DYNAMIC DEBUGGING SYSTEM

Computer Mart of New Jersey, Inc.
501 Route 27
Iselin, New Jersey 28830
(201) 283-0600

1. PROGRAM OVERVIEW

The DYNAMIC DEBUGGING SYSTEM (DDS) is a powerful debugging facility for 8080 assembly language programs. DDS operates in a variety of ways which you select to meet your needs. DDS can run your program one instruction at a time, automatically maintaining on the screen a display of all registers, the ensuing instructions, and portions of memory. Like conventional debugging programs, DDS can also operate in a breakpoint mode in which the program runs at full speed until it reaches one of several selected instructions. However, when using breakpoints you have to know what the program will do and programs are debugged because they don't do what they are supposed to! Thus, although a breakpoint capability is necessary, if it is the only facility available the debugger will be tedious to use, since you will have to breakpoint around almost all branch instructions. The automatic program monitoring facilities provide a powerful alternative to breakpoints. Rather than running the program yourself with breakpoints, DDS can run the program for you, until a condition specified occurs or DDS detects an error by itself.

DDS functions fall into three general categories:

* Program Display
* Program Execution Control
* Program Monitoring Functions

1.1 Program Display

DDS displays the basic data used by your program. This display is automatically updated by DDS. It does not have to be continuously requested. The DDS display includes the following:

* The next five instructions to be executed in mnemonic format together with hex addresses and op codes.
* All operating registers and the words pointed to by the BC, DE and HL registers pairs.
* The stack pointer and the last five words pushed onto the stack together with the words they point to.
* A mnemonic expansion of the condition codes.
* The return addresses from the last five CALL instructions.

In addition to the information above, the user can request memory displays in a combined hexadecimal and character format, or as program mnemonic codes.

1.2 Program Execution Control

DDS provides a wide range of commands that allow you to communicate with your program, change registers, memory, instructions, data, etc. Among the facilities provided are:

* register and register-pair modification
* data modification, by byte or word, in hex or characters
* stack modification, commands to PUSH and POP data to the stack
* memory moves, memory filling, memory searching

1.3 Program Monitoring

You specify the number of program instructions to be executed before control is returned from DDS. After executing each instruction, DDS tests various stop conditions and pauses with a message if any are met. You select some of the conditions while DDS automatically checks for others.

Following are some of the stop conditions DDS can monitor for you:

* Address Stop - DDS stops when a specified address is reached. This is similar to a breakpoint, except all other stop conditions are tested.
* Opcode Stop - DDS stops when a specified opcode is executed. This facility makes it easier to use DDS when you don't have the most recent program listing.
* Value Stop - DDS stops when a register, register-pair, byte or word attains a specified value. This is especially valuable when you want to see what happens at a Particular iteration of a long loop and don't want to breakpoint through each iteration.
* Store Stop - DDS stops when the program attempts to store into a specific location. This is useful when you know the program is storing data incorrectly, but you don't know which instruction is responsible.
* Address Ranges - DDS stops when the program references data onside a set of predefined ranges.
* Stack Pointer Ranges - DDS stops when the stack pointer goes outside a set of predefined ranges.

2. DDS DISPLAY

The DDS display is divided into four distinct regions:

* Program Status

* Memory Display

* Error Message Line

* Command Input Line

2.1 Program Status Display

The program status display indicates the current contents of the programs operating registers and important memory locations. It is divided into the following sections:

* Call Stack

* Stack

* Instructions

* Registers

* Condition Flags

2.1.1. Call Stack

The call stack displays the return addresses from the last five subroutine CALL instructions. The stack itself can contain up to 15 entries. The topmost entry is the return address for the last subroutine called. Only the actual number of subroutines called are displayed. If there have been no calls the call display will be blank.

2.1.2. Stack

The stack display shows the value of the stack pointer and the last five entries pushed onto the stack. The topmost entry is the last item pushed onto the stack. The format of an entry on the stack is:

```
nnnn=xxyy zz
```

where:

nnnn	is the actual address on the stack
xxyy	is the word located at address nnnn. This word is displayed reversed to conform to standard 8080 conventions, i.e., yy is the byte at address nnnn, xx is the byte at address nnnn+1

zz	is the character representation of xxyy. Non-displayable characters are represented with periods.

2.1.3. Instructions

The next five instructions to be executed are displayed in program mnemonics. The address of the instruction is displayed to the left of the mnemonics, the hex opcode is displayed to the right. The P register is implicitly displayed; it is the address of the first instruction. Both 8 and 16 bit operands are displayed in hex in the instruction. The 16 bit operands are reversed. Therefore the instructions can be interpreted in hex, in the actual order stored in memory, by reading the hex bytes from right to left starting with the hex opcode.

2.1.4. Registers

The register display shows the contents of the operating registers, as well as the words pointed to by the BC, DE and HL register pairs. The PSW is displayed on the first line in the following format:

```
AF=xxyy
```

where:

xx	is the value of the A register
yy	is the value of the condition flag portion of the PSW. A mnemonic interpretation of the condition flags is also provided and is described in 2.1.5.

The register pairs are displayed on the next three lines in the following format:

```
NM=nnmm=xxyy zz
```

where:

NM	is a register pair designator, either BC, DE, or HL.
nnmm	is the value of the register pair, register N has the value nn, register M has the value mm.
xxyy	is the word located at address nnmm. This word is displayed reversed to conform to standard 8080 conventions, i.e., yy is the byte at address nnmm, xx is the byte at address nnmm+1. In particular, if NM is HL, then the byte located by the HL registers and referenced as M in the 8080 assembly language, is given by yy. The character representation of xxyy is zz.

Non-displayable characters are represented with periods.

2.1.5. Condition Flag Mnemonics

The condition flats displayed with the PSW in hex are also expressed in more meaningful mnemonics. The format depends on the display supported. For 64 column displays the flags are expressed as single characters, and appear to the right of the hex PSW. In 80 column displays the flags are displayed as text in their own section of the display.

2.1.5.1. 64 Column Condition Flats

In the 64 column display the condition flags are displayed as single letters to the right of the PSW. Each of the five testable conditions is assigned a letter code, which is displayed if the condition is true, i.e. the corresponding bit in the PSW is on. If the condition is false, a blank is displayed instead. The following codes are used to display the condition flags:

M	- Sign (minus)
Z	- Zero
A	- Auxiliary Carry
P	- Parity (even)
C	- Carry

2.1.5.2. 80 Column Condition Flags

In the 80 column display the condition flats are displayed as text to the right of the registers. Each possible condition flag appears on a separate line, which is displayed if the condition is true, i.e. the corresponding bit in the PSW is on. The following text is used to display the condition flags:

```
MINUS
ZERO
AUX CARRY
EVEN CARRY
CARRY
```

2.2 Memory Displays

DDS can display memory in either a combined hexadecimal and ASCII format or as program mnemonic codes. The memory display follows the program status display and is either 6 lines long for 16 row displays, or 11 lines for 24 row displays.

2.2.1. Hexadecimal/ASCII display

The format of the hexadecimal/ASCII memory display is:

```
nnnn xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx CCCCCCCCCCCCCCCC
```

where:

nnnn	is the starting address of the current display line.
xxxx	are the hexadecimal value of the bytes starting at address nnnn and displayed from left to right in ascending addresses. The data bytes are not reversed in this display.
CCCC	is the character value of the corresponding bytes. Non-displayable characters are represented with a period.

2.2.2. Program Instructions

DDS can also display memory in program mnemonics. The format is identical to that used in the program status display.

2.3. Error Message Line

After executing each instruction in the program, and after each typein, DDS checks a series of conditions which the programmer has specified, and certain conditions of its own. When this process detects errors, they are displayed on the error line. Each error has a two character code, which corresponds to the first two characters of the command typein which is being detected, or a two character code assigned by DDS for conditions it checks itself. The error line is the second to last line of the display.

2.4. Command Input Line

The command input line is the last line of the DDS display. It begins with the prompting characters =>. Input entered from the keyboard is displayed here. The command line is displayed until it has been successfully executed at which time it is cleared.

3. KEYBOARD PROCESSING

DDS accepts input from the keyboard and displays it on the command line after the prompting characters (_>). Input errors can be corrected using either the rubout or break keys. Depending on the hardware supported, the rubout key will either erase the last character entered, or echo it back to the console as it is deleted. The break key will erase the entire entry. With the exception of the one character typeins (the stepping character and the repeat line command character) all DES commands terminate with a period ‘.’ Immediately upon receiving the period, DDS checks the syntax of the entry. If a syntax error is encountered DDS sounds the console bell and the entry remains on the command input line. It can be corrected by rubbing out and re-entering the characters, or erased by entering a break If there is no syntax error, DDS attempts to process it, but other errors might occur; for example, an internal table may be full. If this happens, DES treats the command as if it had a syntax error. When the command has been successfully executed, DDS clears the command input line. The DDS command input line is 40 characters long. If more characters are entered DDS will not enter them on the display; it will be necessary to rubout the last few characters and end the command within 40 characters.

Stepping Character

The stepping character (]) in the release version, but modifiable, is a single character typein which causes DDS to resume execution of the program. The stepping character is not displayed. The effect of the stepping character depends on the usage of the ST and BK commands. If breakpoints were entered using the BK command the stepping character allows the program to run at full speed until the next breakpoint is encountered. An exception occurs when the program is already at a breakpoint address, in which case the stepping character causes an advance to the next instruction. If breakpoints have not been entered or if they have been cleared by a BK. typein, entry of the stepping character causes the program to advance the number of instructions given by the ST typein, or until a stop condition is detected.

Repeat Command Character

The repeat command character (LF) in the release version, but modifiable, causes DDS to repeat the last command entered. This works with all commands, but is meaningfully employed only with selected commands. It is commonly used with the FIND command to locate the next occurrence of the search pattern.

4. PROGRAM MONITORING

One of the most valuable features of DDS is its ability to monitor the progress of your program and stop when predetermined conditions occur. DDS checks these conditions after each instruction is executed even though the DDS display might not be updated. When running with a video display DDS can execute approximately 30 instructions per second when the display is updated after each instruction and over 300 instructions per second when the display is updated every 255 instructions. DDS tests conditions you specify with DDS commands, as well as others it automatically checks by itself. When any of these conditions occur, program execution is stopped and the condition causing the stop displayed.

4.1. Automatic Return Address Validation

DDS analyzes subroutine usage to detect improper stack usage. When subroutines make extensive use of the stack for intermediate storage it is easy to make the mistake of not balancing POPS and PUSHs. When this happens, the return instruction will be invalid and a wild branch will occur. This type of error is particularly difficult to detect with conventional debugging aids. To detect these errors DDS maintains its own stack of return addresses from CALL instructions. Whenever a subroutine is called its return address is added to the call stack. Whenever a return from a subroutine is executed DDS deletes the top entry on the call stack, but only if it agrees with the top entry on the users stack. If the two addresses do not agree, DDS stops with a QR (questionable return) error code. Certain subroutines use the stack in a manner inconsistent with this usage. In this case Automatic Return Address Validation can be disabled with the TC. command.

4.2. Illegal Instruction

Certain 8080 opcodes are undefined. Whenever the program attempts to execute one of these, an II (illegal instruction) error message will be issued. The II error message is also issued whenever an RST or HLT instruction is about to be executed. After the stop, the illegal instruction may be executed if the stepping character is typed.

4.3. Programmed Stop Conditions

DDS provides a wide range of programmer commands which can be used to stop execution of the program. These are described in the section on DES command requests, and are summarized below.

AR	-	Address Range
AS	-	Address Stop
OS	-	Opcode Stop
RS	-	Reference Stop
SR	-	Stack Range

SS	-	Store Stop
VS	-	Value Stop on Register
VSB	-	Value Stop on memory Byte
VSP	-	Value Stop on Register Pair
VSW	-	Value Stop on Memory Word

4.4 Execution in ROM

DDS cannot control the execution of a program in ROM. If breakpoints are not active, and an instruction calls a subroutine in ROM, DDS will lose control until the subroutine returns. In this case the subroutine must return to the instruction immediately following the CALL, if it does not it will be necessary to enter a breakpoint at the return location. While the program is executing code in ROM the program monitoring facilities are inactive, immediately following the return from the subroutine they are reactivated.

5. DDS COMMANDS

DDS accepts commands which:

* control the display

* alter the flow of program execution

* select program monitoring functions

These commands are listed alphabetically in the remainder of this section. In the description of the formats [ ] indicate an optional part of the entry. For example, the command request:

```
     DM[,address[,lines[,start]]].
```

can be used in any of the following ways:

```
     DM.
     DM,3000.
     DM,20FF,1.
     DM,2033,2,4.
```

Three periods)...) indicate that the item to the left may be repeated. The number of repetitions is limited by the size of the keyboard entry (40 characters) as well as table size restrictions for the individual commands. For example, the command request:

```
     EB,address,data[,data] ... .
```

can be used in any of the following ways:

```
     EB,1000,C9.
     EB,2000,C9,CD,05,1.
```

When the format given specifies a byte, a one or two digit hex number may be given if the format specifies a word, up to four hex digits may be given.

ADDRESS RANGE request AR

Purpose:

The AR request causes a DDS stop to occur when the program references data outside of a predefined range.

Format:

```
AR[,low,high].
```

where:

low	is the start of an address range
high	is the end of an address range

Usage:

1. The AR command is in effect for all instructions executed under the control of the step command (ST). Upon returning from the last instruction DDS examines the next instruction to be executed and the ST is terminated if it stores into or references an address not included in any of the address ranges in effect.
2. Up to 5 address ranges can be active at one time.
3. To disable address range checking type AR.

4. To remove an address range clear all address ranges by typing AR., and re-enter those that are to remain.

5. The AR request does not trap stores nor references made by stack instructions. These can be detected using the SR command.

Responses:

AR

This message appears on the error line whenever a memory references outside of any address range is about to be executed.

ADDRESS STOP request AS

Purpose:

The AS request causes a DDS stop to occur before executing an instruction at a specified address.

Format:

```
AS[,address]... .
```

where:

address   is an address at which execution is to stop.

Usage:

1. The AS command is in effect for all instructions executed under the control of a step command (ST). Before executing an instruction DDS examines its address and the ST is terminated if the address is in the address stop list.

2. Up to 8 address stops can be active at one time.

3. To remove an address stop, clear all address stops by typing AS., and re-enter those that are to remain.

4. The address specified must be the address of the opcode of the instruction.

5. The address stop request differs from the BKP and GO requests in that DDS remains in control of the program and performs all other program monitoring facilities.

Responses:

AS

This message appears on the error line whenever an instruction at a specified address is about to be executed.

BREAKPOINT request BK

Purpose:

The BK request allows the program to run at full speed until it reaches a specified address, at which time DDS regains control.

Format:

```
BK[,address] ... .
```

where:

address is an address at which the breakpoint is to be inserted.

Usage:

1. Specifying one or more breakpoints with the BK request places DDS in breakpoint mode. In this mode entering the stepping character runs the program at full speed until it reaches one of the breakpoint addresses, when DDS regains control. DDS will riot allow breakpoint operation if the P register is the same as any breakpoint address. In that case entering the stepping character causes DDS to advance to the next instruction. This makes it possible to install a group of breakpoints in a program, and breakpoint to any of them (including the last one at which a stop occurred).

2. While the program is running at full speed the program monitoring facilities are disabled. When DDS regains control it checks all of the monitored conditions.

3. To clear breakpoints, and resume monitored execution type BK.

4. Breakpoints are implemented by overwriting the first byte of the instruction with an RST 7 (hex 0FFH) opcode. DDS remembers the old value of the instruction in a table of its own, and restores all the overwritten instructions immediately upon regaining control. DDS installs a jump in the RST 7 location, 38H, which goes back to DDS. These considerations imply that:

   * DDS cannot breakpoint through ROM

   * DDS will not work properly if the program being debugged also uses the RST 7 instruction.

   * An instruction can not be modified by the immediately preceding instruction.

   * DDS will regain control automatically if the program attempts to execute from non existent core (returned by processor as hex 0FF).

5. The GO request is similar to the breakpoint request, except the breakpoints do not remain in effect.

Responses:

None.

CALL request CALL

Purpose:

The CALL request allows a return address to be placed on the programmed call stack.

Format:

```
CALL,address.
```

where:

address	is the address to be placed on the programmed call stack.

Usage:

1. The programmed call stack is used by Automatic Return Address Validation to check on the legal use of the stack. Whenever a subroutine is called, its return address is placed on the call stack. The call command allows the user to place a return address on this stack.

2. This allows the user to resynchonize the call stack with the program stack after an invalid return has been detected. .

Responses:

None.

CLEAR SCREEN request CLR

Purpose:

The CLR request causes the screen to be erased and the display refreshed.

Format:

```
CLR.
```

Usage:

1. The CLR request is useful when the program being debugged has placed information on the screen which interferes with the display.

Responses:

None.

DISPLAY INSTRUCTIONS DI

Purpose:

The DI request controls a memory display in program mnemonic codes.

Format:

```
DI[,address[,1]].
```

where:

address	is the starting address for the memory display
1	requests the display to be on the right side of the screen

Usage:

1. The format of the instruction display is identical to the instructions displayed in the system status display; the address of the instruction, followed by the instruction mnemonics, followed by the hex opcode for the instruction.

2. When only the address is specified the instruction display begins on the left hand side of the screen under the system status display.

3. If a number 1 follows the address, the instruction display begins on the right-hand side of the screen under the system status display.

4. When DI. is entered, the instruction display is brought back with the left side of the screen as it appeared during the last DI display and right side blank.

5. The instruction display is normally refreshed each time DDS pauses for a new typein. It is not refreshed while the program is being run under a step command. The REF command can be used to disable the refreshing logic, this is useful for certain kinds of CRT displays, as it speeds up the processing.

Responses:

None.

DISPLAY MEMORY request DM

Purpose:

The DM request controls a memory display in hexadecimal and ASCII characters.

Format:

```
DM [,address[,lines[,start]]].
```

where:

address	is the starting address for the memory display
lines	is the number of lines requested
start	is the starting line number

Usage:

1. When only the address is specified the memory display starts at the specified address, and continues with consecutive addresses for the remaining lines of the display.

2. If the address and lines arguments are given, the display starts at the specified address and continues with consecutive addresses for the number of lines specified.

3. If a starting line number is given, the display is modified to show the requested information starting from the line specified.

4. These combined specifications allow the individual selection of any line in the memory display.

5. When DM. is entered the hex/ASCII memory display is brought back using the same attributes as were last specified.

6. The memory display is normally refreshed each time DDS pauses for a new typein. It is not refreshed while the program is being run under a step command. The REF. command can to used to disable the refreshing logic, this is useful for certain kinds of CRT displays, as it speeds up the processing.

Responses:

None.

DELAY request DY

Purpose:

The DY request is used to delay processing by DDS to facilitate observation of the programs flow.

Format:

```
DY,value.
```

where:

value	is a number from 1 to FFFF controlling the delay rate.

Usage:

1. The fastest DDS operation occurs when the delay factor is 1 and this is the initial setting. The delay is proportional to the delay factor selected. A delay of 1000 results in a 1/4 second delay, a delay of FFFF results in a 2 1/2 second delay.

Responses:

None.

ENTER BYTES request EB

Purpose:

The EB request allows modifying memory by specifying a series of hexadecimal bytes.

Format:

```
EB,address,data[,data] ... .
```

where:

address	is the starting address for the data
data	is a series of byte arguments which are to be stored.

Usage:

1. The data specified is stored in memory starting at the given address.

Responses:

None.

ENTER CHARACTERS request EC

Purpose:

The EC request allows modifying memory by specifying a series of alphanumeric characters.

Format:

```
EC,address,X ... .
```

where:

address	is the starting address for the characters
X	is an alphanumeric character (cannot be a period).

Usage:

1. The character string specified is stored in memory starting at the given address.

2. Like all other DDS commands, the EC request is terminated with a period, and therefore a period cannot be included in the character string.

Responses:

None.

ENTER P REGISTER request EP

Purpose:

The EP request is used to reset the program counter register.

Format:

```
EP,address.
```

where:

address	is the new address for the P register.

Usage:

1. The program counter, or P register, is chanted to resume processing at the specified address.

Responses:

None.

ENTER REGISTER request ER

Purpose:

The ER request allows changing the values of the programs operating registers.

Format:

```
ER,register, byte.
```

where:

register	designates one of the operational registers can be either A,B,C,D,E,H,I, or F.
byte	is a byte argument containing the new value.

Usage:

1. The specified register is changed.

2. To change the condition flags enter register F.

Responses:

None.

ENTER REGISTER PAIR request ERP

Purpose:

The ERP request allows changing the values of the program's operating register in pairs with a single typein.

Format:

```
ERP,pair,word.
```

where:

pair	designates one of the operational register-pairs, can be either B,D, or H.
word	is a word argument containing the new value

Usage:

1. The specified register-pair is changed.

2. The contents of word is not reversed before storing. Thus ERP,B,1234. will set the E register to 12 and the C register to 34.

Responses:

None.

ENTER STACK request ES

Purpose:

The ES request is used to reset the program's stack pointer register.

Format:

```
ES,value.
```

where:

value	is the new value for the stack pointer.

Usage:

1. The stack pointer register is changed to the specified value.

Responses:

None.

ENTER WORDS request EW

Purpose:

The EW request allows modifying memory by specifying a series of hexadecimal words.

Format:

```
EW,address,data[,data]... .
```

where:

address	is the starting address for the data
data	is a series of word arguments which are to be stored.

Usage:

1. The data words specified are stored in memory starting at the given address.

2. The contents of the words are reversed, byte for byte before storing, to conform to standard 8080 conventions. Thus the command 0,3000,1234. will store 34 at address 3002 and 12 at address 3001.

Responses:

None.

FILL request FILL

Purpose:

The FILL request is used to preset a region of memory.

Format:

```
FILL,address,count,value.
```

where:

address	is the starting address to fill.
count	is the number of bytes to fill.
value	is a byte which is used to fill memory.

Usage:

1. The fill command will place the value into each byte starting with the starting address for the number of bytes equal to count.

Responses:

None.

FIND request FIND

Purpose:

The FIND request is used to locate a series of bytes in memory.

Format:

```
FIND,byte[,byte[,byte]].
```

where:

byte	is a byte to be located.

Usage:

1. Up to 3 bytes can be specified. DDS searches memory for a pattern matching the number of bytes specified.

2. The FIND request works in conjunction with the memory display on the lower half of the screen. The search begins at the address following the first address displayed and ends at the. highest possible memory address (FFFF). If a mach is found the memory display is updated to start, with. the matching pattern. If no match is found the memory display is unchanged. Either the DM or DI displays can be used.

3. Using the FIND request in conjunction with the DI display facilitates using a listing which does not have the latest program addresses, since constant portions of the code can be identified and located.

Responses:

None.

GO request GO

Purpose:

The GO request allows the program to advance quickly to a specified address without DES intervention.

Format:

```
GO,address[,address] ... .
```

where:

address	is an address at which a breakpoint is inserted.

Usage:

1. The GO request is identical to the BK request with the single exception that the GC request is not automatically repeating. The addresses specified for a GO are in effect only for the original request. DDS automatically resumes monitored execution following the return from the G0.

Responses:

None.

MOVE request MOVE

Purpose:

The MOVE request is used to move a Clock of memory

Format:

```
MOVE,sending,receiving,count.
```

where:

sending	is the sending address for the move
receiving	is the receiving address for the move
count	is the number of bytes to be moved

Responses:

None.

Usage:

1. The bytes starting at the sending address are moved to the receiving address for the number of bytes specified by the count.

OPCODE STOP request OS

Purpose:

The OS request causes a DDS stop to occur before a specified opcode is executed:

Format:

```
OS[,opcode]... .
```

where:

opcode	is a hex opcode at which execution is to stop

Usage:

1. The OS command is in effect for all instructions executed under the control of a step command (ST). Upon returning from the last instruction DDS examines the next opcode and the ST is terminated if the opcode is in the opcode stop list.

2. Up to 5 opcode stops can be active at one time.

3. To remove an opcode stop, clear all opcode stops by typing OS., and re-enter those that are to remain.

Responses:

OS

This message appears on the error line whenever a specified opcode is about to be executed.

POP request POP

Purpose:

The POP request allows removing data from the users stack.

Format:

```
POP.
```

Usage:

1. The POP command increments the stack pointer by 2, thus simulating the effect of a POP instruction executed by the program. The data removed is not available through DDS.

2. The stack display is updated to indicate the new stack position.

Responses:

None.

PUSH request PUSH

Purpose:

The PUSH request allows entering data onto the users stack.

Format:

```
PUSH,word.
```

where:

word	is a word to be entered onto the users stack.

Usage:

1. Word is entered in the next available position in the users stack, and the stack pointer is decremented by 2. The display.is updated to indicate the change to the stack.

Responses:

None.

REFRESH request REF

Purpose:

The REF request controls whether or not the DM or DI display will be updated automatically.

Format:

```
REF.
```

Usage:

1. The REF request toggles the sense of the refresh switch, that is if it is off, REF. will turn it on, and if it is on, REF. will turn it off.

2. The refresh switch determines whether or not the DM or DI displays will be updated when DDS pauses for operator input. While DDS is running the program the DM or DI displays are not updated.

Responses:

None.

RETURN request RET

Purpose:

The RET request allows a return address to be removed from the programmed call stack.

Format:

```
RET.
```

Usage:

1. The programmed call stack is used by Automatic Return Address Validation to check on the legal use of the stack. Whenever a subroutine is called, its return address is placed on the call stack. The RET command allows the user to remove the top-most entry on this stack.

Responses:

None.

REFERENCE STOP request RS

Purpose:

The RS request causes a LIS stop to occur before the program references a specified location.

Format:

```
RS[,address]
```

where:

address	is an address which is to be protected against program references.

Usage:

1. The RS command is in effect for all instructions executed under the control of a step command (ST). Upon returning from the last intruction DDS examines the next instruction to to executed, and the ST is terminated if it references an address in the reference stop list.

2. Up to 5 references stops can be active at any one time.

3. To remove a reference stop, clear all reference stops by typing RS., and re-enter those that are to remain.

4. The RS instruction traps all memory references made by the following instructions:

```
LDA     addr
MOV     M
LDAX    reg
LHLD    addr  ;both the high and low addresses are trapped
opcode  M
```

5. The RS instruction does not trap references made by instructions which pop data from the stack (POP and RET). These references can be detected using the SR command.

Responses:

RS

This message appears on the error line whenever a reference to a specified location is about to occur.

STACK RANGE request SR

Purpose:

The SR request causes a DDS stop to occur when the stack pointer exceeds a predefined range.

Format:

```
SR [,low,high].
```

where:

low	is the start of the stack range
high	is the end of the stack range

Usage:

1. The SR command is in effect for all instructions executed under the control of a step command (ST). Upon returning from the last instruction DDS examines the stack pointer and the ST is terminated if it is not included in any of the stack ranges in effect.

2. Up to 2 stack ranges can be active at one time.

3. To disable stack range checking, type SR.

4. To remove a stack range clear all stack ranges by typing SR., and re-enter those that are to remain.

Responses:

SR

This message appears on the error line whenever the stack pointer lies outside of all stack ranges.

STORE STOP request SS

Purpose:

The SS request causes a DDS stop to occur before the program stores to a specified location.

Format:

```
SS[,address]... .
```

where:

address	is an address which is to be protected against program stores

Usage:

1. The SS command is in effect for all instructions executed under the control of a step command (ST). Upon returning from the last instruction DDS examines the next instruction to be executed, and the ST is terminated if it stores into an address in the store stop list.

2. Up to 5 store stops can be active at any one time.

3. To remove a store stop, clear all store stops by typing SS., and re-enter those that are to remain.

4. The SS instruction traps all stores made by the following instructions:

```
STA     addr
MOV     M,reg
STAX    reg
SHLD    addr  ;both the high and low addresses are trapped
opcode  M
```

5. The SS instruction does not trap stores made by instructions which push data onto the stack (PUSH and CALL). These stores can be detected using the SR command.

Responses:

```
SS
```

This message appears on the error line whenever a store to a specified location is about to occur.

STEP request ST

Purpose:

The ST request determines the number of instructions executed when the stepping character is entered, as well as the frequency with which the display is updated while DDS is stepping.

Format:

```
ST,instruction[,display].
```

where:

instruction	is the number of instructions DDS will attempt to execute when the stepping character is entered. This can range from l to FFFF. DDS will then run the program in single step mode until the specified number of instructions has been executed, or DDS detects a stop condition.
display	is the number of instructions DDS will execute before refreshing the display. This can range from 1 to FF.

Usage:

1. If breakpoints are active, then the stepping character causes the program to advance to the next breakpoint.

2. Entering ST with only an instruction count leaves the display count unchanged. The initial value of the instruction and display counters is 1.

Responses:

None.

TOGGLE CALL request TC

Purpose:

The TC request controls whether or not the Automatic Return Address Validation feature is enabled.

Format:

```
TC.
```

Usage:

Responses:

1. The TC request toggles the processing of the Return Address Validation, that is if it is enabled, TC will disable it, and if it is disabled, TC will enable it.

2. Automatic Return Address Validation will normally be performed by DDS and in some programs this will result in excessive stops if subroutines return to addresses other than what was pushed on the stack by the CALL. Disabling Return Address Validation will then be desirable.

Responses:

None.

VALUE STOP ON REGISTER request VS

Purpose:

The VS request causes a DDS stop to occur when a register attains a specified value.

Format:

```
VS [,register byte].
```

where:

register	denotes a register and can be either A,B,C,D,E,H,L, or F.
byte	is the value associated with the register.

Usage:

1. The VS command is in effect for all instructions executed under the control of the step command (ST). Upon returning from the last instruction, DDS examines the registers and stops if one of the value stop conditions exists.

2. Up to 5 register value stops can be active at one time. Value stops on bytes, entered with the VSB command, count toward the limit above.

3. To remove a value stop on a register, clear all register value stops by typing VS., and re-enter those that are to remain. Doing so will also clear all byte value stops.

Responses:

VS

This message appears on the error line whenever the value of a register matches one of its stop conditions.

VALUE STOP ON BYTE request VSB

Purpose:

The VSB request causes a IDS stop to occur when a specified byte attains a given value.

Format:

```
VSB[,address,value].
```

where:

address	is the address of the byte to be monitored.
value	is the value being checked for.

Usage:

1. The VSB command is in effect for all instructions executed under the control of the step command (ST). Upon returning from the last instruction DDS examines the addresses specified and stops if one of the value stop conditions exists.

2. Up to 5 byte value stops can be active at one time. Value stops on registers, entered with the VS - command, count toward the limit above.

3. To remove a value stop on. a byte, clear all byte value stops by typing VSB., and re-enter those that are to remain. Doing so will also clear all register value stops.

Responses:

VS

This message appears on the error line whenever the value of a byte matches one of its stop conditions.

VALUE STOP ON REGISTER FAIR request VSP

Purpose:

The VSP request causes a DDS stop to occur when a register-pair attains a specified value.

Format:

```
VSP[,pair,word]
```

where:

pair	designates one of the operational pairs, can be either B, D, or H.
word	is the value associated with the register-pair

Usage:

1. The VSP command is in effect for all instructions executed under control of the step command (ST). Upon returning from the last instruction DDS examines the registers and stops if one of the value stop conditions exists.

2. The contents of word is not reversed. To stop when the register-pair BC attains the value 1234, enter VSP,B,1234.

3. Up to 5 register-pair value stops can be active at one time. Value stops on words, entered with the VSW command, count toward the limit above.

4. To remove, value stop on a register-pair, clear all register-pair value stops by typing VSP., and re-enter those that are to remain. Doing so will also clear all word value stops.

Responses:

VS

This message appears on the error line whenever the value of a register-pair matches one of its stop conditions.

VALUE STOP ON WORD request VSW

Purpose:

The VSW request causes a DDS stop to occur when a word attains a given value.

Format:

```
VSW[,address,value].
```

where:

address	is the address of the word to be monitored
value	is the value being checked for

Usage:

1. The VSW command is in effect for all instructions executed under the control of the step command (ST). Upon returning from the last instruction DDS examines the addresses specified and stops if one of the value stop conditions exist.

2. The value specified is reversed prior to testing, to conform to 8080 conventions. Thus the command VSW,3000,1234 will cause a stop when location 3000 contains 34 and 3001 contains 12.

3. Up to 5 word value stops can be active at one time. Value stops on register pairs, entered with the VSP command, count toward the limit above.

4. To remove a value stop on a word, clear all word value stops by typing VSW., and re-enter those that are to remain. Doing so will also clear all register pair value stops.

Responses:

VS

This message appears on the error line whenever the value of a word matches one of its value stop conditions.