Undocumented Hercules

Table of Contents

Notices

"Linux®" is the registered trademark of Linus Torvalds in the U.S. and other countries.

Introduction

Documenting the undocumented is the object of this manual. Hercules contains many capabilities that are not documented anywhere. This manual is intended to close some of that gap. Where known, differences between the various versions of Hercules and host platforms will be documented.

Three versions of Hercules are in scope for this document:

1. Original Hercules, Hercules, version 3.x;
2. Developer sandbox, Hyperion (Hercules, version 4.x); and
3. Software Development Labs (SDL), SDL Hyperion (also, Hercules 4.x).

Each version has its advantages and disadvantages for any given user. There are additional versions offered for specific contexts. Generally there are few places to go for information on the topics included in this document besides the Hercules code itself. Assume that each capability described in this manual is available in all three versions, *unless otherwise stated*. One can expect Hyperion (2) and (3) to be closer in capability than the original Hercules (1).

The author of SATK and this documents uses Hyperion (2) on OpenSuse Linux for testing.

This document is a work in progress and updates as appropriate will be made.

DIAGNOSE Instruction Usage

Unlike legacy assemblers, ASMA provides a mnemonic for the DIAGNOSE instruction, `DIAG`. The DIAGNOSE instruction uses the register-to-storage format. In assembler source format, `DIAG` is usually coded this way:

```
DIAG Rx,Ry,X'code'
```

The *code* in hexadecimal identifies the function being requested. Registers *Rx* and *Ry* provide information to the function. When a function requires a register pair for one or both of the register operands, *Rx+1* and *Ry+1* also participate in the function.

The DIAGNOSE instruction is a privileged instruction. It will return a condition code indicating success or failure:

- 0 – success of the DIAGNOSE instruction
- 1 – failure of the DIAGNOSE instruction
- 2 or 3 is not returned by Hercules.

In addition to the instruction's success or failure, the DIAGNOSE function may also use one of the instruction's participating registers for a return code. The return code values are dependent upon the function's operation.

A program should check both the condition code and, if used, the return code from the function in determining success or failure.

A Program Interruption Specification Exception may result depending upon the requested function. The program should be prepared to handle this possible situation.

Hercules, in addition to its own usage, provides some emulation of VM's use of the DIAGNOSE instruction. As DIAGNOSE codes are used by SATK, documentation of how they function will be provided in this document. In the case of VM emulation, the documentation of the VM usage may be used, but this document reflects Hercules actual implementation.

DIAGNOSE X'008'

DIAGNOSE instruction code X'008' provides the ability to issue a command to the underlying system and optionally, request a response. On VM the command is a VM command. On Hercules the command is a Hercules command. The instruction is coded as follows:

```
DIAG Rx,Ry,X'008'
```

The code X'008' function is driven by the flag byte provided in the first byte of the *Ry* register.

The flag byte is bit level set of flags. '1' causes the meaning defined, '0' is required, 'x' may be either '0' or '1', and '.' indicates the value is ignored.

Flag Bits	Flag Byte	Meaning
1xx.	X'80'	Reject password in command (ignored by Hercules)
x1x.	X'40'	Provide a command response
xx1.	X'20'	Prompt for password (ignored by Hercules)
...x xxxx	--	Causes a specification exception if any bit is 1

If the flag byte bit 1 is 0, only command execution is requested and only Rx and Ry are used by code X'008'. If the flag byte bit 1 is 1, a command response is also requested and registers Rx+1 and Ry+1 are also used.

Upon successful completion of the Hercules command, register Ry contains a return code. 0 indicates successful execution. Anything other than 0 indicates why the command failed.

When flag bit 1 is 0 or 1:

- *Rx* contains the address of the command to be executed.
- *Ry* byte 0 contains the flag byte.
- *Ry* bytes 1-3 contains the length of the command, Hercules maximum is 256 bytes. On Hercules, if the command length is zero, Hercules places the executing CPU into the **stopped** state.

When flag bit 1 is 1, additionally

- *Rx+1* contains the command response address,
- *Ry+1* contains the length of the command response, Hercules maximum is 256 bytes.

Specification Exception Causes:

Any of the unused bits, bits 3-7, in the flag are set to 1.

Rx and Ry must not be the same.

When flag bit 1 is 1:

- *Rx*, *Rx*+1, *Ry*, and *Ry*+1 must not overlap
- *Rx*, or *Ry* must not be 15 (register wrap around is not supported)

Hercules configuration does not contain `DIAG8CMD enable`.

Condition Codes Returned:

Hercules only returns condition code zero.

Return Codes:

Hercules only sets the *Ry* returns code to zero.

List-Directed Initial Program Load (IPL)

The `ipl` command is usually used to initiate IPL from a device. All that is required is the device address as defined by the Hercules configuration file, for example, `ipl 220`. This type of IPL is well described in various publications, in particular, the relevant *Principle of Operations* manual. Device targeted IPL will not be discussed further here.

List-directed IPL is a much later innovation. It appeared when Linux started to be installed on mainframe systems using the Host Management Console. This simple form of list-directed IPL allowed the Linux installation. More recent versions of the Principle of Operations manuals have more complex descriptions of how the "list" is built, but as of yet none of those are supported by Hercules.

List-directed IPL, as implemented by Hercules, is initiated with the same command, `ipl`, but instead of a device being specified, a host platform file is specified, for example: `ipl path_to_control_file`. That part is well documented within the various web pages supported by the different Hercules versions. What is not documented is how the directory that contains the file is structured nor the contents of the targeted file.

The control file contains text statements that do two things. First, a file within the same directory as the control file is identified and, second, where in absolute storage the file is to be loaded by Hercules. The location is a hex value that starts with a "`0x`" followed by the load address. The following is an example of a list-directed IPL control file:

```
IPLPSW.bin 0x0
IPLPGM1.bin 0x300
```

This results in at least three files within this directory. There could be other files that would be ignored by the list-directed IPL process. The three files would be:

- `IPLPGM1.bin`
- `IPLPSW.bin`
- `pgm1.txt`

This example control file identifies two files within the same directory as the control file itself that are to be loaded into absolute storage. In each case the file is treated as a binary image file, loaded directly into absolute storage. Hercules uses the load address to position the file in storage. The files are loaded in the sequence specified in the control file. `IPLPSW.bin` would be loaded first, followed by `IPLPGM1.bin`.

Following the successful loading of the files specified in the control file, a PSW is fetched by the IPL cpu from locations X'0'-X'7' and program execution starts. And then the fun begins, so to speak. Whatever is located in these absolute addresses is loaded into the active PSW.

Hyperion Linux® Considerations

While on another common platform, a fully qualified file path and name in the `ipl` command is permitted. Linux and some other similar platforms do not allow it. This is a "feature" of how the file is accessed within Hercules using the `realpath()` function. This function treats a fully qualified file path and name as an error condition. Hercules reports the file as non-existent.

The easiest way to get this to work is to change the current working directory to another lower down the directory tree, and use a non-fully qualified name within the ipl command. This allows `realpath()` to find additional directories to add to is argument producing a fully qualified file. Hercules will use this resulting path to perform the IPL.

Assume the control file targeted by the `ipl` command is:

```
/home/harold/SATKREPO/SATK/samples/guide/pgm1/ldipl/pgm1.txt
```

The user could change (before launching Hercules) the current working directory to:

```
cd /home/harold/SATKREPO/SATK/samples/guide/pgm1
```

The ipl command, either typed or part of the Hercules run control file (`.rc`), would be:

```
ipl ldipl/pgm1.txt
```

This change, while tying the `ipl` command to the current working directory, succeeds on Linux with Hyperion.