spindle speed

A DFLS requires 64 bytes of user-programmable memory.

I. ABORT SUBROUTINE, ABTS

Normally after a subroutine is executed, Unidex 16 returns to the main program. The ABTS command allows the current level of subroutine to be aborted and execution to continue from a place programmed by the user. Example:

```
(ABTS,nnnn)
```

The nnnn in the above example is an entry point defined by a (DENT) command. This command must be located inside the subroutine (which is enclosed in parentheses). Example:

```
(DFS,TEST
.
.
.
(JUMP,ENT1,VAR1.EQ.2)
/(ABTS,ENT2)
(DENT,ENT1)
.
.
.
)
(DENT,ENT2)
```

The above example controls jumps to the entry point "ENT2" at the end of the subroutine "TEST" rather than going back to the main program (if the Block Delete Switch is turned off).

ABTS is not a modal command.

An abort subroutine command can refer to a variable rather than an entry point. Example:

```
VAR1 = ENT2
(DFS,TEST
.
.
.
(ABTS,#VAR1)
.
.
.
)
(DENT,ENT2)
```

J. SCALING FACTOR, SCF

Unidex 16 allows an individual axis to be scaled up or down by a scaling factor. Example:

```
(SCF.X0.33333,Y1.5)
```

The above example defines the scaling ratio on the X axis to be 0.33333. The scaling ratio must be a positive number, ranging in value from .00001 to 99.99999. No error will accumulate from move to move when the scaling factor is turned on.

This command is modal. The default value is a ratio of 1.

NOTE: Scaling does not effect inch to metric conversion. Unidex 16 executes the scaling operation first and then takes care of the inch to metric conversion.

K. TURN ON/OFF SCALING, SCO

To turn on the scaling, program:

```
(SCO,1)
```

To turn off the scaling, program:

```
(SCO,0)
```

The scaling function is a modal command. As with the mirror function mentioned previously, scaling operates in both the incremental and absolute mode. However, when in the absolute mode, it is the user's responsibility to remember that this function is always in reference to the *software home* (established by G92/H92).

When enabling an Interrupt (discussed later in this chapter) or a User-programmable Softkey (see User-Programmable Softkey in chapter 13), the program may (depending on interrupt type) skip the rest of the program block, remember *current machine position* and go to a subroutine or entry point. The user must re-establish the software home before mirror or scaling is reenabled in such cases because when Unidex 16 returns from the subroutine or entry point, it remembers the current machine position and the scaling is no longer in reference to the prior software home.

L. DISPLAY MESSAGE, MSG

Messages can be entered into the program and be displayed at the time of execution. The MSG function can display text or the value of user's or system variables on the CRT, port-A or port-B, or any combination of these.

The format for entering messages into your program is:

```
(MSG,<Pnnnn,variable>,...text...#variable...text...)
```

The message cannot contain parentheses (except for the opening and closing parentheses). Use "<" and ">" instead.

The <**Pnnnn**> shown above is there for the entry of MSG options. "P" stands for port and can be A, B, AB or nothing. If nothing is entered, the message goes to the CRT only.

The "**nnnn**" portion stands for duration of message in seconds and may be a BCD number from 0 - 9999.

The <**Pnnnn,variable**> is an input or option function. At the time of the program run, the value of this variable may be input by the user.

The "#variable", as part of the message text, means "display the value of the variable" and is an output or display variable. At the time of the program run, the values of these variables will be output.

When editing the program, the message line is two characters longer than the line displayed on the CRT within the program. Therefore you must indent each line two spaces to get the message to line up on the screen.

OUTPUT OR DISPLAY

Following are examples of how to program a message:

1. (MSG,...text...) - MSG shown on CRT only. Remains there until <CYCLE START> is pressed.

2. (MSG,<10>,...text...) - MSG shown on CRT only. Will remain for 10 seconds (or less if <CYCLE START> is pressed before the time has elapsed).

3. (MSG,<0>,...text...) - Skip message function.

4. (MSG,<A>,...text...) - Same as #1, except also output to port-A.

5. (MSG,<B>,...text...) - Same as #1, except also output to port-B.

6. (MSG,<AB>,...text...) - Same as #1, except also output to ports A and B.

7. (MSG,<A2>,...text...) - Same as #2, except output data to CRT and port-A. Will remain for 2 seconds (or less if <CYCLE START> is pressed before the time has elapsed).

8. (MSG,<B2>,...text...) - Same as #2, except output data to CRT and port-B. Will remain for 2 seconds (or less if <CYCLE START> is pressed before the time has elapsed).

9. (MSG,<AB2>,...text...) - Same as #2, except output data to CRT, port-A and port-B. Will remain for 2 seconds (or less if <CYCLE START> is pressed before the time has elapsed).

10. (MSG,<A0>,...text...) - No message to CRT, only output data to port-A and then continue program.

11. (MSG,<B0>,...text...) - No message to CRT, only output data to port-B and then continue program.

12. (MSG,<AB0>,...text...) - No message to CRT, only output data to port-A and port-B and then continue program.

A "C" or "D" option may be used with "A" and/or "B". "C" would cause the message to be displayed on the screen, but the motion of the axes would continue, *if the display time is set to zero*. Feedhold is allowed without loss of the message screen, and <NEXT PAGE> may be used to view the other screens and return to the message page while motion continues. Any other functioning key will clear the message.

"D" will produce the same function, but the beeper will be disabled. This may be used in a program loop with a continuously updating message, thereby eliminating a continuous beep.

For example:

<MSG,<ABD0>,....text....)

For display (or output) variables, the following variables are valid:

1. #123 - Display 123
2. #VAR1 - Display the value of variable VAR1 in real constant
3. #H:VAR1 - Display the value of variable VAR1 in hexadecimal number
4. #C:VAR1 - Directly display byte data, so if VAR1 = "ABCD", ABCD will be displayed.
5. #C:VAR1,VAR2,VAR3 - Chains messages together. If VAR1 = "GOOD", VAR2 = " MOR" and VAR3 = "NING", GOOD MORNING will be displayed.
6. #12+SIN<DEG<ATN<VAR1/VAR2 - Display the result in a real constant

## - Display #
##VAR1 - Display #VAR1

If a message to be sent to port A or port B must contain characters not available on the front panel of Unidex 16, for instance, ASCII Control Key <CR> <LF>, the ASCII code (see appendix A) equivalent to the required characters can be sent within the message.

If our example of carriage return/line feed, <CR> <LF>, is to be sent to a port, you would send:

(MSG,<A>,....text...#C:H,0D0A)

Referring to appendix A, you will see the ASCII code hexadecimal equivalent for CR is 0D and for LF is 0A.

When characters are sent to a port as shown above, they are sent as 4 bytes of data. Therefore, #C:H,0D0A would be sent as:

byte 1 - 00H
byte 2 - 00H
byte 3 - 0DH
byte 4 - 0AH

To insure the 4 byte format, the first two bytes are sent as zeros, the second two contain the <CR> <LF> characters.

INPUT OR OPTION VARIABLES

1. (MSG,<A,VAR1,VAR2,VAR3,VAR4>,................text.......)
Sends message (text) to port-A and CRT. Required data must then be input from the CRT or port-A. The message will remain until an <ENTER> or a carriage-return <CR> is received.

2. (MSG,<B,VAR1,VAR2,VAR3,VAR4>.................text......)
Sends message to port-B and CRT. Required data must then be input from CRT or port-B. Will remain until an <ENTER> or <CR> is received.

3. (MSG,<AB0,VAR1,VAR2,VAR3,VAR4>,............text.....)
Sends message to port-A and port-B and receives input data from port-A or port-B. Because the 0 follows AB, the CRT does not receive the message or send back input data.

4. (MSG,<100,VAR1,VAR2,VAR3,VAR4>...............text.....)
Will send message to and receive input data from the CRT. The time entered (100 here) is meaningless since an <ENTER> is needed to eliminate the message and continue the program. In fact, no time need be entered at all. Entering (MSG,<,VAR1,VAR2,VAR3,VAR4>,...) will produce the same result.

The sample message:

(MSG,<A,VAR1,VAR2,VAR3,VAR4>,.........text.....)

will show a message on the CRT and port-A. Data must be input from the CRT or port-A, terminated by an <ENTER> or <CR>. If input data is:

12,H,10FA,"Unidex16" <ENTER>

Then:

VAR1=12, as a real number
VAR2=H,10FA, as a hexadecimal number

```
VAR3="Unid"
VAR4="ex16"
```

For a longer character string, more VAR commands would have been necessary, since a variable can only hold 4 characters.

NOTE: In order to handle the variable input function, Unidex 16 must be in the machine mode.

NOTE: For a real constant which is greater than 99999999, Unidex 16 will display 1.000000E8.

For a real constant which is less than 0.0000001, Unidex 16 will display 1.0E-8.

M. MESSAGES TO EXTERNAL HARDWARE, CMD/CMDS

Both the CMD and CMDS commands are variations of the MSG command, discussed in the previous subsection. The two commands are intended for use with external hardware which requires handshaking. There is no provision in these commands for port selection or dwell time. The port is selected by setting bits in the EEPROM (chapter 6). In parameter #541, bit 4 from the right is set to a 1 for port B, bit 5 is set to a 1 for port A. The message text follows the same rules as a MSG command. A carriage return and line feed are automatically sent after the text to terminate transmission. If variables are to be input, they are to be listed in angle brackets before the text (similar to MSG command).

CMDS is different because it will wait to receive a Service Request character before going to the next block. This Service Request character is set up in parameter #542, which is set to ones and zeros to represent the desired ASCII character. For example:

```
(CMD,This is a test)
```

The text "This is a test" will be output to the port(s) programmed in the EEPROM. A carriage return and line feed is sent after the text.

But for the CMDS command:

(CMDS,This is a test)

the command is the same as above, but a service request must be received before moving to the next block.

The 6th bit in parameter #541 is set to a 0 if input from the port is to be ASCII data, and a 1 if input is to be binary data. For example:

Another way to use the CMD command is to store the data in a variable:

(CMD,<VAR1>,This is a test)

The data that is sent back will be stored in a variable (VAR1), similar to a MSG input.

NOTE: Using the Unidex 16 to control one or more Unidex 1 single-axis controllers is one example of an application of these commands.

N. DISPLAY PAGE FOR MACHINE MODE, DISP

A machine mode page is available for customer defined displays. This is page eight of the machine mode displays and is activated by the DISP command in the parts program.

Information may be displayed on this page similar to MSG, but the programmer can control the location on the screen to which it is written by defining the row and column of the first print position, starting from the left. The number of spaces to be cleared can also be programmed. The length of the data, which may be from 1 to 60 characters, will replace previously programmed data. It writes left to right, starting from the first position and going the length of characters programmed. If data overwrites a line, it will wrap around to the next line.

There are special commands to clear the entire screen, activate it or deactivate it. These are shown in the following examples.

(DISP,C) Clear entire display area

(DISP,0) Turn off display page

(DISP,A,1,1,4 TEST) If an "A" follows the DISP command, it will switch to the display page immediately when the program containing it is run.

(DISP,1,7,10,TEXT...) Start at the location of Row 1, Column 7, clear 10 spaces and begin.

(DISP,1,7,10,TEXT...,9,1,0,MORE TEXT....)
Same as above, but also write to new new location of Row 9, Column 1, no spaces cleared.

Be sure to clear the appropriate number of characters of the previous data if it was longer than the new input. If not, something like the following example could result:

```
(DISP,1,10,0 First Time)
.
.
(DISP,1,10,0 Done)
```

The first display will show: First Time
The second will show: Donet Time

because the extra characters of the first data, "First Time", were not cleared before the data "Done" was entered.

O. WRITING TO THE DATAFILE .PP, DATA

The command DATA will permit the parts program to write data to a dedicated file called *DATAFILE .PP* in memory. This file may later be renamed, edited, etc.

This command has the simple form of (DATA,text). The text is written sequentially to the file DATAFILE .PP in memory. The text may take the same form as a MSG command in the use of variables. There is no need or provision to open or close this file in the parts program because it is done automatically with the use of this command.

Specific error messages will appear on the status line if a parts program, which uses this function, is run when the file DATAFILE.PP already exists (or if the file uses all available memory.)

There are many potential uses for this function, such as making a log as parts are manufactured, or even creating a parts program from an inspection routine or through computations. (This program can be renamed later.)

NOTE: **When this function is used, the Unidex 16 must remain in machine mode. No other modes of operation are permitted.** ***The operator may leave machine mode, but this command will become disabled and further writing to the DATAFILE .PP from the parts program will be ignored.***

P. SIX-AXIS SYNCHRONIZED CONTOURING, SYNC

When both sets of axes (X,Y,Z and U,V,W) are programmed to move, they can perform contouring synchronously, i.e., both sets start and end simultaneously. To turn on the six-axis synchronized contouring, program:

```
(SYNC,1)
```

This command locks the U,V,W move onto the X,Y,Z move. The X,Y,Z vector speed is programmed by an F codeword and the U,V,W move simply follows along with the X,Y,Z. In case the X,Y,Z move is not programmed, the previously programmed E codeword is still used as the vector speed of the U,V,W move.

In order to turn off the synchronization, use:

```
(SYNC,0)
```

Synchronization is a modal command.

Q. ACCEL/DECEL TIME CONSTANT, ADTC

The ADTC command turns on the accel/decel function. The time constant (in milliseconds) for each axis may be specified, along with the feedrate threshold (in inches or millimeters per minute). (If time constant and feedrate threshold are not specified, the ones set in the EEPROM will be used.) Example:

```
(ADTC,X65,Y120,F200.)
```

The ADTC command overrides the accel or accel/decel, time constants and feedrate thresholds set in the EEPROM, but not the

G8/G9, H8/H9 command. The ADTC command stays in effect throughout the program unless cancelled by an (ADTC,D) command.

The (ADTC,D) command causes Unidex 16 to revert back to the accel/decel information set in the EEPROM. You may also program:

(ADTC,D,Z300)

This command tells all axes to default to the parameters set in the EEPROM, except the Z axis, which will accelerate and decelerate at a time constant of 300 milliseconds.

Here is a brief explanation of the three ways to utilize accel/decel.

1. Automatic Accel/Decel

When the user's programming feedrate exceeds the feedrate threshold set in the EEPROM, accel/decel automatically turns on, using the time constant set in the EEPROM.

2. Programmable Accel/Decel

a. G code Accel/Decel

The acceleration function can be programmed with a G8 (X,Y,Z) or H8 (U,V,W), using the time constant set in the EEPROM. These codes effect only the program block on which they are located.

The accel/decel function can be programmed with a G9 (X,Y,Z) and H9 (U,V,W), using the time constant set in the EEPROM. These codes effect only the program block on which they are located.

b. ADTC Accel/Decel

The accel/decel function can be programmed by the ADTC command. The time constant is now obtained from the user's program, not the EEPROM, unless no time constant is specified in the program.

The ADTC command remains in effect once it is entered, unless:

1. (ADTC,D) returns all accel/decel functions to whatever parameters are set in the EEPROM.

2. Another ADTC command is entered to change the accel/decel information.

3. A new parts program is run.

TIME CONSTANT

When one axis is included in the ADTC command, (ADTC,X65,F200.) for example, the axis is accelerated and decelerated at the time constant specified in the command. If two or three axes from one group are included in the same command (ADTC,X65,Y150,Z100,F200.) for example, the highest time constant specified is the one that is used, when more than 1 axes is moved. If an axis from the other group (U,V,W) is also included in the command, it retains its own time constant. Example:

```
(ADTC,X65,Y130,U260)
```

MOVE	TIME CONSTANT USED
X10	65

Y10	130
Z10	Parameter setting
U10	260
V10	Parameter setting
W10	Parameter setting
X10 Y10	130
X10 Z10	Greatest of 65 or parameter setting
X10 Y10 Z10	Greatest of 130 or parameter setting
Y10 U10	130 for Y/260 for U

Unidex 16, in order to execute accel/decel's time constant, must divide it by 32.768. The range of permissible time constants is 65 to 8356 milliseconds. This is how the time constant parameter (2-255 range) is calculated. (Refer to chapter 6.) Unidex 16 "rounds off" this value after the division. Therefore, close values, such as 65 to 68, will result in the same time constant. Example:

65/32.768 = 1.98 (2)
68/32.768 = 2.08 (2)

FEEDRATE THRESHOLD

When the feedrate threshold specified in the ADTC command is exceeded by the programmed feedrate, accel/decel will be enabled.

R. INTERRUPT, INT

The interrupts come in from the I/O bus or the MST bus on the CRT board.

INT4 - highest - pin 50 of J5, on I/O bus of CRT board
INT3 - 2nd " - pin 49 of J5, on I/O bus of CRT board
INT2 - 3rd " - pin 48 of J5, on I/O bus of CRT board
INT1 - 4th " - pin 47 of J5, on I/O bus of CRT board

CIRQ - 4th " - pin 50 of J4, on MST bus of CRT board

INT4, INT3, INT2 and INT1 are all negative edge triggered interrupts, i.e., the interrupt can only be generated when the signal goes from high to low.

CIRQ can be generated on any signal edge, both rising and falling.

The user can program any one of these interrupts to do certain interrupt functions.

FORMAT :

```
(INT4,n,xxxx,VAR1,VAR2,...)
(INT3,n,xxxx,VAR1,VAR2,...)
(INT2,n,xxxx,VAR1,VAR2,...)
(INT1,n,xxxx,VAR1,VAR2,...)
(CIRQ,n,xxxx,VAR1,VAR2,...)
```

Any of the interrupt functions must occupy its own block of program.

The "xxxx" gives the "interrupt service" entry point or subroutine.

The "VAR1,VAR2,..." option shown in the above example, allows for parameter passing when using interrupt options 1, 2 and 4 (subroutine cases). See section 10-7 E on parameter passing and global/local variables).

The "n" stands for one of the following options:

n=0 - Disable interrupt

n=1 - Enable type 1 interrupt. Unidex 16 will not stop or abort the current move when jumping to subroutine xxxx. It will perform defined functions (I/O, mathematical, system variable input/output, MST functions) as long as these are not axis moves. After the xxxx subroutine is finished, Unidex 16 continues the next block of functions.

n=2 - Enable type 2 interrupt. Unidex 16 will stop the current move and abort any functions left in this block, while remembering the current machine position. After finishing subroutine xxxx, Unidex 16 will return to the next block of program.

n=3 - Enable type 3 interrupt. Unidex 16 will stop the current move, abort any functions left in this block and remember the current machine position. It will then jump to the defined entry point xxxx, but will not return to the next block when finished. Unidex 16 will look ahead while interpreting the parts program during option 3. Unidex 16 does not look ahead during any of the other interrupt options. This makes program execution slower.

You may enable a "dummy" interrupt function with n=2, 4 or 5 in order to stop Unidex 16 from looking ahead. This is useful when you want to enable the user-programmable softkey function but do not want Unidex 16 to look ahead in order to insure that no blocks of program are skipped over. (See Chapter 13, Unidex 16 Options, for details on the User-Programmable Softkey Option.)

n=4 - Enable type 4 interrupt. Unidex 16 will finish all of the functions in this block before going to subroutine xxxx. After finishing the subroutine, Unidex 16 will return to the next block of program.

n=5 - Enable type 5 interrupt. Unidex 16 will finish all of the functions in this block. It will then jump to the defined entry point xxxx, but will not return to the next block when finished.

NOTE: Once serviced, interrupts (INT1, INT2, INT3, INT4) will remain enabled until disabled by the parts program. You may also clear an interrupt yourself with a (INTn,0) or (CIRQ,0) command. Once disabled, an interrupt can only be used again if reenabled from within the parts program.

(CIRQ) will be disabled after being serviced.

S. UMFO, PROGRAM-CONTROLLED MFO

The front panel switch MFO (Manual Feedrate Override) can be overridden by user-controlled MFO by programming:

(UMFO,n,xxxx)

The "n" represents the UMFO option:

n = 0 - Disable UMFO function. Unidex 16 responds to front panel MFO switch

n = 1 - Enable UMFO function. Unidex 16 does not respond to front panel MFO switch.

The "xxxx" can represent the MFO rate range, from 0% to 200%. Example:

(UMFO,1,150)

In the above example, the UMFO function is enabled and the feedrate is 150% of the programmed feedrate.

The "xxxx" can also represent a variable, whose value may range from 0 to 200. Example:

(UMFO,1,VAR1)

In the above example, the UMFO function is enabled and the feedrate becomes whatever per cent of the programmed feedrate VAR1 specifies.

T. UMSO, USER-CONTROLLED MSO

The front panel switch MSO (Manual Spindle Speed Override) can be overridden by user-controlled MSO by programming:

(UMSO,n,xxxx)

The "n" represents the UMSO option:

n = 0 - Disable UMSO function. Unidex 16 does responds to front panel MSO switch.

n = 1 - Enable UMSO function. Unidex 16 does not respond to front panel MSO switch.

The "xxxx" can represent the MSO rate range, from 0% to 200%. Example:

(UMSO,1,150)

In the above example, the UMSO function is enabled, and the spindle speed is 150% of the programmed spindle speed.

The "xxxx" can also represent a variable, whose value may range from 0 to 200. Example:

(UMSO,1,VAR1)

In the above example, UMSO is enabled and the spindle speed becomes whatever percent of the programmed spindle speed VAR1 specifies.

U. SAFE ZONE, ZONE

The Safe Zone Command, which is a software limit, applies to Unidex 16 system software "2U" and later. The Safe Zone is established by defining one or more restricted zones, i.e., spaces(s) to where an axis (or axes) cannot travel .

The purpose of this function is to ensure that:

1. Fixtures within a workspace are not encountered
2. Limits beyond the workspace are not exceeded

The space restricted by the Safe-Zone command may be defined by only one group of axes (i.e., XYZ and UVW).

Multiple Safe Zones may be established within a program. (Each requires 50 bytes of user RAM.)

The number of safe zones to be included in a program must be fixed at the beginning of the parts program. This is done with the command:

(DZON,n)

(This command is valid in the parts program only.) In this command, "n" is the number of safe zones to be written into the program; "n" must be an integer, ranging from 1 to 65535, and should appear only once in the beginning of the parts program. If "n" is mistakenly entered as less than the actual number of safe zones in the program, the following error message will appear when the program begins to run:

***** STATUS: error, undefined safe zone *****

The Safe Zone command itself is in the following format (bracketed entries indicate optional information):

(ZONE,m,[E/D],[Xnnn,Xnnn],[Ynnn,Ynnn],[Znnn,Znnn],
[Unnn,Unnn],[Vnnn,Vnnn],[Wnnn,Wnnn]

The Safe Zone command can be entered in the parts program and later updated in the mdi mode.

The following definitions will explain each of the Safe Zone command parameters.

ZONE The Safe Zone Command

m The Safe Zone Command number. This may not be less than the value of 1 or greater than the number of safe zones established by the prior command (DZON,n)

E/D Enable or Disable the safe zone command. Disable is the default, so to enable the safe zone command, enter "E". "E" is a modal function, and will remain active until disabled by "D".

"E" may be entered in the safe zone command as shown in the previous example command, or may be entered later as:

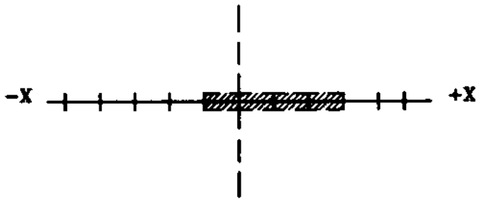
(ZONE,m,E)

Xnnn,Xnnn Enter the X values (+ nnn) to establish the space along the X axis that is restricted. For example:

(ZONE,1,E,X-1.,X3.)

This command would restrict a one-dimensional space along the X axis.

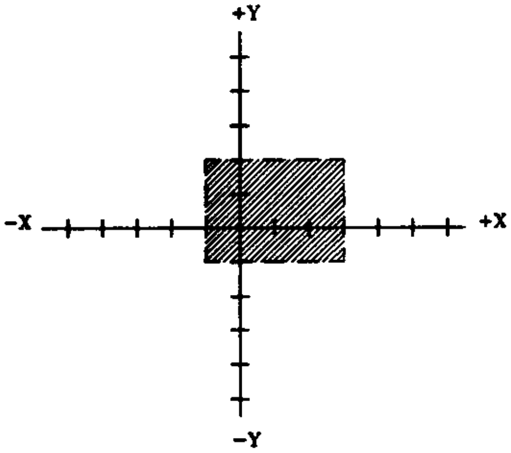
The shaded space in the following illustration is the restricted zone for the above example command.



NOTE: The space established may involve one axis (as shown in the prior example), as well as two or three axes (as will be illustrated next).

Ynnn,Ynnn Enter Y values (+ nnn) to establish a space along the Y axis as the restricted zone. For example:

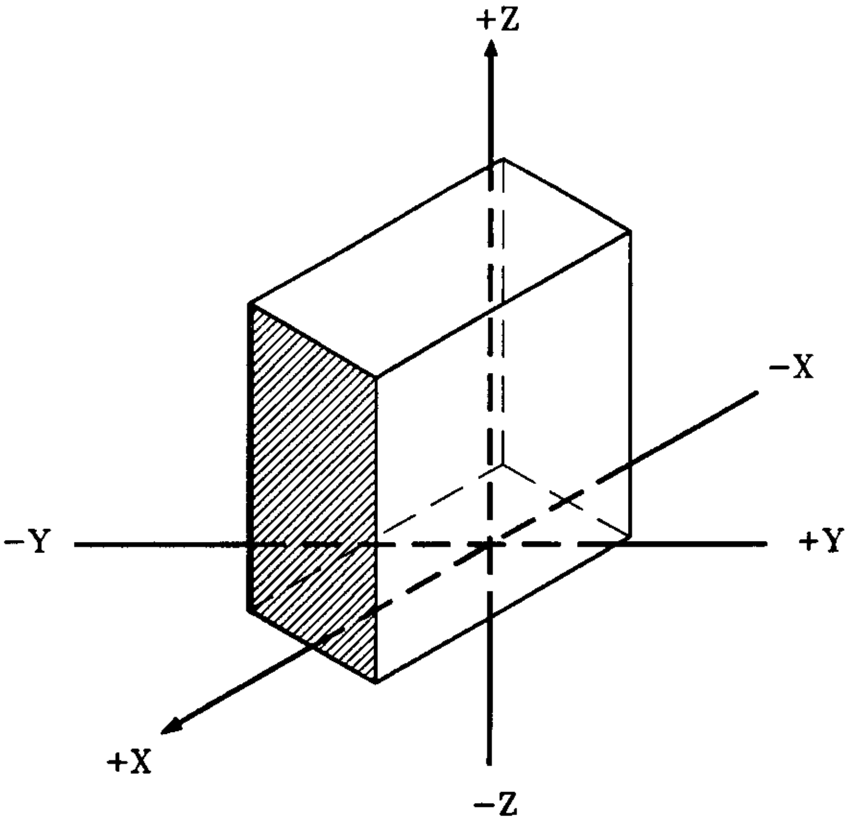
(ZONE,2,E,X-1.,X3.,Y-1.,Y2.)



The shaded space indicates the restricted zone.

Znnn,Znnn Enter Z values (+ nnn) to establish a space along the Z axis as a safe zone. For example:

(ZONE,3,E,X-1.,X3.,Y-1.,Y2.,Z0,Z4.)



The volume illustrated above indicates the restricted zone.

Unnn,Unnn The same as the XYZ group. These axes may be in-
Vnnn,Vnnn cluded within the same Safe Zone command as the
Wnnn,Wnnn XYZ axes. For example:

```
(ZONE,5,E,X2.,X4.,Y-1.,Y3.,U3.,U4.,V-1.,V4.)
```

However, this will establish two restricted zones: one for the XY axes and one for the UV axes.

A Safe Zone command may be disabled by programming the command:

```
(ZONE,m,D)
```

where "m" is the Safe Zone command number. If you program (ZONE,m,E), you will re-enable the command. The parameters will be the same.

To change the parameters of a Safe Zone command, just reprogram it. For example:

```
(ZONE,2,E,X10.,X15.,Y6.,Y12.)
```

This command may be changed by reprogramming its parameters, as:

```
(ZONE,2,E,X20.,X25.,Y11.,Y17.)
```

The Safe Zone command is valid for linear as well as circular moves.

A variable, such as the relative position, may be used to establish a restricted zone as well. For example:

```
(ZONE,4,E,X10.,X15.,Y=$YRP+5.,Y=$YRP+10.)
```

The above command specifies that Safe Zone #4 is set from X10. to X15., and from Y5. to Y10. relative to the Y axis' current position.

If the axes are moved into a restricted zone, the status line will display the error message:

***** STATUS: error, moving toward safe-zone #n *****

NOTE 1: Each time G92/H92 is updated to the new axes positions, you must program new Safe Zone command information.

NOTE 2: Any axis with the Safe Zone function enabled cannot be commanded to go home. The Safe Zone command must be disabled first.

If operating with an Aerotech Tool Changer, however, the Z axis may still Go Home during a tool operation, even when the Safe Zone is enabled.

NOTE 3: The Safe Zone function can restrict axis movement in the machine run, mdi and jog modes. (It applies to the jog mode and the mdi mode only when they are executed from within a parts program.)

NOTE 4: The Safe Zone function is inoperable when Unidex 16 is in the Joystick Slew Mode or Joystick Digitize mode.

NOTE 5: The Safe Zone function is inoperable for any axis operating in the Axis Free-Run Mode.

V. CUTTER COMPENSATION, CCP

The cutter compensation command is used to adjust the tool diameter information stored in the tool file (or instead of the tool file).

This information is used when ICRC is activated. The value of CCP can be positive or negative. Example:

(CCP,T1,-0.0011,T2,0.0018)
or
(CCP,T1=VAR1,T2=VAR2)

In the above example, the T codewords specify which tools are to be adjusted and the variables or values following the T codewords specify the amounts of compensation.

Unidex 16 bases its ICRC calculations on tool diameters in the tool file and CCP values in the parts program (if any).

Both tool diameters and CCP values can be expressed as variables.

CCP is a modal command and must be set as shown in the above example.

SECTION 10-3 ICRC

ICRC stands for Intersectional Cutter Radius Compensation. In cutting a workpiece, sometimes the radius of the cutter must be taken into consideration. For example, when an endmill is used to cut the sides of a workpiece, the center of the endmill follows the programmed path. The outside edge of the endmill cuts around the actual workpiece.

Cutter radius compensation is an option which allows the operator to program the center of the cutter in such applications, so that the outside edge of the endmill cuts along the programmed path. Without this option the operator would have to offset the actual piece dimensions with the radius of the tool. When programming angles other than 90 degrees is needed, it is no longer just a radius offset. This option dramatically decreases the programming effort by handling all the X and Y axis offsets. Also, ICRC option allows the same program to be used with tools of different diameters just by changing the radius information in the T table.

A. ICRC COMMANDS

There are 3 commands to use:

G40 - Cutter compensation/Offset, cancel
G41 - Cutter compensation-Left, turn ON
G42 - Cutter compensation-Right, turn ON

The commands G41 and G42 are oriented in the direction of cutter motion. Example:

- A G41 causes the cutter to make a path to the left of the nominal path by the amount of the radius determined from the T table. Left is relative to the direction in which the cutter is moving.

- A G42 causes the cutter to make a path to the right of the nominal path by the amount of the radius determined from the T table. Right is relative to the direction in which the cutter is moving.

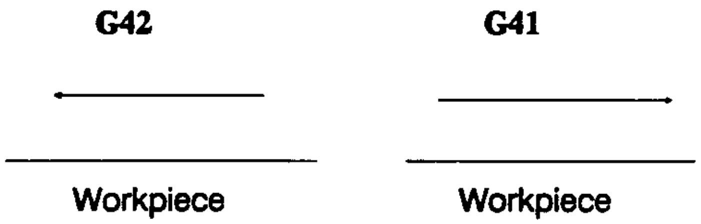


Figure 10-5 illustrates when to choose G41 or G42.

FIGURE 10-3: ICRC G41/G42 COMMANDS

B. START-UP

To start ICRC, enter a G41 or G42 followed by a start-up move before starting a cut. Then the operator can program the center of the tool to follow the workpiece contour.

As you can see in figure 10-2, the solid line is the shape to be cut. The cutter center traces the dotted line, which is offset from the true shape of the workpiece by the amount of the radius.

The following example shows all X position registers, assuming Unidex 16 is in the middle of move #3 at the dashed circle position.

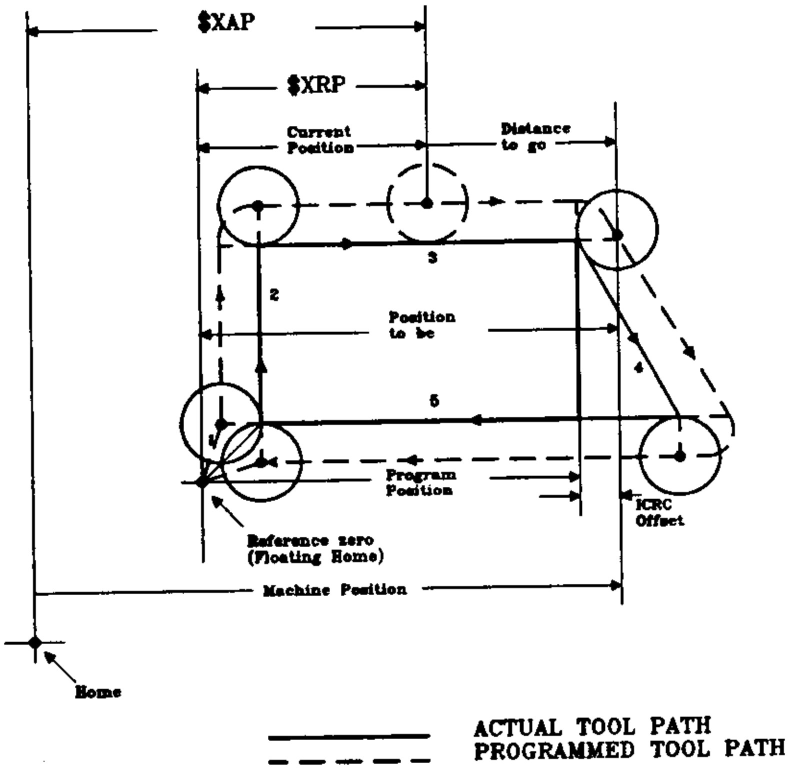


FIGURE 10-4: PROGRAMMED VS. CUTTER PATH

Figure 10-4 demonstrates the tool adjustments necessary to accommodate ICRC.

When making a linear to linear move using ICRC, you may switch from G41 to G42 and vice versa with no problem.

For the sake of accuracy however, when making a linear to circular or circular to linear move and switching from G41 to G42, a transitional move containing a G40 (cancel ICRC) must be placed between the two.

Another method of switching sides with ICRC is to break the transitional move (the one between linear and circular) into two moves. Execute one in G41 and one in G42.

C. CANCELLING ICRC

To end ICRC, make an ending move to direct the cutter away from the workpiece after cutting, using the G40 command to turn off the ICRC.

ICRC must be cancelled before a tool change or M2 command. After a tool change, to turn on ICRC again, program G41 or G42.

D. CUTTER COMPENSATION

(See section 10-2 V for information on the CCP command.)

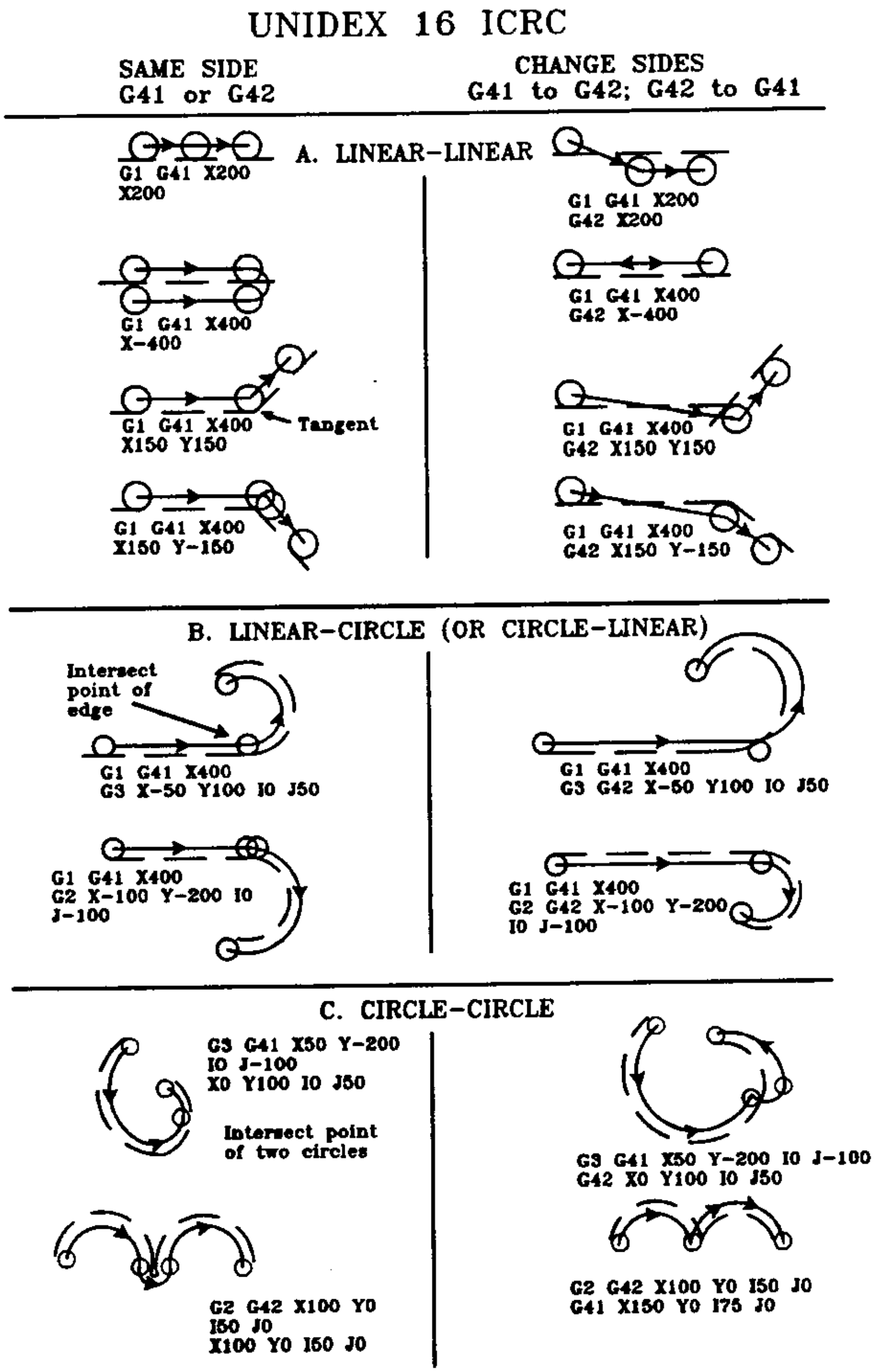


FIGURE 10-5: ICRC CONFIGURATIONS

CHAPTER 12: VARIABLES AND MATH PACKAGE

SECTION 12-1 VARIABLES

One of the powerful features of Unidex 16, which allows a motion control program to be written with maximum flexibility, is the variable. A variable in programming is like a variable in algebra. It has a name but its value is not fixed.

There are three types of variables. User variables, system variables and I/O variables. User variables are defined by the user. System variables are inherent to Unidex 16. I/O variables are defined by data on the I/O bus.

NOTE: **If you are unfamiliar with the relationship between binary and hexadecimal numbers, refer to appendix 2.**

A. USER VARIABLES

The user variables can have user-defined names of from two to four characters. The first two characters must be letters, while the last two can be alphanumeric characters or can be omitted altogether. Some examples are: VAR1, PART, AA, XRMS. Some illegal cases are: A, A/D, X2DR.

If system global variables are defined with a (DSGV,n) command (explained in the next subsection), then system global variables can be listed as $Gnn (where "nn" can be any number, limited only by memory.)

The purpose of variables is to retain changing data during program execution. This data is then used when making moves or performing other operations. Therefore, when a program is being written, the programmer need not know the exact value of the variable. The actual value can be determined at the time of execution.

B. SYSTEM VARIABLES

System variables are built into Unidex 16 and have fixed names which always begin with a dollar ($) sign.

There are three groups of system variables. The first group is the current absolute position registers. They are:

$XAP: Current X axis absolute coordinate in programming steps
$YAP: Current Y axis absolute coordinate in programming steps
$ZAP: Current Z axis absolute coordinate in programming steps
$UAP: Current U axis absolute coordinate in programming steps
$VAP: Current V axis absolute coordinate in programming steps
$WAP: Current W axis absolute coordinate in programming steps

All absolute registers use the home position as their origin. Home position is the position to which the type 2 commands (REF) and (HOME) refer. These variables cannot be altered, they are "read only" registers.

The second group is the current relative position registers. They are:

$XRP: Current X axis relative coordinate in programming steps
$YRP: Current Y axis relative coordinate in programming steps
$ZRP: Current Z axis relative coordinate in programming steps
$URP: Current U axis relative coordinate in programming steps
$VRP: Current V axis relative coordinate in programming steps
$WRP: Current W axis relative coordinate in programming steps

All relative registers use the floating home as their origin. Floating home is the reference position established by the G92/H92 codeword. The relative position registers can be altered by using the G92/H92 codeword to change the floating home position.

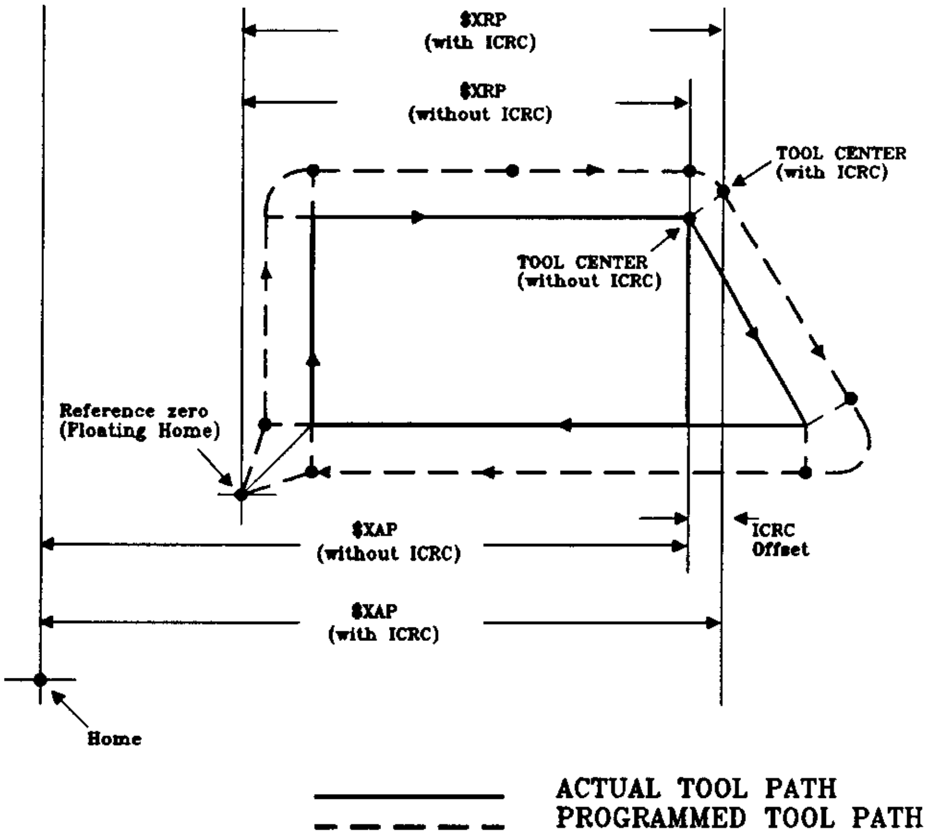


FIGURE 12-1: TOOL CENTER IN RELATION TO $XAP & $XRP

Figure 12-1 shows the absolute and relative position registers in relation to the tool center (both with and without ICRC). Whether ICRC is on or not effects the contents of both the absolute and relative position registers.

The third group of system variables are:

1. *Time of Day*. The $TOD system variable may be used as a message to output the time and date in ASCII code. It will be displayed on the screen in the same format as the Parameter Clock mode.

2. *$MFO System Variable*. The Manual Feedrate Override system variable may be used for a computation or as a message output of the present MFO.

   By using $MFO in a conditional jump command, it's value can determine program flow as well. For example:

   (JUMP,ENT1,$MFO.GT.1.5)

   The output is floating point and is output as a multiplier, i.e., 100% is output as 1.0, 145% is output as 1.45, etc.

3. *$MSO System Variable*. (Explanation for $MFO, above, applies to Manual Spindle Speed Override as well)

C. I/O VARIABLES

There are 2048 I/O ports that can be used for data input/output. Each one is like a variable and can be used to transmit or receive an 8 bit piece of data from the I/O channel. The I/O ports are:

$000: I/O port #0
to
$7FF: I/O port #7FF

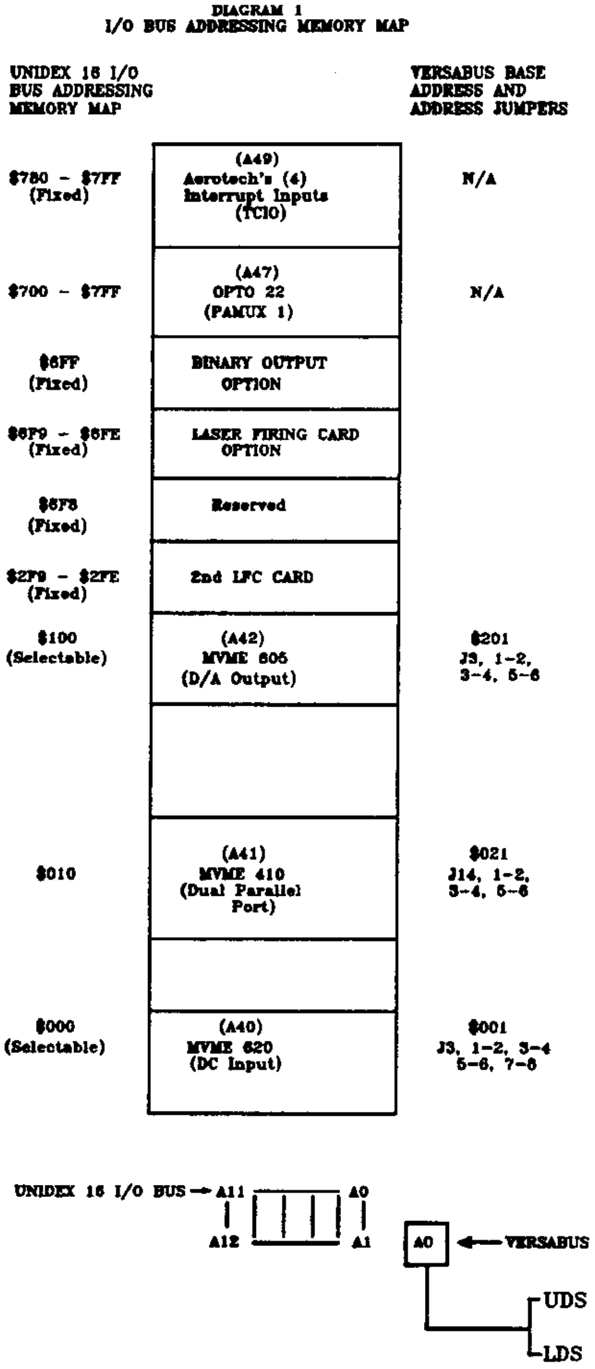


DIAGRAM 1
I/O BUS ADDRESSING MEMORY MAP

UNIDEX 16 I/O BUS ADDRESSING MEMORY MAP		VERSABUS BASE ADDRESS AND ADDRESS JUMPERS
$780 - $7FF (Fixed)	(A49) Aerotech's (4) Interrupt Inputs (TCIO)	N/A
$700 - $7FF	(A47) OPTO 22 (PAMUX 1)	N/A
$6FF (Fixed)	BINARY OUTPUT OPTION	
$6F9 - $6FE (Fixed)	LASER FIRING CARD OPTION	
$6F8 (Fixed)	Reserved	
$2F9 - $2FE (Fixed)	2nd LFC CARD	
$100 (Selectable)	(A42) MVME 605 (D/A Output)	$201 J3, 1-2, 3-4, 5-6
$010	(A41) MVME 410 (Dual Parallel Port)	$021 J14, 1-2, 3-4, 5-6
$000 (Selectable)	(A40) MVME 620 (DC Input)	$001 J3, 1-2, 3-4 5-6, 7-8

FIGURE 12-2: I/O BUS ADDRESSING MEMORY MAP

The number after the dollar sign has to be a hexadecimal number.

On the I/O bus, there are 12 address lines; when data is transferred over the bus, the port number becomes the lower 11 address lines in binary form. The most significant bit is always a zero. Many peripheral devices can attach to this same bus and each respond to a unique address.

Consecutive addresses can be grouped together to form a word of 16, 24 or 32 bits of data, using the I/O format statement.

D. EXTENDED TYPE 1 CODEWORDS

Two kinds of codewords are designated for data transfer between variables: Assignment Codewords and Parametric Codewords.

1. ASSIGNMENT CODEWORDS

Assignment codewords are used to assign a value to a variable. For example:

```
VAR1=10.0
AA=20
VAR1=AA
```

In the second example above, AA will have a value of 20.0 assigned to it because Unidex 16 inserts a decimal point for you. (For the sake of clarity, it is recommended that you insert the decimal point yourself.)

a. I/O Channel Constant

Unidex 16 can assign the current value of an I/O channel to a variable. Example:

```
INP3=$004
```

In the above example, Unidex 16 assigns the current value of the I/O port #4 to the variable INP3. Upon execution, Unidex 16 will read data from I/O port #4 and transfer it to the variable INP3. Therefore, an input operation is done and the data is retained as the variable INP3.

Once the value of a variable is assigned, it will stay until another assignment is made.

b. Hexadecimal Constant

For some applications, a hexadecimal constant may be assigned to a variable. This is especially useful for logic functions. Example:

```
VAR1 = H,1234
```

The H before the 1234 lets Unidex 16 know 1234 is a hexadecimal number.

No decimal point is allowed in a hexadecimal constant and it is limited to 8 digits.

c. Character Constant

An assignment codeword can also consist of characters. No more than four characters can be assigned to one variable. Therefore, for a string of characters, use multiple variables. For example:

```
VAR="GOOD" VAR2=" MOR" VAR3="NING"
```

A character string may include any alphanumeric or punctuation character.

2. PARAMETRIC CODEWORDS

Parametric codewords allow variables to be assigned to codewords such as X, Y, etc. (This applies to all type 1 codewords except G, H, M and T.) For example:

```
X=VAR1
Y=-AA
```

The first letter is the address, which serves the same purpose as the addresses in the RS-274-D standard. Therefore, X means this is an X codeword, etc. The following information specifies the magnitude and sign of the codeword. In the above case, if variable VAR1 has a value of 10.0 at the run time, the final codeword will be X10.0. If variable AA has a value of -20 at the run time, the final codeword will be Y20.

This programming technique can be used to generate parametric subroutines. Subroutines can be written with dimensions expressed in variables, (the value of each variable will be determined by the calling program). The calling program calls the subroutine and passes specific information into it. This way, a generic subroutine can be written to perform a sequence of operations which may appear several times within the program but whose dimensions may vary each time. The parametric subroutine saves the programmer from writing this sequence many times.

This feature greatly enhances the capabilities of Unidex 16 by providing to the customer a means of writing a customized canned cycle, designed to meet specific needs.

E. I/O FORMAT, IOFT

When an input/output operation is done, using assignment code words through an I/O channel, the format and the length of data can be programmed.

It may be entered as (IOFT,BCD,n) for specifying BCD (Binary Code Decimal) or (IOFT,BIN,n) for specifying binary. The "n" may be the number 1, 2, 3 or 4. This is the number of bytes to be transferred.

Internally the value of a variable is always stored in a 4 byte field in floating point format with 3 bytes of positive binary mantissa. During I/O operation, this 3 byte field is transferred to or from I/O port. Therefore, a variable is always treated as a positive integer during I/O operations. (In cases where the value is fractional or greater than 3 bytes long, erroneous results will occur.) For example:

```
(IOFT,BIN,n) or (IOFT,BCD,n)
```

The first field after IOFT specifies data type, whether in binary or BCD representation. When BIN is specified, the absolute value is sent out. When BCD is specified, the absolute value is converted into BCD format and then sent out.

The second field, represented by "n" in the example, is a number of 1, 2, 3 or 4. This is the number of bytes of data to be transferred. In either BCD or BIN, up to 4 bytes of data can be transferred with one command. The most significant byte is transferred first to the address specified by the I/O code word, and the following bytes are transferred to the following locations. Example:

```
(DVAR,VAR1)
VAR1=1234567
(IOFT,BCD,4)
$008=VAR1
```

In this example, variable VAR1 is given a value of 1234567, then this number is sent to the I/O location in BCD format. I/O location 008 receives "01", location 009 receives "23", location 00A receives "45" and location 00B receives "67".

1. BCD Data Type

Programming $001 = 34 will output the BCD number 34.

Programming $001 = H,34 will output the BCD number 0, since Unidex 16 treats 00000034 as a floating point number (converted to BCD is 0).

Programming $001 = H,88000046 will output the BCD number 34, since Unidex 16 treats 88000046 as a floating point number and converts it to BCD 34.

Programming $001 = BTF(H,34) will output BCD number 52.

Programming $001 = VAR1 + SIN(12/VAR2) outputs the BDC number of the result.

Programming VAR1 = $001 will input BCD data, then store it as floating point format.

Programming VAR1 = FTB($001) will input BCD data, then store it as binary.

2. Binary Data Type

Programming $001 = 34 will output BIN number 88000046.

Programming $001 = H,34 will output BIN number H,34.

Programming $001 = FTB(34) will output the BIN number H,22.

Programming $001 = VAR1 + SIN(12/VAR2) will output the BIN of the result.

Programming VAR1 = $001 will input the BIN data and store it as binary.

Programming VAR1 = BTF($001) will input the BIN data and store it as floating point.

NOTE: Refer to appendix 2 for a chart on binary to hex conversions.

F. CONDITIONAL JUMP, JUMP

A conditional jump uses a type 2 variable as the deciding factor in whether a jump should be made or not. A jump command can be executed depending on whether a condition is true or not. Example:

```
(JUMP,nnnn,$XRP.EQ.VAR1)
```

This jump will be taken if the X axis coordinate equals what is in VAR1. The place to jump to is nnnn, which is defined by a define entry command. Legal test conditions for floating point format (real constants) include:

1. **.EQ. Jump if equal**
2. **.NE. Jump if not equal**
3. **.GT. Jump if greater than**
4. **.GE. Jump if greater or equal**
5. **.LT. Jump if less than**
6. **.LE. Jump if less than or equal**

As you can see here, the comparison in a conditional jump can be done between two variables. They can be user variables or system variables. You can compare a variable with a constant, as well. Example:

```
(JUMP,nnnn,$005.EQ.0.)
```

This will program Unidex 16 to read I/O channel #5 and jump to nnnn if all bits are low.

For binary format these conditions are limited to:

1. **.EQ. Jump if equal**
2. **.NE. Jump if not equal**
3. **.HI. Jump if higher than**
4. **.LS. Jump if lower than or same**

G. REPEAT LOOP, RPT

The repeat counter can be a user-defined variable. The value of the variable will be rounded off into an integer, then taken as the repeat count. Example:

```
VAR1 = 2.6
(RPT,VAR1
.
.
)
```

This loop will repeat 3 times, since 2.6 becomes 3 after truncation.

If a minus number is assigned to the variable, it will be converted to a positive number before rounded off. For example, if VAR1 = -2.6, the loop will also repeat three times.

H. SCALING FACTOR, SCF

The scaling factor used in the SCF command can be a variable. For example:

```
(SCF,X=VAR1)
```

I. USE OF VARIABLES WITH SUBROUTINES

NOTE: All variables are initialized to a value of zero when they are defined.

1. DEFINE VARIABLES, DVAR

A named user-variable in the main program must be defined by a DVAR command. Example:

```
(DVAR,VAR1,AA,XRMS,VAR2)
```

Above, four variables are defined: VAR1, AA, XRMS and VAR2. Their values are all set to 0.

Each variable occupies 12 bytes of RAM, 4 for the name, 4 for its current value and 4 for cutter radius compensation looking ahead. All user variables are stored in floating point format with 3 bytes of mantissa and 1 byte of sign and exponent. This memory will be taken from the stack, which is part of the user's RAM space. Therefore, the number of variables is limited by the user's memory. It is a good programming practice to define variables at the beginning of a program.

2. CALL SUBROUTINE WITH PARAMETERS, CLS

The parametric subroutine is one of the key attributes of contemporary motion control. It allows a sequence of operations to be programmed into a subroutine without knowing specific dimensions. This has two advantages:

a. Similar operations can be done with one subroutine by changing the value of parameters. The user can virtually create his own canned cycle and do modular programming.

b. The sequence of operation can have critical values determined at the run time instead of being preprogrammed. With proper probing and sensing devices, these value assignments can vary, depending on external conditions.

Both modular programming and adaptive control can be achieved.

The example below shows such a subroutine call:

```
(CLS,nnnn,1234,123.4,VAR1)
```

The nnnn is the subroutine name. Following the name are three parameters. These parameters can be actual values or variables. If a real value is used, this value will be passed into a subroutine. If a variable is used, the value of this variable will be passed into a subroutine. All variables must have already been defined by the DVAR or the DSGV commands, if in the main program. If in a subroutine, the variables must have been defined by DSGV, DFS or DVAR, if the global declaration (GLOB) is used.

3. DEFINE SUBROUTINE WITH PARAMETERS, DFS

The DFS command is used to define a parametric subroutine. It also defines variables implicitly. Example:

```
(DFS,nnnn,AAAA,BBBB,TRY1
X=AAAA Y=BBBB Z1.5 F=TRY1
.
)
```

This defines a subroutine with the name nnnn. Also, it defines variables AAAA, BBBB and TRY1. Only variables can be placed in a parameter list, real values are not allowed. The value of these variables will be determined at the run time by the calling program. If the DFS in this example is used in conjunction with the CLS in the previous example, values of these variables will be:

```
AAAA=1234.
BBBB=123.4
TRY1=Value of VAR1
```

In the subroutine, these variables can be used as distance, feedrate, repeat loop count, scaling factors, cutter radius compensation, etc.

4. DEFINE LIBRARY SUBROUTINE, DFLS

All features that are available for the DFS command also apply to the DFLS command. The only difference is the DFLS command preserves the machine statuses, as mentioned in chapter 8.

5. RETURN-VALUE INSTRUCTION (RVAL)

The (RVAL) command permits the value of variables to be passed back to the calling routine (main program or subroutine) by position rather by name. The position refers to the CLS (Call Subroutine) and DFS (Define Subroutine) listing of variables*. In the following example, VAR1 is in position 1, VAR2 is in position 2.

```
(DFS,SUB1,VAR1,VAR2,....
        .
    (RVAL,2)*
        .
    (RVAL,2,1)
```

- * This function is for *named* variables only. A real value or a $Gnn variable is not counted as a position.

In the previous example, the second line returns the value of the variable in position two (VAR2) of the calling routine, and the third line returns both. Note that these may be listed in any order.

6. ABORT SUBROUTINE, ABTS

Normally after a subroutine is executed, Unidex 16 returns to the main program. The ABTS command allows the current level of subroutine to be aborted and execution to continue at a place programmed by the user. Example:

```
(ABTS,nnnn)
```

The nnnn in the above example is an entry point defined by a (DENT) command. This command must be located inside the subroutine (which is enclosed in parentheses). Example:

```
(DFS,TEST
.
.
(JUMP,ENT1,VAR1.EQ.2)
(ABTS,ENT2)
(DENT,ENT1)
.
.
)
(DENT,ENT2)
```

Unidex 16 jumps to the entry point "ENT2" at the end of the subroutine "TEST" rather than going back to the main program, if VAR1 does not equal 2.

7. METHODS OF USING VARIABLES

There are conventions which must be observed when using variables in both the main section of a program and in subroutines. You have a choice of methods, depending on preference and requirements of the program. The three methods are:

a. Separate Variables for Program and Subroutine

This option was the only choice before Unidex 16 software version 2Y. The DVAR command initializes variables named in the listing as the main program variables. These may be accessed freely in the main program for any function.

To pass the value of a variable to a subroutine, the CLS command must list the main program variable as a parameter. The DFS command must list a variable to be used by the subroutine, and the value of the main program variable will be loaded into the subroutine variable. The subroutine may then use its variable for any functions. Main program variables may not be used in the subroutine for any functions except to assign a new value to the main program variable. The variables defined by the subroutine cannot be used by the main program. For example:

```
(DVAR,VAR1,VAR2,VAR3)
VAR1=350
(CLS,SUB1,VAR1,500)
.
.
.
(DFS,SUB1,SVR1,SVR2,SVR3
SVR3=SVR1+SVR2
VAR1=SVR3
)
```

In this example, the main program variables are VAR1, VAR2 and VAR3. The subroutine variables are SVR1, SVR2 and SVR3. When the subroutine is called, SVR1 has a value of 350, which was the value of VAR1. SVR2 has a value of 500, which was a real number in the parameter listing. SVR3 has a value of zero because there was no corresponding parameter in the CLS command. The second line of the subroutine assigns the value 850 to SVR3 (350 + 500). VAR1 is then updated by the subroutine (the only operation allowed on VAR1 in the subroutine) to the value of 850.

If a subroutine calls another subroutine, the same convention is used to pass values back and forth.

b. Global Declaration Option (GLOB)

The programmer may find it more convenient to declare all variables defined by the DVAR command to be global. The term global means accessible by both the main program and any subroutines used. This eliminates the need to list variables in CLS and DFS commands. Therefore, there is no need to pass a value back to the main program because the variables are used directly in all parts of the program. The Global declaration option is activated by the use of the command (GLOB) in the program. The DFS command may still define variables for use in the subroutine, but these are not global. Parameters may be passed with DFS as explained above, in method 1.

c. Special Global Variables (DSGV,n)

The use of the global variables of the form $Gnn works the same as the global variables explained above, but will save memory. The declaration (DSGV,n) sets up the total number of these variables the program will use, and is all that is required. The variables are initialized to zero, and may then be accessed by number. Because names are not used, there is a 1/3 savings in memory, and when a large number of variables are used, access should be faster than the old method. For example:

(DSGV,50)

sets up 50 global variables. $G22 would be variable #22, for instance.

SECTION 12-2 MATHEMATICAL PACKAGE

There are other variations to the assignment and parametric codewords.

A. ARITHMETIC OPERATIONS FOR REAL CONSTANTS

Arithmetic operations allow Unidex 16 to be programmed to its greatest flexibility. These operators apply only to real constants (floating point format). They include:

+	**Addition**
-	**Subtraction**
*****	**Multiplication**
/	**Division**
( )	**Module**
!	**Exponentiation**

These operators can be used in both assignment codewords and parametric codewords. Example:

X = 1.2*2/(78.789!23)+VAR1-44.0*VAR2

or

VAR1 = VAR2*(VAR3/2)+0.125!0.5

B. MATHEMATICAL FUNCTIONS FOR REAL CONSTANTS

Unidex 16 provides the following math functions for the most sophisticated operations. These functions apply only to real constants (floating point format). They include:

SIN(X)	**Sine function of X**
COS(X)	**Cosine function of X**
TAN(X)	**Tangent function of X**

In the above function, all angles are programmed in decimal degree. For example, SIN(22.5) means the sine function of 22.5 degrees.

ATN(X)	**Arctangent of X**

The arctangent function always results in an angle in radians. To do the conversion between degrees and radians, the following two functions are available:

DEG(X)	**Radians to degrees conversion of X**
RAD(X)	**Degrees to radians conversion of X**

The following two functions complement all of the above.

ABS(X)	**Absolute value of X**
SQR(X)	**Square root of X**

When a real number needs to be converted to an integer number, use:

INT(X) Integer of X

In the above function, the number is rounded off. For example a command of INT(123.05) would give 123 as the integer.

The function BTF (binary to floating) is used to convert binary format to floating format. Example:

BTF(H,2)

In the above example, the hexadecimal number 2 can be included in a floating format equation, since it will be converted.

Floating format and binary logic cannot be in the same equation unless one or the other is converted. For example, VAR1 = 2 + H,3 will not generate 5. You must enter:

2+BTF(H,3)
or
FTB(2).ADD4.H,3

The FTB command and logic functions in the second equation are covered in the next section.

These math functions can be used together with all arithmetic operations in parametric or assignment codewords. In descending order of priority, these operators can be listed as:

SIN,COS,TAN,ATN,DEG (highest)
RAD,ABS,SQR,INT,BTF
( )
!

*** and /**
+ and -

A higher priority operator is evaluated before lower ones in an equation. Two operators of the same priority will be evaluated by their sequence of appearance. This order matches the convention in the normal free-hand math format. Example:

```
X = XOFS+0.5*SIN(30)/(2.5+(SQR(0.75!2+0.25!2)))
Y = YOFS+0.5*COS(26+45/60+10/3600)/0.125*0.22
```

In the second equation, a cosine function of 26 degrees, 45 minutes and 10 seconds is specified using an arithmetic expression. Therefore, an angle in degrees, minutes and seconds can be easily programmed into the control.

C. LOGIC FUNCTIONS FOR BINARY NUMBERS

With binary format (hexadecimal constants and character constants), Unidex 16 provides specific logic functions.

With the exception of FTB, these functions may be followed by a number (represented by an n). This number indicates how many of four bytes (each bytes containing 8 bits) of memory location are effected by the function. (Unidex 16 assigns 4 bytes of memory location for each variable.)

MSB			LSB
B4	B3	B2	B1

If no number is entered after a function, the default number is 1, and only B1 is effected by that function.

The arithmetic operations for binary numbers are:

1. **FTB -**

Floating to binary. Converts floating format to binary format. For example, FTB(34) is equivalent to H,22.

NOTE: Do not mix binary logic and floating format mathematics together in the same equation without first converting one type with a BTF or FTB command.

2. **.NOTn. -**

This function requires only one operand. It changes ones to zeroes and zeroes to ones. Example:

VAR1 = .NOT2.VAR1

Bytes 1 and 2 of VAR1 are "notted". If it was H,F32, which in binary is:

B1	B2
00001111	00110010

It is now H,F0CD, which in binary is:

B1	B2
11110000	11001101

3. **.ANDn. -**

This function requires 2 operands in any sequence. It ANDs two variables. To explain ANDing, refer to the AND table below:

```
1 = 1 .AND. 1
0 = 1 .AND. 0
0 = 0 .AND. 1
0 = 0 .AND. 0
```

NOTE: *Both* must be a 1 before a 1 results.

An example of ANDing two variables is:

```
VAR1=VAR1.AND2.VAR2
```

In the above example, B1 and B2 of VAR1 will be ANDed with B1 and B2 of VAR2.

If VAR1 is H,38A, which in binary is:

B2	B1
00000011	10001010

and VAR2 is H,29C, which is:

B2	B1
00000010	10011100

then the resultant VAR1 is H,288, which is:

B2	B1
00000010	10001000

The ANDing function is used to mask off other bits while reading a particular bit. If, for example, data on I/O port #1 must be checked, you must compare it to the hexadecimal number which corresponds to the bits to be checked. Example:

```
VAR1=$001
```

VAR1 = VAR1.AND.H,1
(JUMP,nnnn,VAR1.GT.0)

VAR1 would be ANDed with H,1:

$001	0	0	0	0	0	0	0	1
H,1	0	0	0	0	0	0	0	1

ANDing byte 1 of I/0 port #1 with the hexdecimal number 1 enables you to check bit 1 of byte 1. If this results in a 1 for VAR1, it indicates that the bit is set and the jump should be made.

(If the bit had been 0, ANDing it with a 1 would have resulted in a 0.)

4. . ORn. -
This function requires 2 operands in any sequence. It ORs two variables. To explain ORing, refer to the OR table below:

1 = 1 .OR. 1
1 = 0 .OR. 1
1 = 1 .OR. 0
0 = 0 .OR. 0

NOTE: If *either* is a 1, a 1 results.

An example of ORing two variables is:

VAR1 = VAR1.OR.VAR2

In the above example, B1 of VAR1 will be ORed with B1 of VAR2.

If VAR1 is H,28A, which is:

B2	B1
00000010	10001010

and VAR2 is H,29C, which is:

B2	B1
00000010	10011100

the resultant VAR1 would be H,29E, which is:

B2	B1
00000010	10011110

ORing is usually used to set a bit once it has been read, despite its previous status. Example:

```
VAR1=$002
VAR2=VAR1.OR.H,1
$002=VAR2
```

VAR1 would be ORed with H,1:

$002	0	0	0	0	0	0	0	0
H,1	0	0	0	0	0	0	0	1

After VAR1 byte 1, bit 1 is ORed with H1, it is set and is assigned to I/0 port #2.

5. .XORn. -

Exclusive ORing. Requires two operands in any sequence. To explain XORing, refer to the XOR table below:

0 = 1 .XOR. 1
1 = 0 .XOR. 1
1 = 1 .XOR. 0
0 = 0 .XOR. 0

NOTE: Only when the two are *different* does a 1 result.

An example of XORing two variables is:

VAR1 = VAR1.XOR.VAR2

In the above example, B1 of VAR1 will be XORed with B1 of VAR2.

If VAR1 is H,28B, which is:

00000010 10001011

and VAR2 is H,6C9, which is:

00000110 11001001

the resultant VAR1 would be:

00000010 01000010
(or H,242)

Byte 2 of VAR1 would remain the same since it wasn't involved in the .XOR. function.

XORing is very useful because it can be used to flip the status of a bit. Once a bit has been XORED with a 1, it will change status. If a zero is XORed with a 1, it will become a 1. If a 1 is XORed with a 1, it will become a zero. Example:

```
VAR1=$001
VAR1=VAR1.XOR2.H,2
```

Byte 1 of VAR1 will be XORed with H,2:

	B1							
$001	0	0	0	0	0	0	1	1
H,2	0	0	0	0	0	0	1	0

Byte 1 of $001 will now read 00000001 and the status of bit 2, byte 1 will have changed after being XORed with a 1.

6. .LSLn. -

Logical shift left. This function requires 2 operands in a specific sequence. The first operand names the variable. The second specifies the number of bits to be shifted to the left. Example:

```
VAR1=VAR1.LSL.H,3
```

Only the bits in byte 1 will have its bits shifted left. They will shift left 3 spaces (specified by H,3). If the contents of byte 1 are:

10101010

the first 3 bits (101) will be replaced by the next 3 (010). The 3 empty spaces on the right will be replaced with zeroes. The resulting byte 1 will be:

01010000

An example where bytes 1 and 2 are effected is:

```
VAR1=VAR1.LSL2.H,3
```

In this case, the 3 bits to be shifted to the left in byte 1 will shift over to byte 2. The 3 bits to be shifted out of byte 2 will "fall off" (as in the first example). The 3 empty spaces on the far right of byte 1 will be replaced with zeroes. Example:

B4	B3	B2	B1
00000110	00100011	10010111	01101111

After VAR1 = VAR1.LCL2.H,3, VAR1 will be:

B4	B3	B2	B1
00000110	00100011	10111011	01111000

7. **.LSRn. -**

Logical shift right. This function is the same as .LSL, except it shifts to the right.

8. **.ROLn. -**

Rotate left. This function requires 2 operands in a specific sequence. The first operand names the variable. The second specifies the number of bits to rotate left. Example:

VAR1 = VAR1.ROL.H,3

Only the bits in byte 1 will rotate left 3 spaces. The first 3 bits will "wrap around", filling in the empty spaces on the far right. Example:

01010111

Becomes:

10111010

When 2 bytes are involved, as in:

VAR1 = VAR1.ROL2.H,3

the 3 first bits in byte 2 "wrap around", filling in the spaces on the far right of byte 1. Example:

B4	B3	B2	B1
00000110	00110111	01110010	11001010

will result in:

B4	B3	B2	B1
00000110	00110111	10010110	01010011

9. **.RORn. -**

Rotate right. This function is the same as .ROL, except it rotates right.

10. **.ADDn. -**

Adds 2 or more variables. Requires at least 2 operands in any sequence. For example:

VAR1 = VAR1.ADD.VAR2

Binary numbers are ADDed by carrying numbers as in conventional addition. Example:

```
 10111011
+01001011
1|00000110
```

In ADDing binary numbers in an 8 bit register, the last carried-over number is dropped.

11. .SUBn. -

Subtracts one variable from another. Requires 2 operands in a specific sequence. For example:

VAR1 = VAR1.SUB.VAR2
or
VAR1 = VAR1.SUB2.H,3E

Binary numbers are subtracted by borrowing numbers as in conventional subtraction. Example:

```
 10111011
-01001011
 01110000
```

12. .TSTA.nnnn -

The TSTA (test AND) function needs 2 operands in a specific order. The first specifies the variable to be tested. The second specifies which bits are to be tested.

This function is called TSTA because the result is true (1) only if *all* bits tested are 1. This function involves all 4 bytes (32 bits). An example of TSTA.nnnn is:

VAR1 = H,3F.TSTA.H,11

```
H,3F  001|1| 111|1|
H,11  000|1| 000|1|
```

VAR1 will equal 1 because both bits to be tested are true (1).

A practical example of TSTA is testing an I/0 port. Example:

```
(JUMP,ENT1,$002.TSTA.H,12.EQ.H,1)
        $002  0011 1010
        H,12  0001 0010
```

Since both test bits are true, the jump to ENT1 is accomplished.

13. .TSTO.nnnn -

The TSTO (test OR) function needs 2 operands in a specific order. The first specifies the variables to be tested. The second specifies which bits are to be tested.

This function is called TSTO because the result is true (1) if *any* of the bits tested is 1. This function involves all 4 bytes (32 bits). An example:

```
VAR1 = H,35.TSTO.H,3

H,35  0011 01|01|
H,3   0000 00 |11|
```

In the above case, one test bit is true and one is false. In TSTO just one needs to be true, so the result in the above example is 1, so VAR1 = 1. A practical application of TSTO is:

```
(JUMP,ENT1,$001.TSTO.H,6.EQ.H,1)

          $001   0010  0101
          H,6    0000  0110
```

The jump will be accomplished since at least one test bit is true. Another example is:

```
$023=H,3F.TSTO.H,C0

     H,3F  0011 1111
     H,C0  1100 0000
```

Data on I/O port $023 will be set to zero since no test bit is true.

The order of priority in which the logic functions are executed is as follows:

```
FTB  (highest)
.NOT. (second)
.ANDn.  .ORn.  .XORn.  .LSLn.
.LSRn.  .ROLn. .RORn.  .ADDn.
.SUBn.  .TSTA. .TSTO.
```

NOTE: The "n" shows byte-wide operation:
n = 1 or no n is 1 byte
n = 2 is 2 bytes
n = 3 is 3 bytes
n = 4 is 4 bytes

D. BIT MANIPULATION FOR THE I/O CHANNEL

New instructions BTRD and BWRT permit the writing and reading of single or multiple bits on the I/O channel. These commands also permit the use of variables for the I/O address. Only one I/O address one byte long may be processed per block. The following list explains the new bit manipulation commands.

1. (BWRT,$xxx,bv,...) where b=0-7 and v=0 or 1

An example of the above command would be:

```
(BWRT,$700,30,11) or
(BWRT,$70030,11) or
(BWRT,$70030,1)
```

All of the above examples set bit 3 to 0 and bit 1 to 1. As can be seen, the second comma is optional. The third example shows that the status of bit 1 is not entered. Therefore it defaults to the value of 1.

2. (BWRT,$xxx,VAR,...) VAR is a 2 digit ASCII variable

For example:

```
VAR1="30"
VAR2="11"
(BWRT,$700,VAR1,VAR2)
```

Similar to the first group, the bit number and value are determined by the value of an ASCII variable. Care must be taken to set the variable to an appropriate value. Allowed values are:

00,01,10,11,20,21,30,31,40,41,50,51,60,61,70,71

3. (BWRT,VAR,bv,...) VAR="$xxx" ASCII

For example:

```
ADDV="$700"
VAR1="30"
VAR2="11"
(BWRT,ADDV,VAR,VAR2)
```

Results of above example are the same as in (2).

4. (BWRT,VAR,v) VAR="xxxb"

Example 1:

```
ADDV="7001"
(BWRT,ADDV,0)
```

At address $700, bit 1 is set to 0.

Example 2:

```
(BWRT,ADDV)
```

Bit 1 is set to 1 by default.

The use of fixed values and variables may be mixed in a block of instruction.

Example 3:

```
VAR="7000"
 (BWRT,VAR,1,11)
```

At $700, bit 0 is set to 1, bit 1 is set to 1. In this case, do no use an implied "1" status for bit zero, or an error will result.

6. BIT READ COMMAND

The bit read function is similar to the bit write function in the structure and use of variables. A variable is specified and bits will be set in this variable if the specified tested bits are set. Once this variable is set up by this command, it may be used with logic functions in other blocks to alter program flow, etc. For example:

```
(BTRD,VAR1,$700,1,3,6)
```

$700 is tested for the status of bits one, three, and six. If any of these are a one, the matching bit in the variable VAR1 is set.

SECTION 13-10 TCIO MODULE OPTION

For the hardware and installation considerations of this option, see the Unidex 16 Hardware Manual.

The TCIO Module option provides the user with the following features:

- 4 (opto-isolated or logic) inputs to Unidex 16
- 4 input-activated interrupts
- Interface for the Opto 22 Pamux 1 card
- Reset circuitry
- Termination for the I/O bus
- Power input to the I/O bus
- Single-high (3U) VME card form factor
- Jumper selection for interrupts
- Jumper selection for interrupt edge sensing

A. INPUTS

In order to read the 4 inputs, Unidex 16 will read the first 4 bits of the input/interrupt status register. An input is indicated when the appropriate bit is set high. The input/interrupt status register is always located at address $780. The second 4 bits are designated as the interrupt flags.

INPUT/INTERRUPT STATUS REGISTER

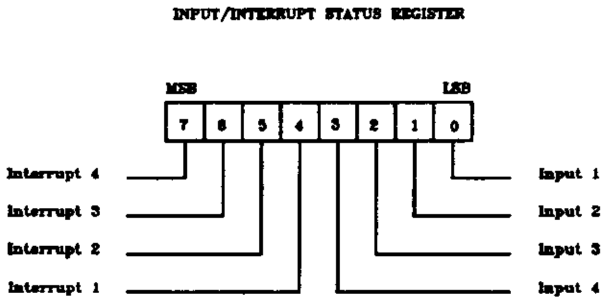


DIAGRAM 1

Following is an example of programming Unidex 16 to read the inputs:

```
VAR1=$780
(MSG,<2>,#H:VAR1)
```

In the above example, Unidex 16 will read the inputs and store the data in a user's variable (VAR1).

See chapter 12 for more on the use of variables and the logic functions that help you to utilize your input information.

In the above example, the second block will cause the input information to be displayed on the CRT for 2 seconds in a hexadecimal format.

B. INTERRUPTS

The Unidex 16 I/O bus has 4 interrupts which are shared by all I/O modules. On the TCIO board, these 4 interrupts are accessed via the

4 inputs. Each interrupt line has a certain priority, INT4 having the highest priority, INT1 having the lowest.

The interrupt lines are used to interrupt Unidex 16 while it is performing another task. An interrupt is generated when a change in voltage level is detected (as opposed to an input, which detects a voltage level).

When an input is activated, the corresponding interrupt flag will be set and the interrupt line will be held low by the TCIO board until that flag is cleared by Unidex 16 by reading or writing to the appropriate TCIO address (see table below).

TCIO ADDRESS	INTERRUPT FLAG
$790 ----------	Interrupt flag 1
$7A0 ---------	Interrupt flag 2
$7B0----------	Interrupt flag 3
$7C0----------	Interrupt flag 4

Because the interrupt lines are shared by all I/O modules, one of the first tasks performed by Unidex 16 upon receiving an interrupt will be to determine which module generated the interrupt. By reading the Input/Interrupt Status Register, Unidex 16 can determine if the TCIO board generated the interrupt by checking to see if any of the bits are set (see diagram 1).

The Status Register check and the clearing of the interrupt flags must be generated by the user's program.

INTERRUPT FLAG OUTPUT

In addition to the internal functions of the interrupt flags, each flag provides an output and LED indicator as well. The output allows the user to see when an interrupt flag has been set as well as when its been cleared.

The output is available through an optical isolator (Motorola 4N33).

A typical example of an interrupt is as follows:

```
N1 (INT1,4,SUB1)          ; Enable interrupt.  Upon INT1,
                          ; go to subroutine SUB1
.
.
.
N10 (DFS,SUB1             ; Define subroutine SUB1
N11 VAR1=$780             ; Read Input/Interrupt Status
                          ; Register
.
.
N20 $790=H,00             ; Clear interrupt flag #1
                          ; to zero
N21 (MSG,<2>,INT1 -       ; Display message showing INT1 as well as the
VAR1=#H:VAR1)             ; hex value of VAR1
N22 )                     ; End of subroutine
```

In the above program, N1 will arm Unidex 16 to act on a level 1 interrupt, utilizing the #4 option. The #4 option tells Unidex 16 to finish all functions in the present program block before servicing the subroutine and when the subroutine is complete to go back to the next block of the main program.

C. OPTO 22 PAMUX 1

Each Opto 22 Pamux 1 board has 2 (8-bit) bytes of data, each bit corresponding to one I/O module, and is provided with an address switch. An address ranging from $700 to $77F may be selected. When the address switch for one Opto 22 board is set to a certain address, for example $700, the first 8 modules are assigned to this address.

Automatically, the second 8 modules are assigned to the next address (in this case $701).

To select addresses, see the following diagram:

		OPTO 22 PAMUX 1			
OPTO 22 PAMUX 1	ADDRESS SWITCHES	ADDRESS	MODULES	ADDRESS	MODULES
	654321				
A47(A)	CCCCCC	$700	0-7	$701	8-15
A47(B)	CCCCCC	$702	0-7	$703	8-15
.	.	.	.	.	.
.	.	.	.	.	.
	000000	$77E	0-7	$77F	8-15

C = Closd
0 = Open

NOTE: Terminating resistors on the last Pamux 1 board only. Remove all others.

SECTION 13-11 BINARY OUTPUT BOARD OPTION (BO-16)

For the hardware and installation considerations of this option, see the Unidex 16 Hardware Manual.

The Binary Output Board Option is dedicated to providing the actual positions of the axes, on the fly, when requested to do so by the user's host computer.

The Binary Output board also features the following:

- Unidex 16 Versabus compatible
- Compatible with DEC DRV-11J I/O Interface
- Counter information which can be latched and output at any time
- Address selection for output of each axis
- Individual axis reset via Unidex 16 I/O Channel
- Jumper selection for one free-run (rotary) axis
- Remote read-out display option
- For rotary axis, switch select "once per revolution" reset on both the + and - directions
- Remote display interface, a 50 pin connector, can interface to Aerotech's Read-out Module

A. I/O BUS

The axis counters on the board may be reset in unison, individually or in any combination, via the I/O bus. (This feature is remotely accessible.)

Writing to address 6FF in hexadecimal, using the Unidex 16 format $6FF = H,nn, resets the desired axis counters.

NOTE: RESETTING THE BINARY OUTPUT BOARD BY USE OF THE I/O BUS DOES NOT RESET THE AXIS OR ANYTHING PERTAINING TO IT WITHIN THE UNIDEX 16 SYSTEM.

Sending $6FF accesses * * W V U Z Y X

To reset the Binary Output Board via the I/O bus, write:

$6FF=H,nn (nn represents the byte of reset information being sent to the Binary Output Board. A zero is sent to reset the axes counters.) For example:

$6FF=H,12

would send out **0 0 0 1 0 0 1 0** and would reset the W, U, Z and X counters.

NOTE: All of these conditions are true for the remote display as well.

SECTION 13-12 FRONT PANEL OPTO INTERFACE BOARD

For the hardware and installation considerations of this option, see the Unidex 16 Hardware Manual.

The Front Panel Opto Interface Board allows the user to remotely access any ten of the front panel keys.

The front panel opto interface board provides the following features:

- Opto isolation
- Remote access of up to ten Unidex 16 front panel keys
- Isolated power supply
- Matrix jumper selection of remote keys

See the Unidex 16 Hardware Manual for all jumper selections.

SECTION 13-13 LASER FIRING CARD OPTION (LFC-16)

The Laser Firing Card enables a laser to be fired with precision by calculating the vectorial positions of any 1 or 2 axes (X, Y, Z, U, V or W) at any given time. It also outputs the vectorial data rate as an analog signal.

The laser firing option is a function of the I/O bus and is enabled through the parts program or the mdi mode. The program determines:

1. Distance (in machine steps) to be covered before laser fires
2. Laser firing option enabled or disabled
3. Presence or absence of warm-up pulse
4. The 1 or 2 axes selected
5. Data rate analog signal ratio

A. PULSE WIDTH

The pulse width (in μS) is programmable through the I/O channel via commands $6FA = H,nn and $6FB = H,nn, where:

$6FA may range from 00 to FF (MSB)
$6FB may range from 01 to FF (LSB)

B. DATA RATE ANALOG SIGNAL

The data rate analog signal depends on the the data rate from the encoder feedback. A 0 volt (slowest speed) to 9.9 volt (highest speed) analog signal will be output through the same Weidmuller connector. The pin assignments are:

PIN 10 = ANALOG VOLTAGE
PIN 9 = COMMON

To insure your system of getting the maximum analog signal for your particular highest data rate, a "proportional factor", which is an I/O function, must be programmed as follows:

$6F9=H,nn

where "nn" may range from 01 to FF (hexadecimal). How to find your highest data rate and, based on that, your proportional factor, is as follows.

Unidex 16's highest data rate on each axis is 125 KHz. Therefore, the maximum 2 axes vectorial data rate is:

System's highest data rate $= \sqrt{[(125)^2 + (125)^2]}$
$= 176.777$

Based on that information, the proportional factor is found:

Proportional Factor = (System's Highest data rate/176.777) * 255
= 255 (program hex equivalent of H,FF)

Although 125 KHz is the maximum data rate of Unidex 16, there are several other factors that may limit the maximum system data rate. These may include:

1. **Motor-amplifier match**
2. **Encoder limitations**
3. **Mechanical limitations**
4. **Application**

Since this is the case, you may calculate your own system's highest data rate and optimum proportional factor in the same fashion, i.e.

Your system's highest data rate

$$= \sqrt{[(\text{Axis 1 highest data rate})^2 + (\text{Axis 2 highest data rate})^2]}$$

Proportional Factor

$$= (\text{Your system's highest data rate}/176.777) * 255$$
$$= \text{nn}$$

Take the hexadecimal equivalent of "nn" and enter that into your 6F9 = H,nn command.

If, for example, the highest data rate for both axes is 80KHz:

$$\text{Your system's highest data rate} = \sqrt{[(80)^2 + (80)^2]} = 113$$

$$\text{Proportional factor} = (113/176.777) * 255 = 163$$

Program the hexadecimal equivalent of 163, which is H,A3. The laser firing card will now output 9.9V when both axes run at 80KHz., 4.95V when both axes run at 40KHz., etc.

C. OPERATION

The Laser Firing Card can sense either a command clock (stepper motor system) or a feedback clock (DC motor system). In either case, once the desired amount of clock pulses have accumulated in the counter, the laser firing card outputs a pulse, causing the laser to fire.

The programmable clock sources can be 1 axis or 2 axes. If 2 axes, it can be any combination of X, Y, Z, U, V or W.

The laser firing will occur each time the axis or 2-axes combination moves a distance equal to or greater than that which was specified in the laser-firing command. The command can be cancelled by a new laser-firing command or by a command to disable the function.

D. PROGRAMMING THE LASER FIRING CARD

As mentioned previously, the laser firing command is sent via the parts program or the mdi mode. The command follows the I/O format discussed in section 12-2B. To control the laser firing card via the I/O channel, send:

I/O ADDRESS HEX DATA	COMMENTS
$6F9=H,nn	ANALOG RANGE
$6FA=H,nn	PULSE WIDTH MSB
$6FB=H,nn	PULSE WIDTH LSB
$6FC=H,nn	SPACING (DISTANCE) MSB
$6FD=H,nn	SPACING (DISTANCE) LSB
$6FE=H,nn	LASER FIRING CARD FORMAT

All of the I/O channel addresses shown above are fixed. The first address controls the analog signal range. $6F9=H,nn may range from H,01 to H,FF. The second and third addresses control output pulse width, which may range from 1 μS to 65535 μS. The first byte, $6FA = H,nn may range from H,00 to H,FF. The second byte, $6FB = H,nn may range from H,01 to H,FF.

The fourth address, 6FC = H,nn, is the high byte input data. (Bits 7 - 4 are fixed at a value of zero.) This byte may range from H,00 to H,0F.

The fifth address, 6FD = H,nn, is the low byte input data. It may range from H,01 to H,FF.

These two blocks give the distance traveled by the axes before the laser is commanded to fire. The distance must be given in machine steps which have been converted to hex numbers. This information (S) is used in the vectorial distance calculation:

$$S \leq \sqrt{}[X^2 + Y^2]$$

Where:

S = Distance traveled (in machine steps) before laser fires

X = Accumulated clock pulses of axis 1

Y = Accumulated clock pulses of axis 2

The sixth address, $6FE = H,nn, can be broken down as follows:

BIT 7: The warm-up pulse will be sent if this bit is set to 0. Warm-up pulse enabled means that as soon as the $6FE command is sent the laser will fire, regardless of whether the axes have begun to move or not.

1 = disabled 0 = enabled

BIT 6: Laser firing enable. The laser firing enabled (Bit 6 is set to 0) will allow laser firing to occur at the programmed locations when the axes begin to move.

1 = disabled 0 = enabled

(If Bit 7 and 6 are enabled, the warm-up pulse will fire as soon as the command is sent and the normal laser firings will begin when the axes begin to move.)

BIT 5:	W axis:0 = selected, 1 = not selected
BIT 4:	V axis: 0 = selected, 1 = not selected
BIT 3:	U axis: 0 = selected, 1 = not selected
BIT 2:	Z axis: 0 = selected, 1 = not selected
BIT 1:	Y axis: 0 = selected, 1 = not selected
BIT 0:	X axis: 0 = selected, 1 = not selected

The following example will assume a 1:1 resolution, i.e., machine steps and program steps will be of equal value.

EXAMPLE #1

```
N5   (IOFT,BIN,1)         ; Select I/O format
:                         :
N9   $6F9=H,A3            ; Set highest data rate as
                          ; 113000 steps/second
N10 $6FA=H,00             ; High byte pulse width =
                          00 (hex)
N11  $6FB=H,03            ; Low byte pulse width = 03
                          (hex)
N12  $6FC=H,00            ; High byte spacing = 00
                          (hex)
N13  $6FD=H,05            ; Low byte spacing = 05
                          (hex)
                          ; Laser will fire when vectorial
                          ; distance of 5 machine steps
                          ; have been traveled.
N14  $6FE=H,BC            ; Bit 7 6 5 4 3 2 1 0
                          ;     1 0 1 1 1 1 0 0
:
:
N17  G1 X80. Y60. F100.   ; Linear move
:
N40  M30                  ; End of program
```

The above program segment will establish a laser firing command every time the X/Y axes (N14, Bit 1=0 and Bit 0=0) make a vectorial move of 5 machine steps (N13, H,05). It also calls for no warm-up pulse (N14, Bit 7=1) and enables the laser firing card (N14, Bit 6=0).

The vectorial move would be determined as follows:

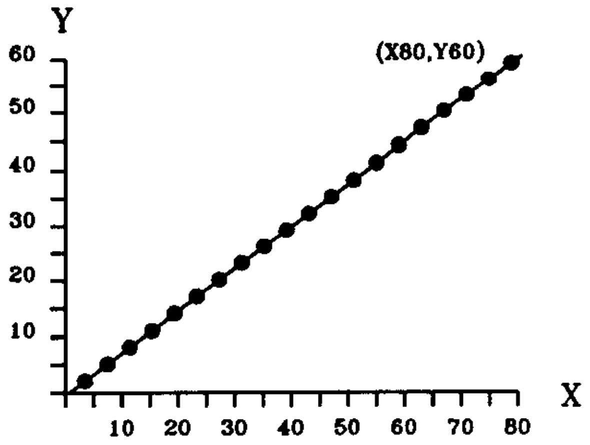
$$S \leq \sqrt{[X^2 + Y^2]}$$

The two axes' counters' (X and Y in this example) accumulate steps proportional to what has been programmed. These steps are incorporated into the above formula. When the sum of each axis' steps squared is equal to or greater than the steps to travel before laser firing (S) squared, the laser fires.

For example, the machine distance to travel before the laser fires was programmed as 5 in the above program. The proportional XY moves (reduced from X80, Y60) that will satisfy the above formula when the laser is to fire every 5 machine steps are X4, Y3. Therefore,

$$S \leq \sqrt{[X^2 + Y^2]}$$

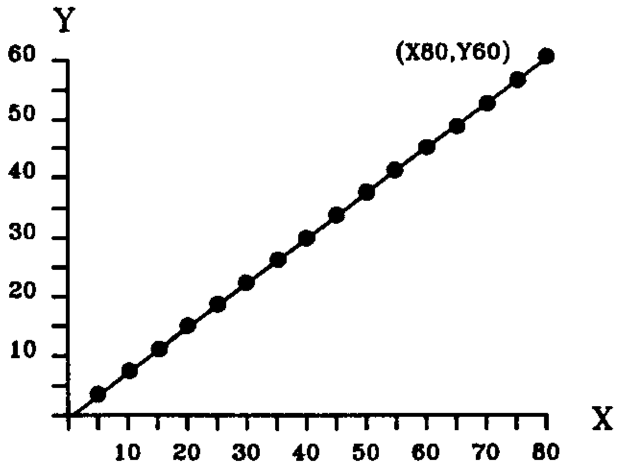
The sample programming segment would cause the following move. The laser firing occurs here every 5 machine steps, along the X/Y vector.



You may also lock laser firing onto one axis. This would change the distance between laser firing. For example, if everything was the same in the sample program except:

N14 $6FE=H,BE

only the X axis would be enabled for laser firing and the previous graph would now look like this:



The vectorial distance traveled is still X80, Y60. However, since the laser firing is locked on the X axis, the laser fires when 5 machine steps along the X axis have been traveled, rather than along the X/Y vector as shown previously. How this changes the vectorial distance between laser firings may be determined as follows.

The formula $S \leq \sqrt{}\,[X^2 + Y^2]$ will demonstrate how many machine steps along the vectorial path will be traveled before the laser fires. To find the appropriate numbers to plug into this formula, divide the X distance to be traveled before the laser fires by the total X distance (in this case 5/80). Then find the proportional Y distance to be traveled before the laser fires, i.e.,

$$Y/60 = 5/80$$
$$Y = 5/80 * 60$$
$$Y = 5/4 * 3$$
$$Y = 3.75$$

Since the distance is in whole steps only, the above result must be rounded off. Therefore, Y equals 4, and:

$$S \leq \sqrt{}\,[X^2 + Y^2]$$
$$41 \leq 5^2 + 4^2$$
$$\leq 41 = 6.4$$

The distance between laser firings is now 6.4 machine steps.

The following example demonstrates the same program, only this time the laser firing is locked onto the Y axis.

The numbers for the $S \leq \sqrt{}\,[X^2 + Y^2]$ formula are found by dividing the Y distance to be traveled before the laser fires by the total Y distance (in this case, 5/60). Then find the proportional X distance, i.e.,

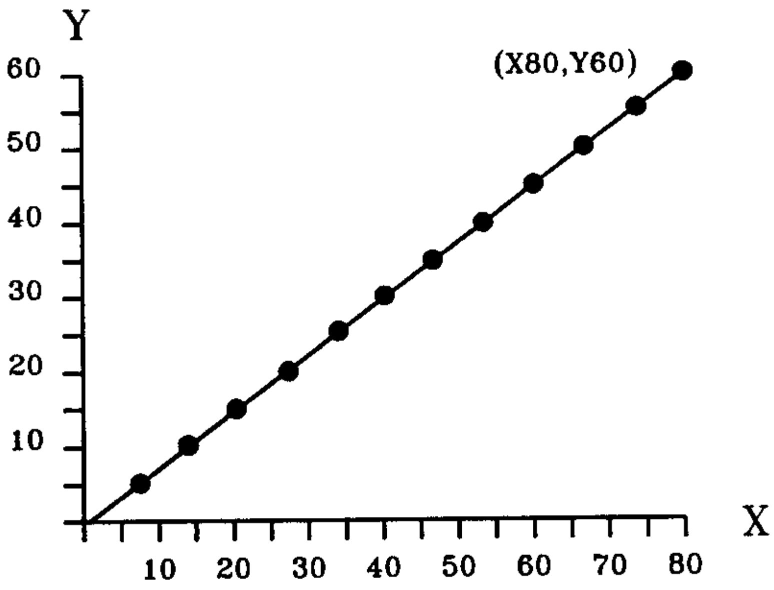
$$X/80 = 5/60$$
$$X = 5/60 * 80$$

$$X = 5/3 * 4$$
$$X = 6.667$$

Therefore,

$$S \leq \sqrt{[X^2 + Y^2]}$$
$$74 \leq 7^2 + 5^2$$
$$\leq 74 = 8.6$$

The laser firing now occurs every 8.6 machine steps along the X/Y vector.

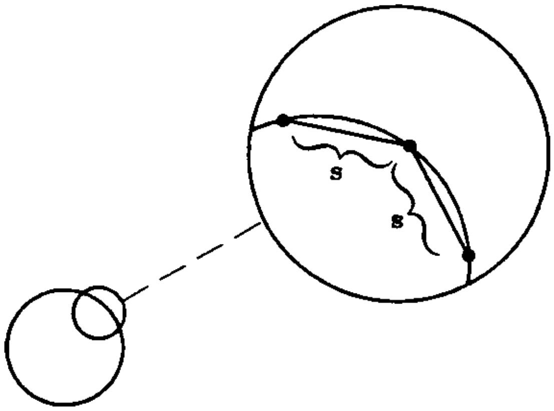


NOTE: **Remember that when making a vectorial move, if the 2 axes involved are not the same resolution, compensations will have to be made before programming.**

E. CIRCULAR INTERPOLATION WITH LASER FIRING

When programming the laser firing card during circular contouring, "S" will be the linear distance in machine steps between output pulses, as shown in the following illustration.

NOTE: Circular interpolation is not allowed between 2 axes of different resolutions.



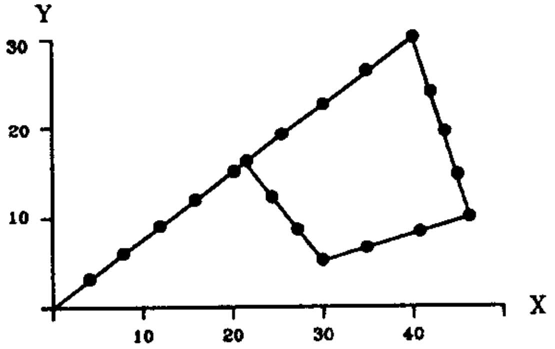
Following in another example of laser firing:

EXAMPLE #2

```
(IOFT,BIN,1)            $6FC=H,00
$6FD=H,05               ; Output C=5 to Laser Firing
                        Card
:                       :
$6FE=H,BC               ; Turn on Laser Firing Card.
                        Enable X/Y axes counts

G1 X40. Y30. F100.
G1 X5. Y-20. F30.
G1 X-15. Y-5. F30.
```

```
G1 X-5. Y14. F30.
$6FE=H,FF                              ; Turn off Laser Firing Card
```



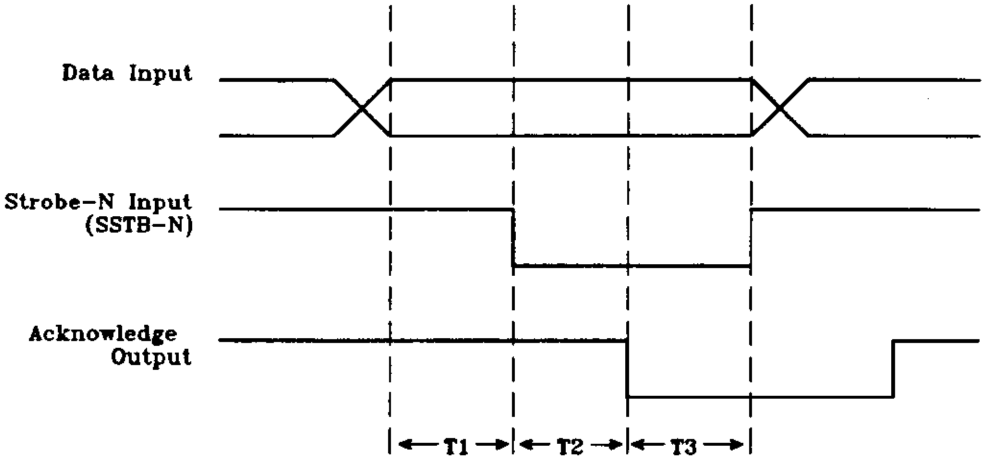
NOTE: A Continuous Laser Firing Option is available. Information will be provided with the purchase of the option.

SECTION 13-14 SPINDLE FUNCTION D/A CARD (SD-A)

The purpose of the Programmable S-function D/A converter card is to enable a digital S-function output from Unidex 16 to be converted to an analog signal. The analog signal is used to control either a unipolar or bipolar spindle, or other type of device requiring an analog output signal.

A. TIMING

The timing diagram below illustrates the timing cycle utilized by Unidex 16 when sending data to the S-function D/A card.



T1 Data to SSTB-N Low — 1mS typical, determined by Unidex 16 parameter #402 (00001)

T2 SSTB-N Low to ACK-N Low — 250 µS typical, determined by "S" D/A board debounce circuit M1 and C17

T3 ACK-N Low to Strobe-N Hi — 1 mS typical, determined by Unidex 16 parameter #410 (0 0 0 0 1)

NOTE: **Standard Unidex 16 default value for parameters 402 and 410 is 00010 for 10mS. Although 10mS for T1 and T2 will operate properly with the "S" D/A board, it is not optimum timing.**

Appendix 1

ASCII Character Set

APPENDIX 1: ASCII CHARACTER SET

ASCII CODE	HEX CODE FROM PORTS A OR B	UNIDEX 16	HEX CODE FROM KEYBOARD
NUL	00	RESERVE	RESERVE
SOH	01	SOFTKEY 1	76H
STX	02	SOFTKEY 2	B6H
ETX	03	SOFTKEY 3	D0H
EOT	04	SOFTKEY 4	50H
ENQ	05	SOFTKEY 5	E4H
ACK	06	SOFTKEY 6	64H
BEL	07	RESET	RESET
BS	08	CUSOR LEFT	B2H
HT	09	END FILE	RESERVE
LF	0A	LINEFEED	RESERVE
VT	0B	REMOTE ON	RESERVE
FF	0C	CURSOR RIGHT	94H
CR	0D	ENTER	52H
SO	0E	HOME	80H
SI	0F	REMOTE OFF	RESERVE
DLE	10	DEL CHAR	1CH
DC1	11	X-ON	RESERVE
DC2	12	BACK TAB	86H
DC3	13	X-OFF	RESERVE
DC4	14	END FILE	RESERVE
NAK	15	SHFT CLR LN	29H
SYN	16	RESERVE	RESERVE
ETB	17	ROLL UP	C8H
CAN	18	SHFT NXT PG	F1H
EM	19	SHFT PRV PG	3AH
SUB	1A	TAB	BEH
ESC	1B	RESERVE	RESERVE
FS	1C	CYC START	C2H
GS	1D	NEXT PAGE	DCH
RS	1E	PREV PAGE	10H
US	1F	SHFT CYCL ST	E9H
SPACE	20	SPACE	32H
!	21	!	FEH
"	22	"	81H
#	23	#	01H
$	24	$	46H
%	25	%	26H
&	26	&	C6H
'	27	'	AEH
(	28	(	EEH
)	29	)	6EH
*	2A	*	A6H
+	2B	+	1EH
,	2C	,	0AH
-	2D	-	BCH
.	2E	.	7CH
/	2F	/	E0H
0	30	0	30H
1	31	1	D4H
2	32	2	B4H
3	33	3	34H
4	34	4	70H
5	35	5	08H
6	36	6	F0H
7	37	7	84H
8	38	8	C4H
9	39	9	44H
:	3A	:	88H
;	3B	;	24H
<	3C	<	25H
=	3D	=	89H
>	3E	>	49H
?	3F	?	DAH
@	40	RESERVE	RESERVE
A	41	A	92H
B	42	B	A4H
C	43	C	48H
D	44	D	74H
E	45	E	FCH
F	46	F	38H
G	47	G	CAH
H	48	H	90H
I	49	I	18H
J	4A	J	58H
K	4B	K	20H
L	4C	L	22H
M	4D	M	96H
N	4E	N	B0H
O	4F	O	54H
P	50	P	3CH
Q	51	Q	E8H
R	52	R	72H
S	53	S	A0H
T	54	T	60H
U	55	U	D6H
V	56	V	A2H
W	57	W	62H
X	58	X	E2H
Y	59	Y	6AH
Z	5A	Z	F2H
[	5B	RESERVE	RESERVE
\	5C	SHFT DEL CHR	31H
]	5D	RESERVE	RESERVE
^	5E	CURSOR UP	66H
_	5F	CURSOR DOWN	A6H
`	60	RESERVE	RESERVE
a	61	a	B9H
b	62	b	9EH
c	63	c	66H
d	64	d	41H
e	65	e	C9H
f	66	f	0EH
g	67	g	E5H
h	68	h	BAH
i	69	i	36H
j	6A	j	76H
k	6B	k	1AH
l	6C	l	19H
m	6D	m	B6H
n	6E	n	56H
o	6F	o	7EH
p	70	p	09H
q	71	q	D6H
r	72	r	45H
s	73	s	9AH
t	74	t	5AH
u	75	u	F6H
v	76	v	99H
w	77	w	59H
x	78	x	D9H
y	79	y	A5H
z	7A	z	C5H
{	7B	ROLL DN	F6H
|	7C	RECALL	28H
}	7D	INSERT	9CH
~	7E	FEEDHOLD	82H
DEL	7F	CLR LINE	02H

Appendix 2

Hexadecimal Numbers
and
Equivalents

DECIMAL	BINARY	HEX	DECIMAL	BINARY	HEX	DECIMAL	BINARY	HEX	DECIMAL	BINARY	HEX	DECIMAL	BINARY	HEX
0	00000000	00	52	00110100	34	104	01101000	68	156	10011100	9C	208	11010000	D0
1	00000001	01	53	00110101	35	105	01101001	69	157	10011101	9D	209	11010001	D1
2	00000010	02	54	00110110	36	106	01101010	6A	158	10011110	9E	210	11010010	D2
3	00000011	03	55	00110111	37	107	01101011	6B	159	10011111	9F	211	11010011	D3
4	00000100	04	56	00111000	38	108	01101100	6C	160	10100000	A0	212	11010100	D4
5	00000101	05	57	00111001	39	109	01101101	6D	161	10100001	A1	213	11010101	D5
6	00000110	06	58	00111010	3A	110	01101110	6E	162	10100010	A2	214	11010110	D6
7	00000111	07	59	00111011	3B	111	01101111	6F	163	10100011	A3	215	11010111	D7
8	00001000	08	60	00111100	3C	112	01110000	70	164	10100100	A4	216	11011000	D8
9	00001001	09	61	00111101	3D	113	01110001	71	165	10100101	A5	217	11011001	D9
10	00001010	0A	62	00111110	3E	114	01110010	72	166	10100110	A6	218	11011010	DA
11	00001011	0B	63	00111111	3F	115	01110011	73	167	10100111	A7	219	11011011	DB
12	00001100	0C	64	01000000	40	116	01110100	74	168	10101000	A8	220	11011100	DC
13	00001101	0D	65	01000001	41	117	01110101	75	169	10101001	A9	221	11011101	DD
14	00001110	0E	66	01000010	42	118	01110110	76	170	10101010	AA	222	11011110	DE
15	00001111	0F	67	01000011	43	119	01110111	77	171	10101011	AB	223	11011111	DF
16	00010000	10	68	01000100	44	120	01111000	78	172	10101100	AC	224	11100000	E0
17	00010001	11	69	01000101	45	121	01111001	79	173	10101101	AD	225	11100001	E1
18	00010010	12	70	01000110	46	122	01111010	7A	174	10101110	AE	226	11100010	E2
19	00010011	13	71	01000111	47	123	01111011	7B	175	10101111	AF	227	11100011	E3
20	00010100	14	72	01001000	48	124	01111100	7C	176	10110000	B0	228	11100100	E4
21	00010101	15	73	01001001	49	125	01111101	7D	177	10110001	B1	229	11100101	E5
22	00010110	16	74	01001010	4A	126	01111110	7E	178	10110010	B2	230	11100110	E6
23	00010111	17	75	01001011	4B	127	01111111	7F	179	10110011	B3	231	11100111	E7
24	00011000	18	76	01001100	4C	128	10000000	80	180	10110100	B4	232	11101000	E8
25	00011001	19	77	01001101	4D	129	10000001	81	181	10110101	B5	233	11101001	E9
26	00011010	1A	78	01001110	4E	130	10000010	82	182	10110110	B6	234	11101010	EA
27	00011011	1B	79	01001111	4F	131	10000011	83	183	10110111	B7	235	11101011	EB
28	00011100	1C	80	01010000	50	132	10000100	84	184	10111000	B8	236	11101100	EC
29	00011101	1D	81	01010001	51	133	10000101	85	185	10111001	B9	237	11101101	ED
30	00011110	1E	82	01010010	52	134	10000110	86	186	10111010	BA	238	11101110	EE
31	00011111	1F	83	01010011	53	135	10000111	87	187	10111011	BB	239	11101111	EF
32	00100000	20	84	01010100	54	136	10001000	88	188	10111100	BC	240	11110000	F0
33	00100001	21	85	01010101	55	137	10001001	89	189	10111101	BD	241	11110001	F1
34	00100010	22	86	01010110	56	138	10001010	8A	190	10111110	BE	242	11110010	F2
35	00100011	23	87	01010111	57	139	10001011	8B	191	10111111	BF	243	11110011	F3
36	00100100	24	88	01011000	58	140	10001100	8C	192	11000000	C0	244	11110100	F4
37	00100101	25	89	01011001	59	141	10001101	8D	193	11000001	C1	245	11110101	F5
38	00100110	26	90	01011010	5A	142	10001110	8E	194	11000010	C2	246	11110110	F6
39	00100111	27	91	01011011	5B	143	10001111	8F	195	11000011	C3	247	11110111	F7
40	00101000	28	92	01011100	5C	144	10010000	90	196	11000100	C4	248	11111000	F8
41	00101001	29	93	01011101	5D	145	10010001	91	197	11000101	C5	249	11111001	F9
42	00101010	2A	94	01011110	5E	146	10010010	92	198	11000110	C6	250	11111010	FA
43	00101011	2B	95	01011111	5F	147	10010011	93	199	11000111	C7	251	11111011	FB
44	00101100	2C	96	01100000	60	148	10010100	94	200	11001000	C8	252	11111100	FC
45	00101101	2D	97	01100001	61	149	10010101	95	201	11001001	C9	253	11111101	FD
46	00101110	2E	98	01100010	62	150	10010110	96	202	11001010	CA	254	11111110	FE
47	00101111	2F	99	01100011	63	151	10010111	97	203	11001011	CB	255	11111111	FF
48	00110000	30	100	01100100	64	152	10011000	98	204	11001100	CC			
49	00110001	31	101	01100101	65	153	10011001	99	205	11001101	CD			
50	00110010	32	102	01100110	66	154	10011010	9A	206	11001110	CE			
51	00110011	33	103	01100111	67	155	10011011	9B	207	11001111	CF			

APPENDIX 3

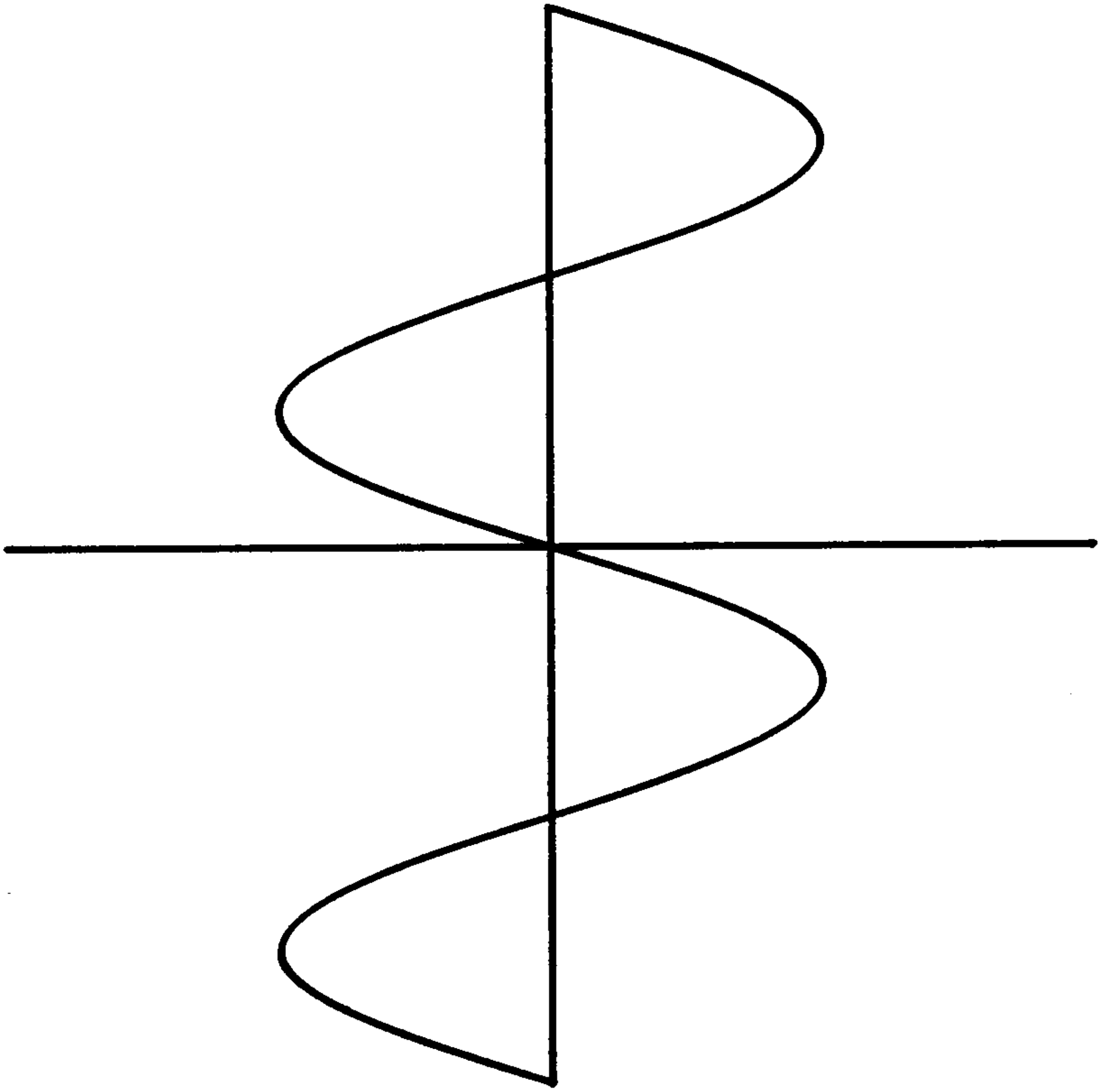
Sample Programs

SAMPLE 1

file name: SINE
file type: .PP
length (bytes): 1385
last edit date: 2-1-85

```
%SINE DRAWING PROGRAM
(DVAR,ANG1,VARA,VARI,VARP,VAR1,VAR2,VAR3,VAR4,VAR5)
(REF,X,Y,Z)                      ; GO HOME
G24                              ; NON-CORNER ROUNDING
G0 X12.0 Y-8.0                   ; HOME OFFSET
G0 Z-5.0                         ; Z OFFSET
G92                              ; RESET POSITION REGISTERS
G1 F200.0                        ; LINEAR MODE  200 IPM
G91                              ; INCREMENTAL MODE
Z1.0                             ; PEN UP
X-3.0 Y0                         ;
Z-1.0                            ;
X6.0 Y0                          ;
Z1.0                             ; DRAW X AND Y AXES
X-3.0 Y-3.0                      ;
Z-1.0                            ;
X0 Y6.0                          ;
Z1.0                             ;
VARA=1.5                         ; AMPLITUDE
VARI=1.0                         ; X INCREMENT
VARP=1.0                         ; PERIOD
ANG1=-360                        ; STARTING ANGLE
X-3.0 Y-3.0                      ; STARTING POINT
Z-1.0                            ; PEN DOWN
G23 F35.0                        ;
VAR3=0                           ;
VAR1=INT(10000*VARA*SIN(VARP*RAD(ANG1)))   ; Y START
VAR4=INT(30000*VAR3)/360)        ; X START
(DENT,ENT1)                      ;
ANG1=ANG1+VARI                   ; INCREMENT ANGLE
VAR3=VAR3+VARI                   ; INCREMENT X
VAR5=INT((30000*VAR3)/360)       ; X FINISH
VAR2=INT(10000*VARA*SIN(VARP*RAD(ANG1)))   ; Y FINISH
```

```
X = (VAR5-VAR4)/10000            ; XY MOVE
Y = (VAR2-VAR1)/10000
VAR1 = VAR2  VAR4 = VAR5         ; INCREMENT Y POSITION
(JUMP,ENT1,ANG1.LT.360)          ; KEEP GOING UNTIL ANG .GE 2 PI
(REF,Z)                          ; Z HOME
M30
```

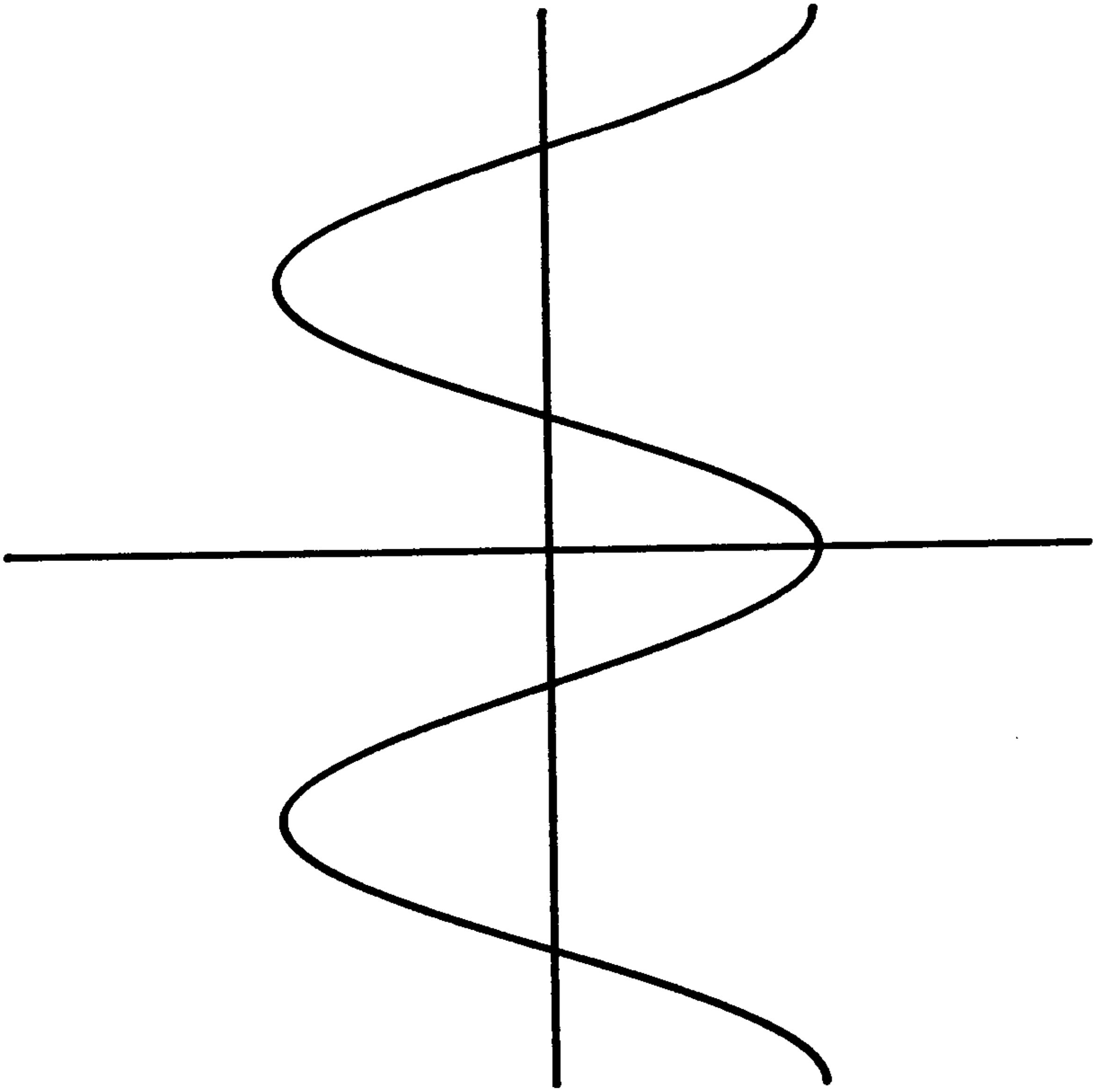


SAMPLE 2

file name : COSINE
file type : .PP
length (bytes) : 1354
last edit date : 2-1-85

```
%COSINE DRAWING PROGRAM
(DVAR,ANG1,VARA,VARI,VARP,VAR1,VAR2,VAR3,VAR4,VAR5)
(REF,X,Y,Z)                          ; GO HOME
G24                                  ; NON-CORNER ROUNDING
G0 X12.0 Y-8.0                       ; HOME OFFSET
G0 Z-5.0                             ; Z OFFSET
G92                                  ; RESET POSITION REGISTERS
G1 F200.0                            ; LINEAR MODE  200 IMP
G91                                  ; INCREMENTAL MODE
Z1.0                                 ; PEN UP
X-3.0 Y0                             ;
Z-1.0                                ;
X6.0 Y0                              ;
Z1.0                                 ; DRAW X AND Y AXES
X-3.0 Y-3.0                          ;
Z-1.0                                ;
X0 Y6.0                              ;
Z1.0                                 ;
VARA=1.5                             ; AMPLITUDE
VARI=1.0                             ; X INCREMENT
VARP=1.0                             ; PERIOD
ANG1=-360                            ; STARTING ANGLE
X-3.0 Y-1.5                          ;
Z-1.0                                ; PEN DOWN
G23 F35.0                            ;
VAR3=0                               ;
VAR1=INT(10000*VARA*COS(VARP*RAD(ANG1)))  ; Y START
VAR4=INT(30000*VAR3)/360)            ; X START
(DENT,ENT1)                          ;
ANG1=ANG1+VARI                       ; INCREMENT ANGLE
VAR3=VAR3+VARI                       ; INCREMENT X
VAR2=INT(10000*VARA*COS(VARP*RAD(ANG1)))  ; Y FINISH
VAR5=INT((30000*VAR3)/360)           ; X FINISH
```

```
X = (VAR5-VAR4)/10000 Y = (VAR2-VAR1)/10000        ; XY MOVE
VAR1 = VAR2 VAR4 = VAR5              ; INCREMENT XY POSITION
(JUMP,ENT1,ANG1.LT.360)              ; KEEP GOING UNTIL ANG .GE 2PI
(REF,Z)                              ; Z HOME
M30
```

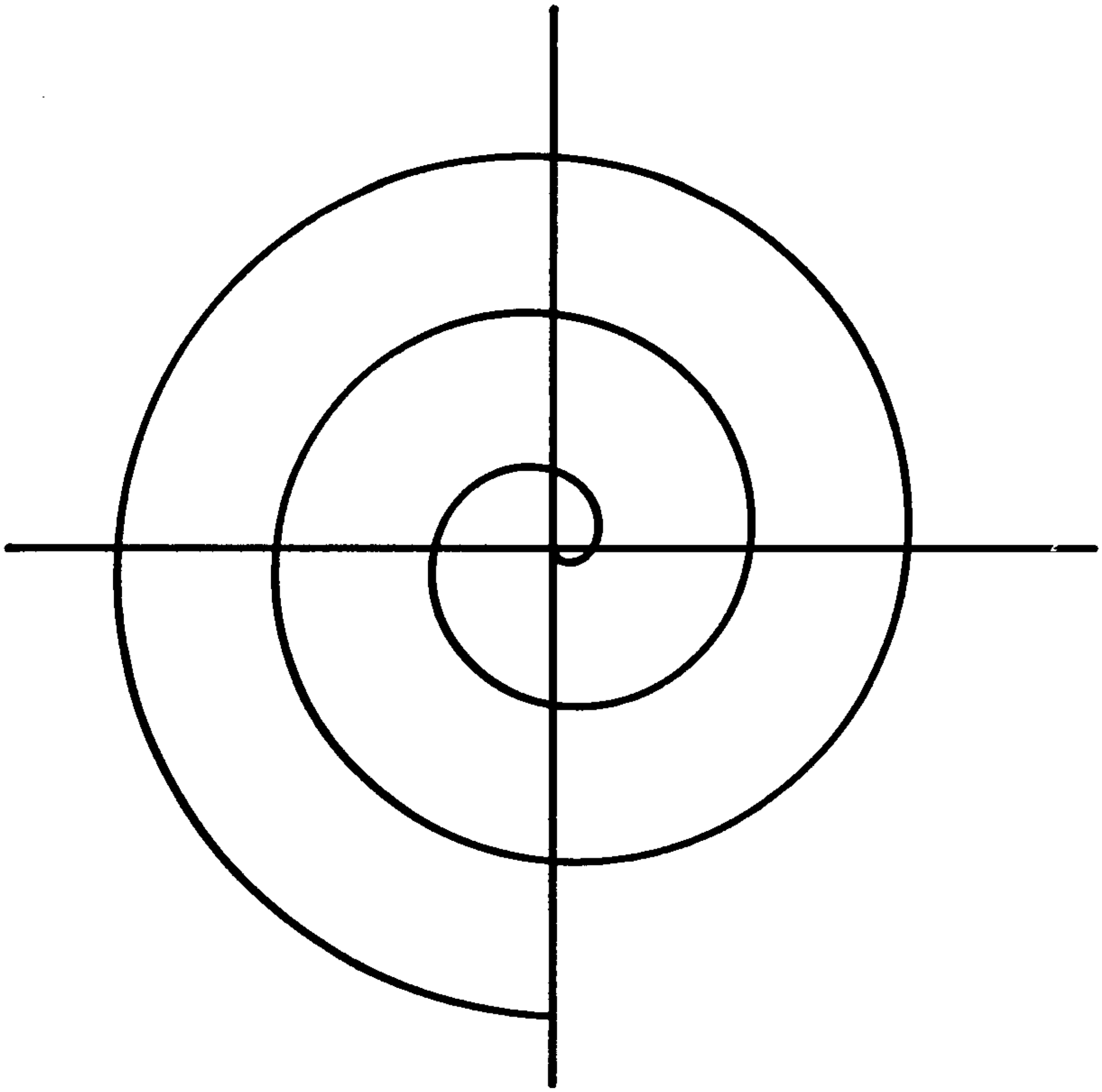


SAMPLE 3

file name : SPIRAL
file type : .PP
length (bytes) : 1384
last edit date : 3-1-85

```
%SPIRAL DRAWING PROGRAM
(DVAR,ANG1,VAR1,VAR2,VAR3,VAR4,VARI,VARC,VARS)
(REF,X,Y,Z)                        ; GO HOME
G24                                ; NON-CORNER ROUNDING
G0 X12.0 Y-8.0                     ; HOME OFFSET
G0 Z-5.0                           ; Z OFFSET
G92                                ; RESET POSITION REGISTERS
G1 F200.0                          ; LINEAR MODE 200 IPM
G91                                ; INCREMENTAL MODE
Z1.0                               ; PEN UP
X-3.0 Y0                           ;
Z-1.0                              ;
X6.0 Y0                            ;
Z1.0                               ; DRAW X AND Y AXES
X-3.0 Y-3.0                        ;
Z-1.0                              ;
X0 Y6.0                            ;
Z1.0                               ;
Y-3.0                              ; STARTING POINT
ANG1=0                             ; STARTING ANGLE
VARC=3                             ; NUMBER OF CYCLES
VARI=1.0                           ; INCREMENT OF ANG1
VARS=50.0                          ; SCALING FACTOR
Z-1.0                              ; PEN DOWN
G23 F50.0                          ;
VAR1=INT(10000*(VARS*RAD(ANG1/360)*COS(RAD(ANG1))))
VAR2=INT(10000*(VARS*RAD(ANG1/360)*SIN(RAD(ANG1))))
(DENT,ENT1)
ANG1=ANG1+VARI                     ; INCREMENT ANGLE
VAR3=INT(10000*(VARS*RAD(ANG1/360)*COS(RAD(ANG1))))
VAR4=INT(10000*(VARS*RAD(ANG1/360)*SIN(RAD(ANG1))))
X=(VAR3-VAR1)/10000 Y=(VAR4-VAR2)/10000       ; XY MOVE
VAR1=VAR3 VAR2=VAR4                ; INCREMENT POSITION
(JUMP,ENT1,ANG1.LT.(VARC*360))     ; KEEP GOING UNTIL 2 CYCLES
```

```
(REF,Z)                                  ; Z HOME
M30
```

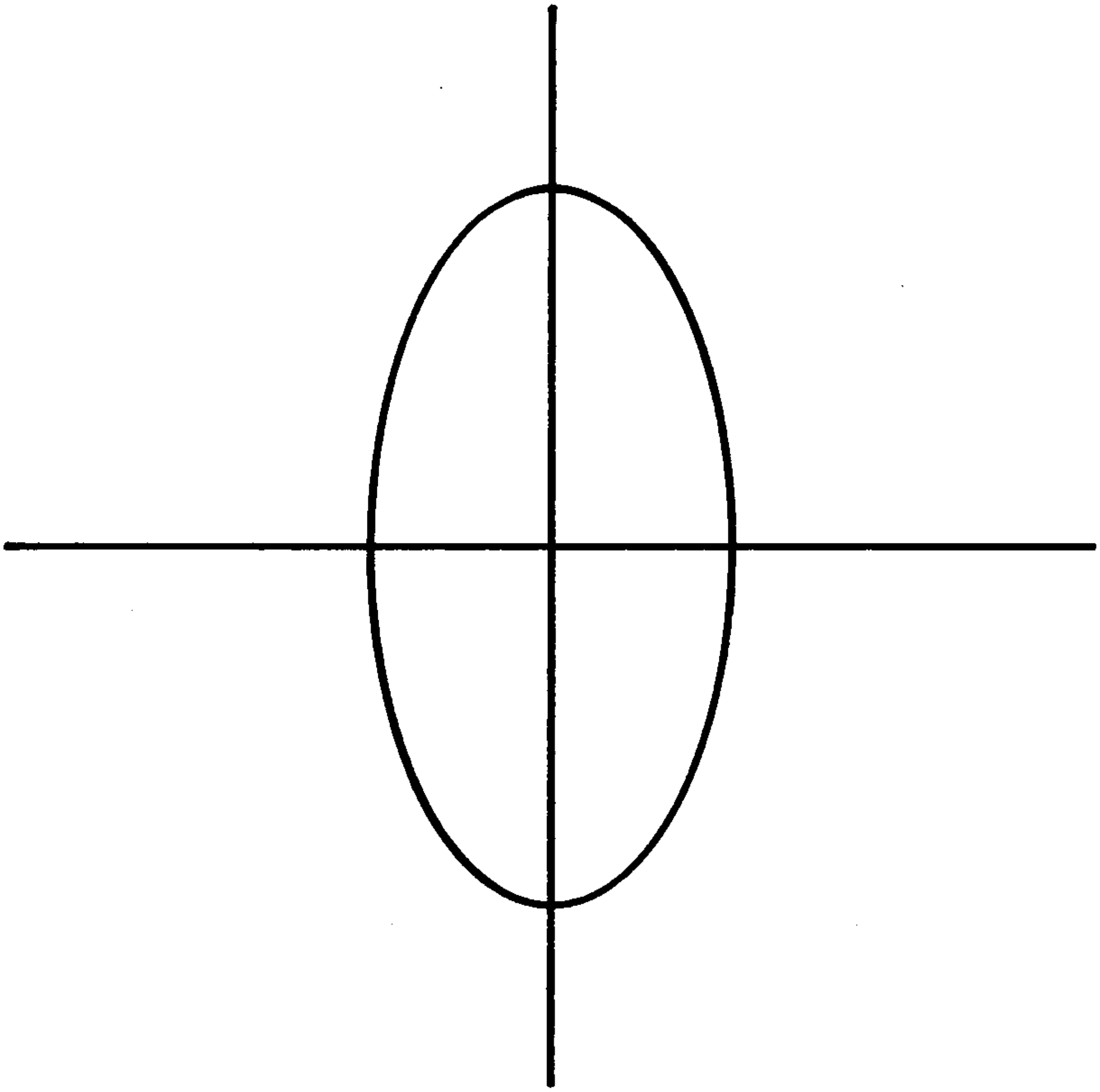


SAMPLE 4

file name : ELLIPSE
file type : .PP
length (bytes) : 1217
last edit date : 2-1-85

```
%ELLIPSE DRAWING PROGRAM
(DVAR,ANG1,VARR,VAR1,VAR2,VAR3,VAR4,VARA,VARB)
(REF,X,Y,Z)                             ; GO HOME
G24                                     ; NON-CORNER ROUNDING
G0 X12.0 Y-8.0                          ; HOME OFFSET
G0 Z-5.0                                ; Z OFFSET
G92                                     ; RESET POSITION REGISTERS
G1 F200.0                               ; LINEAR MODE  200 IPM
G91 G23                                 ; INCREMENTAL MODE
Z1.0                                    ; PEN UP
X-3.0 Y0                                ;
Z-1.0                                   ;
X6.0 Y0                                 ;
Z1.0                                    ; DRAW X AND Y AXES
X-3.0 Y-3.0                             ;
Z-1.0                                   ;
X0 Y6.0                                 ;
Z1.0                                    ;
X2.0 Y-3.0                              ; STARTING POINT
Z-1.0                                   ; PEN DOWN
G23 F35.0                               ;
VARA = 2.0                              ; RADIUS ON X AXIS
VARB = 1.0                              ; RADIUS ON Y AXIS
ANG1 = 0                                ; STARTING ANGLE
VAR1 = INT(10000*VARA*(COS(RAD(ANG1))))           ; Y START
VAR2 = INT(10000*VARB*(SIN(RAD(ANG1))))           ; X START
(DENT,ENT1)
ANG1 = ANG1 + 1
VAR3 = INT(10000*VARA*(COS(RAD(ANG1))))           ; Y FINISH
VAR4 = INT(10000*VARB*(SIN(RAD(ANG1))))           ; X FINISH
X = (VAR3-VAR1)/10000
Y = (VAR4-VAR2)/10000
VAR1 = VAR3 VAR2 = VAR4
(JUMP,ENT1,ANG1.LT.360)
```

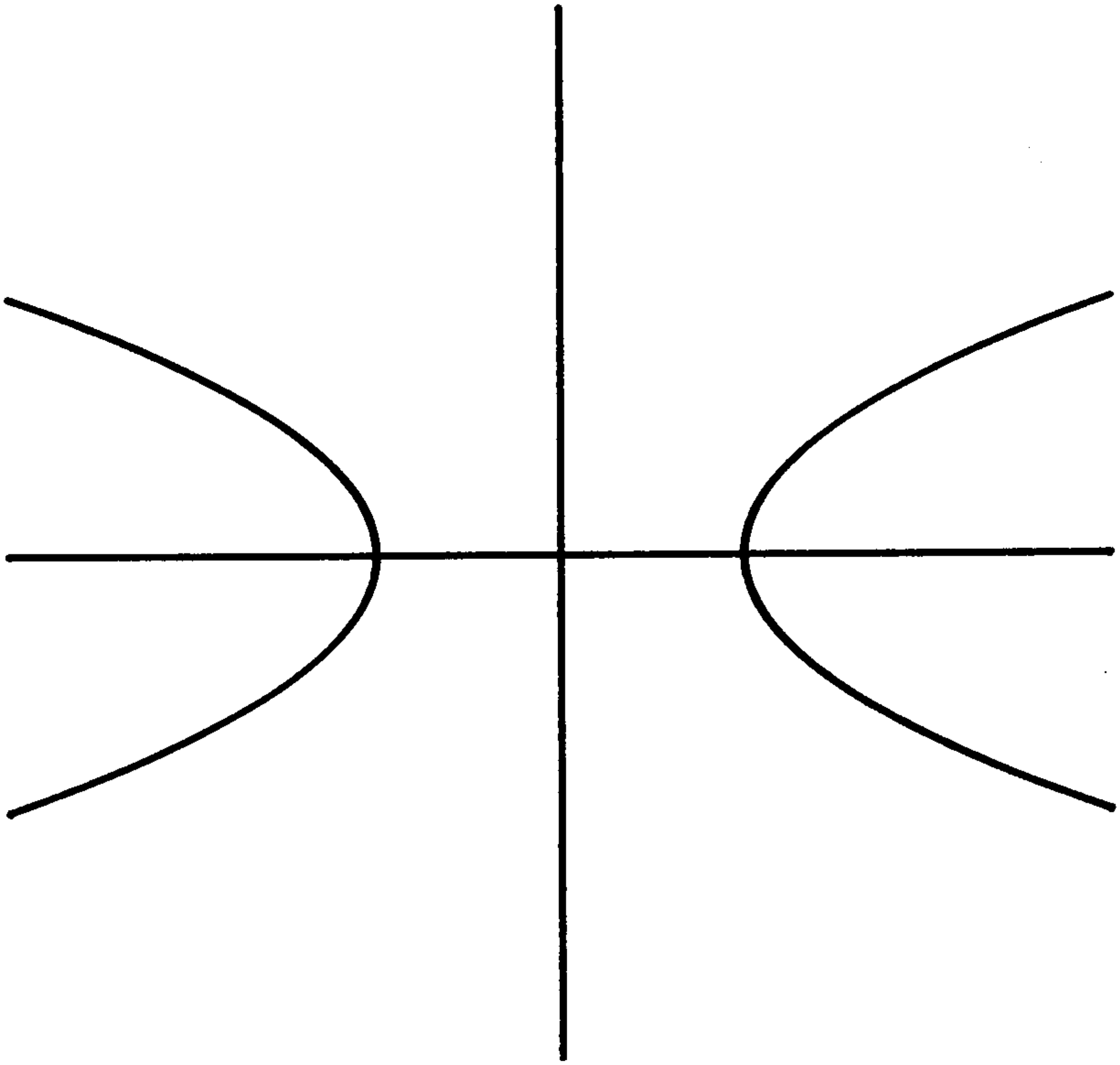
```
(REF,Z)                          ; Z HOME
M30
```



SAMPLE 5

file name : PARABO
file type : .PP
length (bytes) : 1027
last edit date : 2-1-85

```
%PARABOLA DRAWING PROGRAM
(DVAR,VAR1,VAR2)
(REF,X,Y,Z)                    ; GO HOME
G24                            ; NON-CORNER ROUNDING
G0 X12.0 Y-8.0                 ; HOME OFFSET
G0 Z-5.0                       ; Z OFFSET
G92                            ; RESET POSITION REGISTERS
G1 F200.0                      ; LINEAR MODE 200 IPM
G91                            ; INCREMENTAL MODE
Z1.0                           ; PEN UP
X-3.0 Y0                       ;
Z-1.0                          ;
X6.0 Y0                        ;
Z1.0                           ; DRAW X AND Y AXES
X-3.0 Y-3.0                    ;
Z-1.0                          ;
X0 Y6.0                        ;
Z1.0                           ;
VAR1 = 1.0
VAR2 = 1.0
G90 X = -1.4142 Y = 3.0
G91
(CLS,SUB1,VAR1,VAR2)
G90 X = -1.4142 Y = -3.0
G91
(CLS,SUB1,-VAR1,-VAR2)
G90 X0 Y0
(REF,Z)
M2
(DFS,SUB1,VAR3,VAR4,XPOS,SVR1,SVR2,VARI
XPOS = -1.4142
VARI = 0.01
Z-1.0
G23 F50.0
SVR1=INT(10000*((VAR3*(XPOS*XPOS))+VAR4))
(DENT,SENT)
XPOS=XPOS+VARI
SVR2=INT(10000*((VAR3*(XPOS*XPOS))+VAR4))
X=VARI Y=(SVR2-SVR1)/10000
SVR1=SVR2
(JUMP,SENT,XPOS.LT.1.4142))
G24 F200.0
Z1.0
)
M30
```



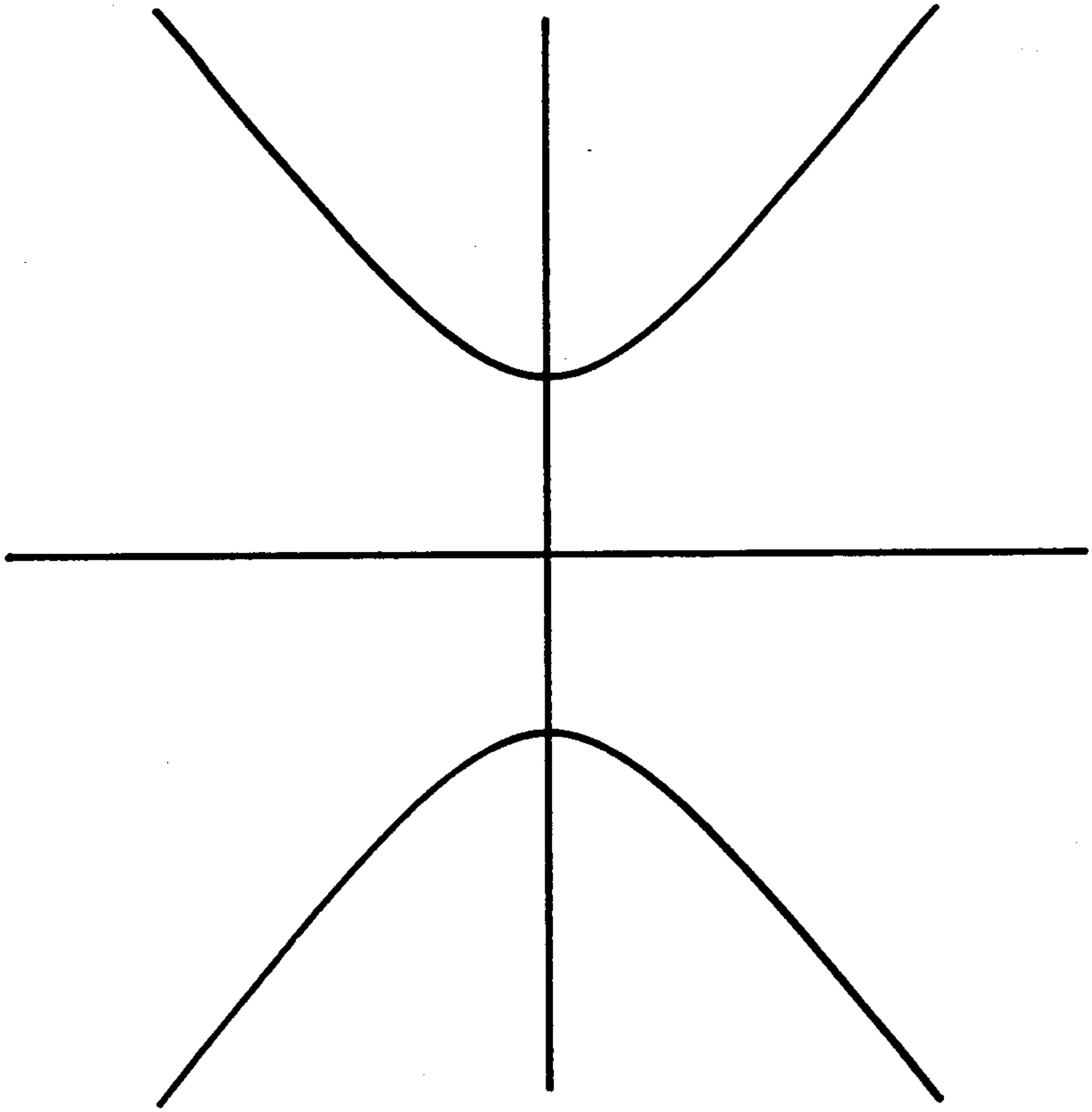
SAMPLE 6

file name : HYPER
file type : .PP
length (bytes) : 1220
last edit date : 2-1-85

```
%HYPERBOLE DRAWING PROGRAM
(DVAR,VAR1,VAR2)
(REF,X,Y,Z)
G24
G0 X12.0 Y-8.0
G0 Z-5.0
G92
G1 F200.0
G91
Z1.0
X-3.0 Y0
Z-1.0
X6.0 Y0
Z1.0
X-3.0 Y-3.0
Z-1.0
X0 Y6.0
Z1.0
VAR1=1.0
VAR2=0.75
G90 X=VAR1 Y0
G91
(CLS,SUB1,VAR1,VAR2)
G90 X=VAR1 Y0
G91
(CLS,SUB1,VAR1,-VAR2)
G90 X=-VAR1 Y0
G91
(CLS,SUB1,-VAR1,VAR2)
G90 X=-VAR1 Y0
G91
(CLS,SUB1,-VAR1,VAR2)
G90 X0 Y0
```

```
(REF,Z)
M2

(DFS,SUB1,VAR3,VAR4,ANG1,SVR1,SVR2,SVR3,SVR4,VARI
ANG1 = 0
VARI = 1.0
Z-1.0
G23 F50.0
SVR1 = INT(10000*(VAR3/COS(RAD(ANG1))))
SVR2 = INT(10000*(VAR4*(SIN(RAD(ANG1))/COS(RAD(ANG1)))))
(DENT,SENT)
ANG1 = ANG1 + VARI
SVR3 = INT(10000*(VAR3/COS(RAD(ANG1))))
SVR4 = INT(10000*(VAR4*(SIN(RAD(ANG1))/COS(RAD(ANG1)))))
X = (SVR3-SVR1)/10000
Y = (SVR4-SVR2)/10000
SVR1 = SVR3 SVR2 = SVR4
(JUMP,SENT,ANG1.LT.71)
G24 F200.0
Z1.0
)
M30
```



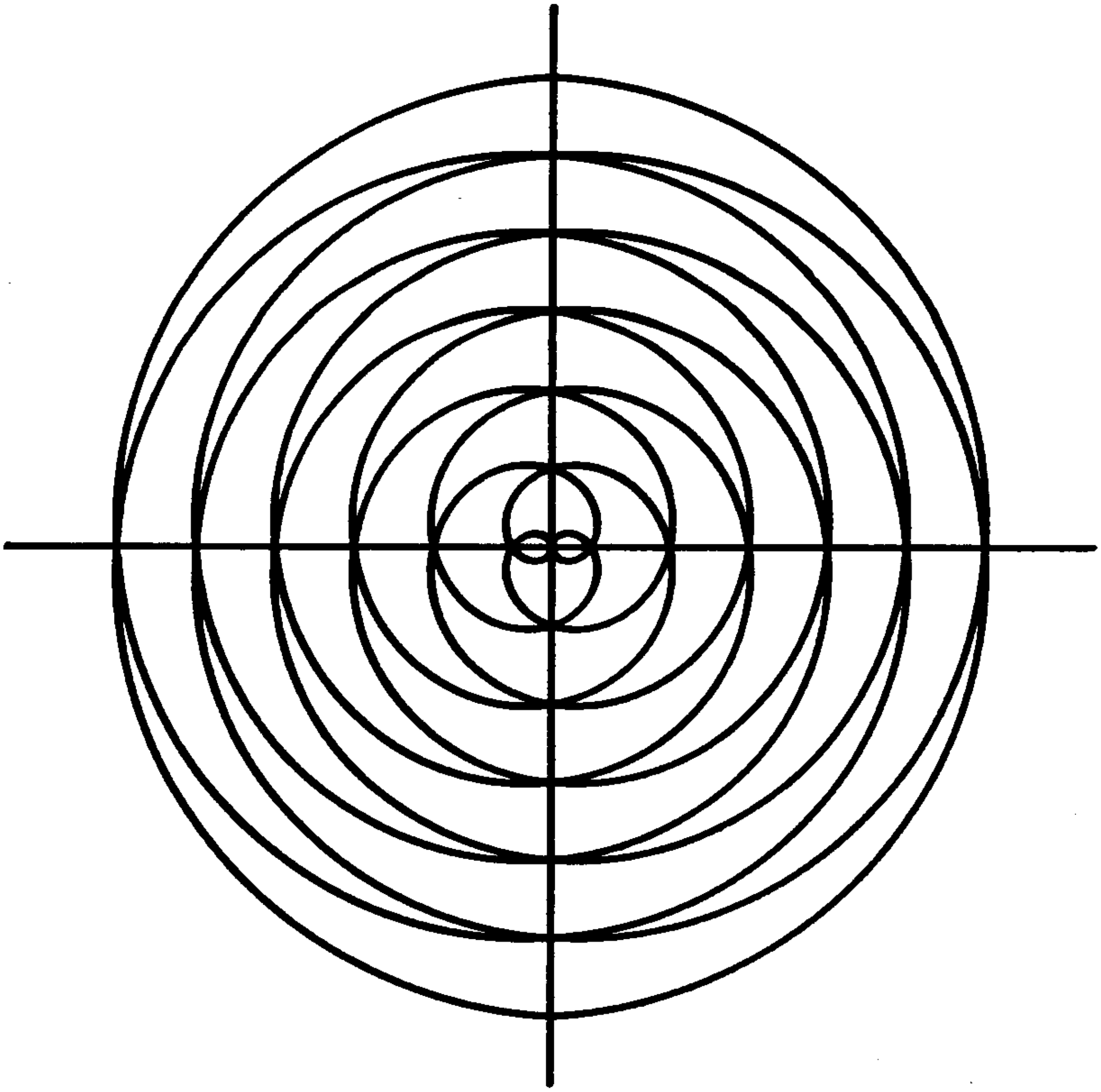
SAMPLE 7

file name : MIRROR
file type : .PP
length : 1650
last edit date : 5-5-82

```
%MIRRORED SPIRAL DRAWING PROGRAM
(DVAR,ANG1,VAR1,VAR2,VAR3,VAR4,VARI,VARC,VARS,CNT1,MIR1,MIR2,MIR3)
(REF,X,Y)                     ; GO HOME
G24                           ; NON-CORNER ROUNDING
G0 X6.0 Y4.0                  ; HOME OFFSET
G92                           ; RESET ABSOLUTE POSITION REGISTERS
G1 F100.0                     ; LINEAR MODE 100 IPM
G91                           ; INCREMENTAL MODE
Z1.0                          ; PEN UP
G4 F0.5                       ;
X-3.0 Y0                      ;
Z-1.0                         ;
G4 F0.5                       ;
X6.0 Y0                       ;
Z1.0                          ; DRAW X AND Y AXIS
G4 F0.5                       ;
X-3.0 Y-3.0                   ;
Z-1.0                         ;
G4 F0.5                       ;
X0 Y6.0                       ;
Z1.0                          ;
G4 F0.5                       ;
Y-3.0                         ; STARTING POINT
CNT1=0                        ;
(DENT,ENT2)                   ;
ANG1=0                        ; STARTING ANGLE
VARC=3                        ; NUMBER OF CYCLES
VARI=1.0                      ; INCREMENT OF ANG1
VARS=50.0                     ; SCALING FACTOR
Z-1.0                         ; PEN DOWN
G4 F0.5                       ;
G23 F25.                      ;
VAR1=INT(10000*(VARS(RAD(ANG1/360)*COS(RAD(ANG1))))
VAR2=INT(10000*(VARS*RAD(ANG1/360)*SIN(RAD(ANG1))))
```

```
(DENT,ENT1)
ANG1=ANG1+VARI                                  ; INCREMENT ANGLE
VAR3=INT(10000*(VARS*RAD(ANG1/360)*COS(RAD(ANG1))))
VAR4=INT(10000*(VARS*RAD(ANG1/360)*SIN(RAD(ANG1))))
X=(VAR3-VAR1)/10000                             ; XY MOVE
Y=(VAR4-VAR2)/10000
VAR1=VAR3 VAR2=VAR4                             ; INCREMENT POSITION
(JUMP,ENT1,ANG1.LT.(VARC*360))                  ;KEEP GOING UNTIL 2
                                                CYCLES

Z=1.0
G4 F0.5
CNT1=CNT1+1
(JUMP,MIR1,CNT1.EQ.1)
(JUMP,MIR2,CNT1.EQ.2)
(JUMP,MIR3,CNT1.EQ.3)
(REF,X,Y)
M2
(DENT,MIR1)
(MIR,X1,Y0)
G90 X0 Y0
G91
(JUMP,ENT2)
(DENT,MIR2)
(MIR,X1,Y1)
G90 X0 Y0
G91
(JUMP,ENT2)
(DENT,MIR3)
(MIR,X0,Y1)
G90 X0 Y0
G91
(JUMP,ENT2)
M30
```



Appendix 4

Command and Parameter Summary

UNIDEX 16 MACHINE G-CODES

CODE#	CODE TYPE	CODE OPERATION
G0/H0	Modal	Rapid traverse positioning. Traverse rate is parameter #240/241.
G1/H1	Modal	Linear contour with constant vectorial velocity
G2/H2	Modal	Circular CW contouring. IJK/QPR used for incremental arc centers.
G3/H3	Modal	Circular CCW contouring. IJK/QPR used for incremental arc centers. Polar coordinates use L/O = radius, C/A = start. angle, D/B = end angle.
G4	Own block	Dwell delay. Follow by F code of duration in seconds. (For example, G4 F100.)
G8/H8	Block	Acceleration on. Parameter 250 = auto; 251, 252 = rate; 510, 511 = threshold feedrate.
G9/H9	Block	Accel/Decel on. Parameter 250 = auto; 251, 252 = rate; 510, 511 = threshold.
G17/H17	Modal	Select X and Y/U and V to be on same plane for circular interpolation.
G18/H18	Modal	Select X and Z/U and W to be on same plane for circular interpolation.
G19/H19	Modal	Select Y and Z/V and W to be on same plane for circular interpolation.
G23/H23	Modal	Selects corner rounding mode.
G24/H24	Modal	Cancels corner rounding mode.
G40	Modal	Cancel/turn-off cutter radius compensation/ offset.
G41	Modal	Turn on cutter radius compensation to the LEFT side of workpiece.
G42	Modal	Turn on cutter radius compensation to the RIGHT side of workpiece.
G70/H70	Modal	Select English Programming units. 1 = .0001 inch, 1.0 = 1.0000 inch.
G71/H71	Modal	Select Metric programming units. 1 = .001 mm, 1.0 = 1.000 mm.

G76	Own block	Circular Peck Drilling canned cycle. L=radius, C=#holes, D=angle, Z=depth, F=feedrate, S=spindle rpms
G77	Own block	Circular Pocket Milling canned cycle L=radius (+ = CW, - = CCW), I=Increment per cutter pass, F=feedrate, S=spindle rpms
G78	Own block	Rectilinear Pocket Milling canned cycles X=X radius (+ =CW, -=CCW), Y=Y radius, I=increment X-axis, J=increment Y- axis, F=feedrate, S=spindle rpms
G80	Modal	Cancel canned cycles G81-G85
G81	Modal	Linear Peck Drilling canned cycle X=axis length, Y=axis length, Z=axis depth, R=Z return, C=#holes, D=dwell, F=feedrate, S=rpms
G82	Modal	Stop Peck Drilling canned cycle. Same as G81 except spindle stops at hole bottom.
G83	Modal	Incremental Peck Drilling canned cycle. X=axis length, Y=axis length, Z=axis depth, R=Z return, I=peck length, J=peck return, C=#holes, D=dwell, F=feedrate, S=rpms
G84	Modal	Reverse Peck Drilling canned cycle. Same as G81 except spindle reverses at hole bottom.
G85	Modal	Slow Peck Drilling canned cycle. Same as G81 except Z- axis return from hole bottom is at slow speed.
G90/H90	Modal	Absolute programming
G91/H91	Modal	Incremental programming (default)
G92	Immed	Preload programmed axis position registers
G98	Modal	G81-G85, R return point override on Z axis. Returns to starting Z depth.
G99	Modal	G81-G85 use R temporary return point on Z axis.
/	Block	Conditional block delete, when front panel Block Delete Switch is On. First character in block.

UNIDEX 16 ADVANCED PROGRAMMING

VARIABLE NAMES - 2 to 4 characters. First two must be alphabetic upper case (ABCD, VAR1, PT10, etc.) ($Gnn specifies a global variable, where nn is a number which designates the variable label.)

SUBROUTINE/ENTRY POINT NAMES - 4 characters, alphanumeric uppercase. (ABCD, 1234, STRT, etc.)

CONSTANTS - 3.2914 = real constant, all integers converted to real. VAR1 = 12 same as VAR1 = 12.0 ; "ABCD" = ASCII constant (4 chars. max) ; H,1234 = hex/binary constant (8 digits max).

ABSOLUTE POSITION REGISTER VARIABLES - $XAP (X-axis) $YAP (Y-axis) $ZAP (Z-axis) $UAP (U-axis) $VAP (V-axis) $WAP (W-axis)

RELATIVE POSITION REGISTER VARIABLES - $XRP (X-axis) $YRP (Y-axis) $ZRP (Z-axis) $URP (U-axis) $VRP (V-axis) $WRP (W-axis)

MATH OPERATIONS - ABS(x) (absolute value of X); SQR(x) (square root of X); INT(x) (integer of X); ! (exponential); * (multiply); / (divide); + (add); - (subtract).

TRIG OPERATIONS - SIN(x) (sine x); COS(x) (cosine x); TAN(x) (tangent x); ATN(x) (arctangent x).

CONVERSION OPERATIONS - DEG(x) (radians to degrees); RAD(x) (degrees to radians); BTF(x) (binary to floating); FTB(x) (floating to binary).

NOTE: Above operations apply only to real variables and constants.

UNIDEX 16 RS-447 COMMANDS

CMD	USEAGE	COMMAND OPERATION
ABTS	(ABTS,xxxx)	Abort subroutine - returns to given entry point. Allows xxxx = #VAR1
ADTC	(ADTC,D,X65,F1.)	Programmable accel/decel time constant in milliseconds. D = defaults. F or E = vector feedrate threshold in inch/min or mm/min. G70/G71.
CCP	(CCP,T1,0.15,T3,0)	Cutter compensation to tool diameter (modal)
CIRQ	(CIRQ,opt,XXXX)	MST bus interrupt enable. Options 0-5. Refer to INTx command.
CLS	(CLS,xxxx,parm1)	Call subroutine with multiple parameters. Allows xxxx = #VAR1.
CMD	(CMD,..message..)	Message to external hardware when handshaking is required. The CMDS command is similar to CMD, but waits for a Service Request before going to next block.
DATA	(DATA,..text..)	Will permit a parts program to write data to a dedicated file called DATAFILE .PP in memory. You may then rename and edit file.
DISP	(DISP,...message..)	Message may be sent to screen to be included as one of the machine mode displays.
DENT	(DENT,xxxx)	Define an entry point.
DFLS	(DFLS,xx,p1 *--)*	Define library subroutine (restores machine status to that in effect before call)
DFS	(DFS,xxx,p1 *--)*	Define subroutine with variable parameters defined implicitly.
DVAR	(DVAR,var1,var2)	Define variables.
INTq	(INT2,opt,xxxx)	Programmable interrupt enable. q = 1-4. Opt 0 = disable, 1 = data/nxt blk, 2 = abort/nxt blk, 3 = abort/no rtn, 4 = finish/ nxt blk, 5 = finish/no rtn.
IOFT	(IOFT,BIN,4)	I/O channel data format, 1-4 bytes of binary or BCD data (modal).
JUMP	(JUMP,xxx,X.GT.Y)	Jump to defined entry point if condition is true. Allows xxx = #VAR1.
MIR	(MIR,X1,Y0)	Mirror axis image. 1 = 0, 0 = off (modal)

MSG	(MSG,<ABxx>,text)	CRT message (xx=seconds). If A or B, then output to ports also. Wait for cycle-start if no xx time exists. Cycle- start aborts wait. #VAR1 in text displays real, #H:VAR1 for hex, #C:VAR1 for char. For input, use variables within <AB,var1>.
REF	(REF,X,Y,Z)	Go Home. Simultaneous axis home using parameter rates and offsets.
RPT	(RPT,xx *--)*	Repeat loop for xx times.
SCF	(SCF,X0.33)	Set axis scaling factor (modal)
SCO	(SCO,1)	Set all axis scaling on/off (modal)
SYNC	(SYNC,1)	Six axis synchronized contouring, 1=on, 0=off (modal).
UMFO	(UMFO,x,100)	MFO switch control, x=1 disables MFO front panel switch. X=0 enables front panel switch. Disable fixes feedrate at from 0 to 200% of programmed rate.
UMSO	(UMSO,x,150)	MSO switch control, X=1 disables MSO front panel switch. X=0 enables it. Disable fixes spindle speed at from 0 to 200% of programmed rate.

UNIDEX 16 MACHINE M-FUNCTIONS

CODE	RESTRICTIONS	CODE OPERATION
M0	--	Program stop. Cycle start resumes. Parameter #442 controls options.
M1	--	Optional stop. Front panel switch enable. Uses parameter #442 also.
M2	--	Program end. Parameter #442 controls options.
M3	MTI board only	Spindle CW on (once/per/rev spindle types only).
M4	MTI board only	Spindle CCW on (once/per/rev spindle types only).
M5	MTI board only	Spindle off.
M7	MTI board only	Coolant #1 on.
M8	MTI board only	Coolant #2 on.
M9	MTI board only	Coolant off.
M10	MTI board only	Spindle clamp on.
M11	MTI board only	Spindle clamp off.

M30	– –	End of Programmed data. Parameter #442 controls options.
M40	MTI board only	Low range spindle speed select.
M41	MTI board only	High range spindle speed select.
M42	MTI board only	RPM down spindle speed changer. Continues down until M44 command.
M43	MTI board only	RPM up spindle speed changer. Continues up until M44.
M44	MTI board only	Spindle speed changer stop.
M47		Return to program start.

UNIDEX 16 LOGIC PROGRAMMING

BINARY AND REAL - .EQ. (equal to) .NE. (not equal to)

REAL ONLY - .GT. (greater than), .GE. (greater or equal to), .LT. (less than), .LE. (less than or equal to)

BINARY ONLY - .HI. (higher than), .LS. (lower or same), .ADDx. (binary addition), .SUBx. (binary subtraction), .ANDx. (AND operation), .ORx. (OR operation), .NOTx. (NOT operation), .XORx. (Exclusive OR), .LSLx. (logical shift left), .LSRx. (logical shift right), .ROLx. (Rotate left), .RORx. (Rotate right).

NOTE: X = 1-4 bytes; no X defaults to one byte.

TSTA.xxxx (test AND function) needs 2 operands in a specific order. If all bits tested are 1, then result is true.

.TSTO.xxxx (test OR function) needs 2 operands in a specific order. If any bits tested are 1, then result is true.

NOTE: Above two operations apply to all bytes of binary variables and constants.

UNIDEX 16 EEPROM PARAMETERS

PARA#	DEFAULT DATA	PARAMETER DESCRIPTION
100	65535	End address of users RAM; add 32768 for each additional 32K of RAM.
101	60	Time after which CRT blanks to save CRT screen.
102	00000000	Soft key attributes 00000010 = underlined; 11000000 = reverse video; 00000000 = not underlined.
110	00001001	ASCII character signaling transmission end of a single file.
111	00010100	ASCII character signaling transmission end of all files.
112	002	Status line alarm message duration 0-254 seconds, 255 = infinite
120	00000	Real time clock option; 00000 = no real time clock; 00001 = real time clock.
121	00000001	Reserved for future 25 pin J5 control: 00000000 = Parallel Keyboard; and 00000001 = Joystick or Axis Calibration.
122	000	Reserved
130	007	Disk drive format (1-9).
131	000	Reserved
132	000	Reserved
140	000	Reserved
141	00000000	Reserved
142	00000000	Reserved
150	000	A/C power fail. 0 = disable A/C fail interrupt, 1 = enable
151	002	Check amplifier fault. 0 = disable amp fault interrupt; 1 = enable; 2 = enable and force to single mode.
152	001	D/A overrun. 0 = disable D/A overrun interrupt; 1 = enable; 2 = enable and force to single mode.
160	00000000	Select + or - move for jog mode (X, Y and Z axes only)
161	00000000	Set RS-232 Time-Out time in seconds, 0 - 65,535 (0 is default setting of 5 minutes.)
162	00000000	Reserved

UNIDEX 16 EEPROM PARAMETERS

PARA#	DEFAULT DATA		PARAMETER DESCRIPTION
	METRIC / ENGLISH		
	1:1	*1:1*	
200	100000	39370	X axis (G71) metric constant (100000 * counts out/ prog. mm)
201	254000	100000	X axis (G70) english constant (100000 * counts out/ prog. inch)
202	100000	39370	Y axis (G71) metric constant
210	254000	100000	Y axis (G70) english constant
211	100000	39370	Z axis (G71) metric constant
212	254000	100000	Z axis (G70) english constant
220	100000	39370	U axis (H71) metric constant
221	254000	100000	U axis (H70) english constant
222	100000	39370	V axis (H71) metric constant
230	254000	100000	V axis (H70) english constant
231	100000	39370	W axis (H71) metric constant
232	254000	100000	W axis (H70) english constant
240	60000		Feedrate limit for X, Y and Z axis (in counts/min)
241	60000		Feedrate limit for U, V, and W axis (in counts/min)
242	11000000		Axes which exist in system (XYZUVW00)
250	00000000		Acceleration/deceleration default 1 = active (G8,G9,H8,H9,0000)
251	002		XYZ Accel/decel time mS = 32.768 * parameter (range 65.5 - 8,355.8)
252	002		UVW Accel/decel time mS = 32.768 * parameter (range 65.5 - 8,355.8)
260	11111100		Skip limit check 0 = skip (XYZUVW00)
261	300		Byte size of parts program stack
262	10000110		Parts program default - bit 7: 1 = track program step; 0 = machine step bit 2: 1 = G71 metric default; 0 = G70 english bit 1: 1 = H71 metric default; 0 = H70 english

UNIDEX 16 EEPROM PARAMETERS

PARA#	DEFAULT DATA	PARAMETER DESCRIPTION
300	00100000	Axis selected to move 1st in "back-end" mode (XYZUVW00)
301	1000000	Axis selected to move 2nd in "back-end" mode
302	00000100	Axis selected to move 3rd in "back-end" mode
310	00011000	Axis selected to move 4th in "back-end" mode
311	00000000	Axis selected to move 5th in "back-end" mode
312	00000000	Axis selected to move 6th in "back-end" mode
320	00000000	Axis home direction 1=positive direction
321	00300000	Go home feedrate X axis in counts/min (range 229-8,000,000)
322	00300000	Go home feedrate Y axis in counts/min (range 229-8,000,000)
330	00300000	Go home feedrate Z axis in counts/min (range 229-8,000,000)
331	00300000	Go home feedrate U axis in counts/min (range 229-8,000,000)
332	00300000	Go home feedrate V axis in counts/min (range 229-8,000,000)
340	00300000	Go home feedrate W axis in counts/min (range 229-8,000,000)
341	00000000	Go home offset, X axis (in counts)
342	00000000	Go home offset, Y axis
350	00000000	Go home offset, Z axis
351	00000000	Go home offset, U axis
352	00000000	Go home offset, V axis
360	00000000	Go home offset, W axis
361	00000000	Reserved
362	00000000	Reserved

UNIDEX 16 EEPROM PARAMETERS

PARA#	DEFAULT DATA	PARAMETER DESCRIPTION
400	000010	M function time from data latch until strobe (MTMF)
401	000010	M function time to debounce acknowledge or strobe length (MTFN)
402	000010	S function time from data latch until strobe (STMF)
410	000010	S function time to debounce acknowledge or strobe length (STFN)
411	000010	T function time from data latch until strobe (TTMF)
412	000010	T function time to debounce acknowledge or strobe length (TTFN)
420	007	Spindle type 1 = Once/per/rev; 2 = 2's complement; 3 = Unipolar binary; 4 = complement binary; 5 = offset binary; 6 = bipolar complement; 7 = unipolar BCD; 8 = bipolar BCD.
421	020	Spindle speed rpm up/down soft key (once/per rev).
422	09999	High gear range upper rpm limit or bi/unipolar positive high limit.
430	00001	High gear range lower rpm limit or bi/unipolar positive low limit.
431	09999	Low gear range upper rpm limit or bipolar negative high limit.
432	00001	Low gear range lower rpm limit or bipolar negative low limit.
440	0137	Once/per/rev spindle teeth gear ratio = low range * 1024.
441	003	Once/per/rev speed changer accuracy deadband.
442	002	M0, M1, M2, M30, M47 Special M-functions user options.
450	002	M-function user options. Type 0-3 only.
451	002	S-function user options. Type 0-3 only.
452	003	T-function user options. Type 0 software function does tool offset and radius compensation. No type 3 or 7. Type 2, 4, no tool offset, only tool radius compensation.

TYPE CODE:

0 = Perform any software function, output code to bus and wait for acknowledge.

1 = Skip everything, no software function and no code output to bus.

2 = Perform any software function, output code to bus but don't wait for acknowledge.

3 = Perform any software function, output code to bus and timed wait for acknowledge.

4 = Perform any software function, output no code to bus.

5 = Output code only to bus, wait forever for acknowlegde.

6 = Output code only to bus, don't wait for acknowlege.

7 = Output code only to bus, timed wait for acknowlege.

460	0	Reserved
461	0	Reserved
462	0	Reserved

UNIDEX 16 EEPROM PARAMETERS

PARA#	DEFAULT DATA	PARAMETER DESCRIPTION
500	0	Reserved
501	0	Reserved
502	0	Reserved
510	00300000	Feedrate threshold for automatic Accel/Decel XYZ axes. (in counts/min)
511	00300000	Feedrate threshold for automatic Accel/Decel UVW axes. (in counts/min)
520	00000000	Program new name for X axis. New name must consist of two characters (upper case alphabetic) where 01 - 26 represent A - Z respectively.
521	00000000	Program new name for Y axis.
522	00000000	Program new name for Z axis.
530	00000000	Program new name for U axis.
531	00000000	Program new name for V axis.
532	00000000	Program new name for W axis.
540	00000000	Unrelated functions: Bit 0 to 1 = Truncation of message output. Bit 1 to 1 = Port-B will not echo incoming characters. Bit 2 to 1 = Port-A will not echo incoming characters. Bit 3 to 1 = New axes' names will apply to program. Bit 4 to 1 = Execution of "Autoexecute" file.
541	00000000	Enables errors and statuses to be output through the RS-232 ports or I/O channel to host computer. Bit 0 set to1 = Send to Port-B. Bit 1 set to 1 = Send to Port-A. Bit 2 set to 1 = Send MSG command to Port-B. Bit 3 set to 1 = Send MSG command to Port-A. Bit 4 set to 1 = Send CMD command to Port-B. Bit 5 set to 1 = Send CMD command to Port-A. Bit 6 = 1, input from port is Binary. Bit 6 = 0, ASCII. Bit 7 set to 1 = Error code in Hex sent to I/O address $700

UNIDEX 16 EEPROM PARAMETERS

PARA#	DEFAULT DATA	PARAMETER DESCRIPTION
542	00000000	Specifies the Service Request character expected after the CMDS command.
550	00000000	Designates the password that will be required to enter the edit mode (0 = no password).
551	00000000	Designates the password that will be required to enter the file mode.
552	00000000	Designates the password that will be required to enter the machine mode.
560	00000000	Designates the password that will be required to enter the parameter mode.
561	00000000	Designates the password that will be required to enter the test mode.
562	00000000	Designates the password that will be required to run a command file.