Table of Contents

Introduction

A Small Mainframe Assembler (ASMA), the `asma.py` tool, creates output intended for loading a bare-metal program into a mainframe compatible system for execution. With the exception of the ASMA list-directed Initial Program Load output format option, used with the Hercules emulator, some additional external content is required to do so.

The context for this document is the structuring of an ASMA assembly image output or a list-directed IPL directory's content as the source for creating a Hercules emulation medium suitable for IPL, eliminating the need for additional user supplied content.

Creation of device media content suitable for use with the mainframe Initial Program Load (IPL) function ties together the tools used to create the content, the use and attributes of the media itself and the needs of the IPL function. Some intermediate process is required to go from the created binary content to the media ready for use in the IPL function. The capabilities of the binary content creation tooling drives much of the requirements for the intermediate process. The more capable the binary creation tools the less the intermediate process needs to do. The tool described in this document provides the intermediate process required to create the IPL-capable medium.

Concepts and lessons learned from the use of the IPL ELF ABI supplement will be incorporated into the new utility, `iplasma.py`.

Definitions

The following terms are used in this document.

install

> The process of creating content and placing it on a real or emulated medium capable of participating in the mainframe Initial Program Load (IPL) function. The installed content may participate directly in the IPL function or be loaded by a bootstrap loader that supports the loading.

bootstrap loader

> An installed bare-metal program that loads a program to which control is passed upon completion of the loading process.

booted program

> An installed bare-metal program loaded into memory by the installed bootstrap loader to which control is passed by the bootstrap loader.

A list-directed IPL directory constitutes a real medium into which a bare-metal program may

be installed. As the direct output of ASMA, ASMA installs, in the context of the above definition, its output into the directory. The process described and documented herein takes the files installed and installs the bare-metal program, or a bootstrap loader with its booted program onto the emulated medium.

Implementation Status

The `iplasma.py` tool is still under development. This document describes existing and planned capabilities. Currently the tool **supports** only:

- FBA DASD IPL medium creation
- A bare-metal program created by ASMA using the ASMA `--gldipl` command line option.
- A bare-metal program created by ASMA using the ASMA `--image` commad line option.

Refer to the section “Assembling Bare-Metal Programs” for details related to program content, macro SATK supplied macro usage, and other considerations.

Specifically **not supported** are:

- Other device types, including CKD
- Bootstrap loader programs on any medium.

SATK provides as of yet no boot strap loader programs created by ASMA. No bootstrap loaders yet exist for use with the `iplasma.py` tool

References

- *IPL ELF ABI Supplement*, in SATK directory `doc/iplelf`.

Machine Interface

The machine interface remains largely the same as described in the *IPL ELF ABI Supplement*. What differs is the source of the IPL record content, namely an image file or the list-directed IPL directory as opposed to an ELF executable file. The primary list-directed IPL file in the list-directed IPL directory identifies the binary image data and its starting absolute address in storage. Each file of binary image data in a list-directed IPL directory directly corresponds to the ASMA region name associated with it.

The following table identifies the expected types of logical IPL records and their content. The rows in blue are part of the standard IPL function. SATK has enhanced the basic process by providing the optional capability of low-storage initialization. In some cases the IPL process itself is not readily built to load the desired program and instead an intermediate bootstrap loader is used for the loading of the actual program. The enhanced records that are part of the SATK process are identified in yellow.

IPL Record	Status	Program Specific Data	Device Specific Data
0	required	Initial PSW passing control to the program or boot loader	First two explicit channel command words (CCWs) of the I/O command chain
1	optional		Remainder of the I/O command chain
2-n	required	Bare-metal program / boot loader	
n+1	optional	Low storage initialization	
n+2	optional	Program load information	Device attributes
variable	optional	additional boot loader records	

All of the content identified above is placed onto the IPL capable media by the ASMA IPL process. The colored rows (blue and yellow, if present) are loaded into memory by the IPL function itself. The records in the white row are brought into memory by the previously loaded bootstrap loader program.

Because the IPL function loads and uses memory, implementation requires an approach to how memory will be utilized by the function in any given case. The hardware defines use of certain specific locations by the function. These are small areas and any other locations may be used as the implementation elects. For the purpose of debugging it is desirable to separate areas so that they do not overlap, ensuring data at one point is not lost by the process.

In addition, the process must know how to locate specific records with regards to specific media. These locations of the records on the media and their memory resident locations become embedded in IPL record 1, where most of the channel program used by the IPL

function (blue rows above) is located. IPL record 0, must of course know where IPL record 1 will reside so it too can be brought into memory.

For each type of record, the process must define and establish:

- Binary content,
- Memory resident addresses, and
- IPL media resident location.

Because the binary content of IPL records reference memory resident addresses and media resident locations, the simplest approach is to establish fixed locations for each. The most complex case allows all of these variables to change. Neither is ideal. The IPL ELF Supplement addressed some of the variations by establishing fixed locations on the IPL medium for the logical records.

The hardware itself has some constraints on the variables. The IPL function uses Format 0 Channel Command Words, limiting all I/O operations to the first 16 megabytes of memory. This means that, initially, 24-bit CPU address mode is adequate for entering either the program or bootstrap loader. In fact, either must reside below the 16-megabyte boundary. Format 0 CCW address wrapping will not allow any location to be loaded from a device above this boundary.

Another consideration relates to device media constraints. A program brought into memory from a Count-Key-Data (CKD) Direct-Access-Storage Device (DASD) can not exceed the length of a single track, the maximum length of logical IPL records. Fixed-Block-Architecture (FBA) DASD, as emulated by Hercules, can only read or write entire sectors. This means the minimum record size for an FBA device is 512 bytes. For FBA devices, information can be merged into one or more sequential sectors and read as a group. This is typically done for logical IPL records 0 and 1.

All of these issues come together when logical IPL record 1 is created.

IPL ELF Lessons Learned

From development of the *IPL ELF ABI Supplement*, found at "`doc/iplelf/IPL ELF ABI Supplement.odt`" and implemented by `tools/iplmed.py`, a number of lessons were learned.

Media Architectures

The `iplmed.py` tool supported five forms of output as emulated by the Hercules emulator a:

- list-directed IPL directory
- sequential card deck
- sequential tape volume
- direct access CKD DASD volume
- direct access FBA DASD volume

The list-directed IPL directory is itself created directly by ASMA in the context of this document. Special tooling is not required for its use in the context of the IPL function for an ASMA image created with this output option. And in that context, the list-directed IPL directory is the input to the IPL medium creation rather than an output of it for an ASMA image.

This really leaves the remaining IPL volumes as the output of the processes described here. Another lesson learned is that the way in which the emulated medium is accessed has a major impact on the processes involved in creation of the IPL records and bootstrap loader records. In essence, the tooling for sequentially accessed volumes is different than the tooling required to create directly accessed volumes.

Reusable Bootstrap Loaders

As originally conceived, the IPL ELF ABI Supplement expected a bootstrap loader to be assembled with the bare-metal program. Experience showed that this was unnecessary and undesirable. The solution adopted by the `iplmed.py` tool used two separate ELF executable files as input: one being the bare-metal program being loaded and the other the bootstrap loader.

Reuse of the same boot strap loader with ASMA is also desirable. In this context two input files will be used as input to the medium creation tool. One directory contains the bare-metal program. The other optional directory contains the bootstrap loader.

Media Preparation Data

The *IPL ELF ABI Supplement* allows for the creation of medium preparation processor specific data without explicit definition of the contents. This data is defined as separate from the volume identification. Media preparation data is really an assist to a bootstrap loader.

The use of IPL media by the ASMA process standardizes the use of preparation data by the bootstrap loader. A new standard record, `LOD1`, is used to contain boot loader specific information. The `LOD1` record is considered part of the set of IPL records.

Logical IPL record 4, the `LOD1` record, has itself some unique aspects. Does the IPL function need to bring it into memory? It depends upon the needs of the bootstrap loader and the nature of the IPL medium. If the `LOD1` record only contains information related to the program brought into memory by the bootstrap loader, then the bootstrap loader may read it. If, however, the `LOD1` record contains information related to the medium required by the bootstrap loader when accessing the medium, then the IPL function must load it for the bootstrap loader. Without this information, then potentially the bootstrap loader may not be able to read the `LOD1` record itself.

The latter case exists for DASD devices, but not for tape or card media. Why? For both types of DASD devices, one of the I/O operations required to read from the DASD device is the establishment of the area being accessed, the DASD “extent”. During the IPL function, the hardware automatically makes the entire device the accessed extent. However, the bootstrap loader or program must actually inform the DASD of the extent before it can read from the device. The values supplied in this information are controlled by the size of the device. Values in excess of the device size are rejected and the I/O operations fail. For a DASD accessing bootstrap loader, it must either be told this information, or it must figure it out for itself. “Figuring it out” increases the complexity of both the medium creation and bootstrap processes. Hercules can complicate this further because non-standard DASD sizes are possible. In this case only the process creating the DASD medium knows the actual size of the volume.

Bootstrap Loader Interface

The implementation of the *IPL ELF ABI Supplement* by `iplmed.py` utilized the ELF executable file for its bootstrap loader binary content. A bootstrap loader interface structure communicates to the `iplmed.py` tool the bootstrap loader's capabilities and attributes and was updated by the `iplmed.py` tool to communicate IPL medium and record information to the bootstrap loader. This made sense from the perspective of a generic process for which the IPL medium record contents were being constructed for the bootstrap loader about which the tool could make no assumptions.

This assembled interface proved to be overkill in practice after the ability to reuse a bootstrap loader became an option. Tool capabilities have advanced as well. SATK can now add dynamically files to its `PYTHONPATH` directory list. A Python module residing in the list-directed IPL directory will support this function for SATK supplied bootstrap loaders needing to

communicate with the preparation process.

The `LOD1` record replaces the interface for communication with the bootstrap loader by the media preparation process.

The `__boot__.py` file within a bootstrap loader list-directed IPL directory communicates its capabilities to the `iplasma.py` tool. See the `iplasma.py` source for details on the contents of this file.

Stream vs. Directed Boot Program Records

Two types of sequentially accessed bootstrap loader records were supported by `iplmed.py` depending upon the bootstrap loader capabilities:

- stream records, and
- directed records.

Stream records use an initial memory starting address and are loaded sequentially into memory. Directed records, as the name was intended to imply, contain a memory address in its first four bytes and the remainder of the record is loaded into memory at that location. The concept of the directed records was inspired by the way an object deck is loaded using the text records address as the starting point for the records content.

At first blush, it would seem that stream records make sense for use with the ASMA created list-directed IPL directory. That would be true if only one region would be found in the directory. By design multiple files are expected to be found in a list-directed IPL directory, one for each ASMA assembled region. All bootstrap loaders standardize on the directed IPL record format.

Some bootstrap loaders may require a length in bytes 4 and 5 of the record. This is true for any loader that is unable to dictate the content length via the physical record length. FBA devices have this constraint.

Object Deck Records

ASMA supports the creation of an object deck via the output option `--object`. It is restricted for use with sequential media bootstrap loaders supporting card decks or emulated tape volumes.

Assembling Bare-metal Programs

--image File Considerations

Regions should typically not be used, unless the bare-metal program is specifically designed for use of regions within an image file. See the *ASMA.odt* or *ASMA.pdf* manuals for a description how to use an image file with regions.

The remainder of this section assumes only one region exists in the bare-metal program assembly. This means only one region is initiated by the first `START` assembler directive, named or unnamed. No additional `START` assembler directives will exist in the assembly.

The first 8 bytes of the image must contain an IPL PSW used to enter the program. Regardless of where the image is loaded into memory, this IPL PSW will be used for program entry in logical IPL record 0. Assemble the desired PSW using either an explicit PSW directive, or one controlled by the current `XMODE PSW` setting, or as a series of `DC` operands.

Use a `START` assembler directive with an explicit value of zero, or omit the first operand to cause the assembly to start at address 0. The `iplasma.py --load` command-line argument may be allowed to default to 0. In this case, the assigned storage area may be initialized as desired directly by the bare-metal program without the need of IPL Record 3. Use the `TRAP64` and `TRAP128` macros to prepare the assigned areas. An `ORG` is required preceding the macros to correctly position the trap PSW's within the image. See the *SATK.pdf* or *SATK.odt* manuals in the `doc` directory for details concerning these macros.

No mechanism exists for ASMA to communicate to the `iplasma.py` utility if the program was assembled at a starting address other than zero. If this is the case, the `iplasma.py` utility will require the `--load` command-line argument to know where the bare-metal program is to be loaded. The explicit address from the `START` directive must be supplied as the `--load` command-line argument's value. In the case the `START` directive was not set to zero, any initialization of the assigned storage must be done by the bare-metal program. The IPL function will not do so.

Unless the program in the image file understands the requirement for processing a region within the image, `START` directives initiating a new region must be avoided.

--gldipl List-Directed IPL Considerations

The following SATK supplied ASMA macros are provided to assist with creating IPL record content.

Because the list-directed IPL directory control file contains load address information for each region, multiple regions may be used as required. Unless the first region acts like an image file, there will be at least two regions:

- one for the IPL PSW content and
- another for the program.

For IPL Record 0 use a `START` directive initiating a region, by default `IPLPSW`, and a differently named CSECT at address 0. Assemble the desired PSW in this region using either an explicit PSW directive, or one controlled by the current XMODE PSW setting, or as a series of `DC` operands. Leave this region (and CSECT) by using another `START` directive for the remainder of the program.

For IPL Record 3 use either the `ASALOAD` and optionally the `ASAIPL` macros. These macros will create the assigned storage initialization region `ASAREGN`.

See the *SATK.pdf* or *SATK.odt* manuals in the `doc` directory for details concerning these macros.

List-Directed IPL Directory Structure

The `iplasma.py` tool uses one or more list-directory IPL directories as input. The input is used to install either:

- a bare-metal program (using one directory) or
- a bootstrap loader and its booted program (using a directory for each)

onto an emulated install medium.

The command line options are used as follows in each case:

Option	Optional	Default Filename	Bare-Metal Program	Bootstrap Loader and Booted Program
`--dtype`	no	none	required	required
`--medium`	no	none	required	required
`source`	no	none	the bare-metal program	the booted program
`--load`	yes	--	optional for -f image	optional for -f image
`--boot`	yes	none	--	the bootstrap loader
`--psw`	yes	`IPLPSW.bin`	IPL record 0 PSW region	Required (or default)
`--asa`	yes	`ASAREGN.bin`	Assigned storage area region	Required (or default)
`--bare`	yes	`PROGRAM.bin`	Program region	Required (or default)
`--recl`	yes	--	--	Defaults for `--dtype`
`--arch`	yes	--	--	Optional when supported
`--traps`	yes	--	--	Optional when supported

The list-directed IPL directory is implied by the path to the designated file name of the option. The file name is that of the control file. The other files of the directory contain the installed content. They result from a ASMA defined region, created by the second operand of a `START` directive. For a bare-metal program, various files may be identified. The bootstrap loader requires the default file name if it supplies the content in its directory. The tool accepts non-standard names for a bare-metal program.

The IPL function depends upon a sequence of input/output channel commands. These commands tie the physical location of the content on the medium and its memory resident locations. The IPL function is embodied in IPL Record 0 and if needed IPL Record 1. IPL Record 0 contains three distinct pieces of information:

- the IPL PSW used to pass control to the loaded program
- A CCW, frequently used to read IPL Record 1 into memory, and
- an optional CCW that causes the remainder of the operations to occur, usually based

upon the content of IPL Record 1, usually a transfer-in-channel command.

Both IPL Records 0 and 1 are built by the tool itself. The only explicit information used for these records from the list-directed IPL directory is the IPL PSW.

See command-line option `--psw` for details.

Bare-metal Program Content

The `iplasma.py` tool emulates the IPL function of a list-directed IPL directory by loading in the sequence of the control file, each separate region represented by a file in the directory excluding:

- the IPL PSW binary file, used by logical IPL Record 0, if present, and
- the assigned storage initialization from logical IPL Record 3, if present.

All other files are considered part of the “binary program” and are installed on the IPL medium and loaded by the generated IPL Record 1.

These other files become part of a load list used to construct the read sequences required by the IPL medium to read the installed content into memory. This includes the assigned storage initialization region always loaded last. All bare-metal program content must load within the first 16M of memory. Otherwise, a bootstrap loader is required using Format-1 CCW's (and channel subsystem input/output operations) to load content above the 16M boundary.

The IPL medium content built by the `iplasma.py` tool does not restrict the bare-metal program to a single physical medium record, but rather generates read sequences as required to load the bare-metal program's regions. The read sequences must reside in a single logical IPL Record 1. The IPL medium will constrain the size of the logical IPL Record 1, but the design limit for the bare metal programs is much larger.

LOD1 Record Content

The `LOD1` Record must be loaded on an eight-byte memory address boundary. Primarily this is because it contains the Program Status Word (PSW) used to enter the bare-metal program brought into memory by the bootstrap loader. All numeric fields are unsigned binary. Fields altered by the bootstrap loader are identified in blue and must be initialized to binary zeros. Reserved fields are highlighted in gray and must also be initialized to binary zeros.

Disp. (Dec)	Disp. (Hex)	Length	Address	Description
+0	+0	4	000240	Record identification: EBCDIC `C'LOD1'` or `X'D3D6C4F1'`
+4	+4	1	000244	IPL Medium Type: 1. X'02' – Directed records contain two-byte length of content 2. X'04' – Card 3. X'08' – Tape 4. X'10' – FBA DASD (see field at +24) 5. X'20' – CKD DASD (see fields +28 - +34) 6. X'40' – ECKD DASD (to be documented)
+5	+5	1	000245	Bootstrap Loader flags: X'80' – change architecture before entering bare-metal program X'40' – set 64-bit address mode when entering bare-metal program X'20' – set trap new PSW's before entering program X'01' – Use Format-1 CCW's and 31-bit addressing mode during loading (default Format-0 and 24-bit addressing)
+6	+6	2	000246	Maximum length of bootstrap directed record in bytes
+8	+8	4	000248	Cumulative length of bare-metal program content on medium
+12	+C	4	00024C	Cumulative length of bare-metal program content loaded (zero)
+16	+10	8	000250	Booted program entry Program Status Word (from `IPLPSW.bin` or `ASAREGN.bin`) loaded using LPSW instruction
+24	+18	4	000258	FBA DASD bare-metal program starting physical sector
+28	+1C	2	00025C	CKD DASD bare-metal program starting cylinder number
+30	+1E	2	00025E	CKD DASD bare-metal program starting track (head) number
+32	+20	1	000260	CKD DASD bare-metal program starting record number
+33	+21	1	000261	CKD DASD number of bootstrap records per track
+32	+22	2	000262	CKD DASD maximum cylinder number
+34	+24	2	000264	CKD DASD maximum track (head) number
+36	+26	2	000266	Reserved, must be zero
+38	+28	4	000268	IPL device identification from IPL function
+42	+2C	4	00026C	Bootstrap loader input/output area address, if required
+46	+30	16	000270	128-bit PSW required for 64-bit addressing mode loaded by LPSWE instruction.

Disp. (Dec)	Disp. (Hex)	Length	Address	Description
+62	+2A	16	000280 - 00028F	Reserved, must be zero

For FBA DASD, the LOD1 record resides in an entire sector. The LOD1 record itself is placed at displacement X'40'-X'8F' within the sector. This allows the entire sector to be read into storage at X'200', clearing the Hercules IPL parameter data stored at X'200'-X'23F' while placing the LOD1 record at X'240'-X'28F'. The remaining bytes of the 512 byte sector reside at X'290'-'X3FF', the area reserved for the LOD1 sector.

Memory Usage and Program Entry

The following table describes the general usage of memory for all SATK supplied bootstrap loaders. Addresses are absolute. Source refers to the default ASMA region name found in the list-directed IPL directory.

Bare-Metal Program Usage

This section describes the usage of memory by the IPL function when loading a bare-metal program directly. “EOP” means the end of the bare-metal program. “EOM” means the end of memory.

The following table illustrates the sequence in memory where each component resides. First is the absolute assigned storage locations used by the IPL function itself. Next will reside the bare-metal program, followed by the IPL Record 1 that actually reads the bare-metal program. This sequence is possible because `iplasma.py` actually knows the size of the bare-metal program. With that knowledge it can ensure the additional data required to read the program do not conflict with the actual locations in which the bare-metal program resides.

Start	End	Length	Source	Notes	Description
000000	000017	24 - X'018'	IPLPSW.bin or iplasm.py	1, 4	IPL Record 0 (IPL0)
000000	0001FF	512 - X'200'	ASAREGN.bin	1, 4	IPL Record 3 (IPL3)
000200	00023F	64 - X'40'	none	5	Reserved for Hercules IPL parameter data
000240	EOP	varies	varies	6	Bare-metal program regions or image (IPL2)
EOP+	EOP+X	varies	iplasm.py	2, 3, 4	IPL Record 1 (IPL1)
EOP+X+1	EOM	varies	none		Available for bare-metal program use

Note 1: If low-storage is not initialized with an area created by the SATK `ASALOAD` macro, a region containing the IPL PSW, named by default `IPLPSW`, is required.

Note 2: IPL Record 1 is loaded on a 4-byte boundary following the physical end of the bare-metal program.

Note 3: The actual length may vary depending upon the IPL medium and whether the ASA area is loaded during the IPL function.

Note 4: Once control has been passed to the bare-metal program, the areas previously used by the IPL function are available for the program's own use.

Note 5: Hercules IPL parameter data resides in the 16 32-bit registers and if present is visible to the program receiving control following the IPL function. The data must be saved by the program loaded by the IPL function before any register content is altered, that is before a base register is established. SATK standardizes on location X'200' for storing of the Hercules IPL parameter data. Any 64-byte area that starts before address X'1000' may actually be

used. It is entirely up to the bare-metal program to accommodate Hercules IPL parameters or not.

Note 6: The length of the supported physical records for a given medium type dictates the maximum size of the program that can be loaded without a bootstrap program. The record size applies the logical IPL Record 1, containing the input/output operations reading the program content.

- Card – limited to ten successive card images or 800 (X'320') bytes for the bootstrap loader due to a single IPL Record 1 of eighty bytes. Ten CCW's that read 80-bytes each makes the limit 800.
- CKD DASD – maximum physical record size varies based upon device type:
    1. 2305 – 14136,
    2. 2311 – 3625,
    3. 2314 – 7294,
    4. 3330 – 13030,
    5. 3340 – 8368,
    6. 3350 – 19069,
    7. 3375 – 35616,
    8. 3380 – 47476,
    9. 3390 – 56664,
    10. 9345 – 46456.
- FBA DASD – IPL Record 1 is merged with IPL Record 0 into the first sector of the volume. The read sequences (20 bytes each) are constrained to the 488 bytes available for IPL Record 1 in this sector. The bare-metal program is limited to 24 read sequences. However, each read sequence can read 127 512-byte sectors for a maximum size of 65,024 bytes per read sequence (totaling 1,560,576 bytes – X'17D000').
- Tape – 65,535 without data chaining.

Bare-Metal Program Entry

Control is passed to the bare-metal program via the IPL function itself. The Program Status Word (PSW), located at addresses 0 through 7, inclusive, is loaded and control is passed to the bare-metal program by the CPU with the state dictated in the PSW.

The system is in the state prescribed by its *Principles of Operation* manual following the IPL function with the following possible exception on the Hercules emulator.

The Hercules emulator accepts IPL parameters as a character string in its `IPL` commands. This data is presented to the program in general registers 0-15. As many registers as are

needed to contain the data are initialized with it. Following the last character of the IPL parameters, binary zeros are present in the registers not targeted with any data.

If the assigned storage area is initialized, the content will be present in addresses X'008' through X'1FF'. The first eight bytes of the assigned storage area file content are replaced by the required IPL PSW.

IPL PSW Source

The IPL function itself and the IPL records created by the tool dictates the sequence by which memory is loaded from the assembled regions, as follows:

- the IPL PSW (IPL Record 0) always loaded at address 0, default region file `IPLPSW.bin`,
- the bare-metal program (IPL Record 1), default region file `PROGRAM.bin` and last
- the assigned storage area initialization region (IPL Record 3), default region file `ASAREGN.bin`.

If all three regions are placed at address 0, the last loaded region dictates the IPL PSW loaded into the CPU. However, the priority for the source is the reverse of how the IPL function operates. If an IPL PSW is explicitly identified it takes precedence of the other two potential sources. And, the bare-metal program itself takes precedence over the assigned storage area. If a higher priority source is loaded before a lower priority source at address 0, the content of the lower priority source will be modified to use the higher priority source's first eight bytes. This only applies when sources are loaded at address 0.

If no source is loaded at address 0 from the list-directed IPL directory, an IPL PSW will be manufactured to enter the bare-metal program at its starting location. The constructed PSW is, in hex: `0008 0000 00xx xxxx`, where the “x” is the entry address. For systems capable of more than one addressing mode this means the program is entered in 24-bit address mode. If a basic-control mode PSW is requested it takes the form: `0000 0000 00xx xxxx`.

Bootstrap Loader Usage

This section describes how memory is used with a bootstrap loader. The bootstrap loader is itself brought into memory by the IPL function. Everything stated in the previous section “Bare-Metal Program Usage” and “Bare-Metal Program Entry” applies to the bootstrap loader.

For purposes of distinction the “bare-metal” program is designated the “booted-program” in this section.

`EOP` means the end of the booted program. `EOM` means the end of memory. `EIO` means bootstrap loader input/output area, if used.

In general, the first 8,192 bytes of memory are reserved for the bootstrap loader with the booted program starting at or beyond location X'2000'.

Start	End	Length	Source	Notes	Description
000000	000017	24 - X'017'	IPLPSW.bin	1, 4, 11	IPL Record 0 (IPL0)
000000	0001FF	512 - X'200'	ASAREGN.bin	1, 4	IPL Record 3 (IPL3)
000000	001FFF	8,092 - X'2000'	none	4	Reserved for bootstrap program usage
000200	00023F	64 - X'40'	none	9	Reserved for Hercules IPL parameter data
000240	0003FF	448 - X'1C0'	iplasm.py	4, 8, 10	Reserved for IPL Record 4 (LOD1)
000400	0001FF	3,072 - X'C00'	PROGRAM.bin	3, 4, 6	IPL Record 2 for bootstrap loader
002000	EOP	variable	PROGRAM.bin	2, 3, 7	Booted program loaded by bootstrap loader
EOP+	EOP+X	varies	iplasm.py	4	IPL Record 1 (IPL1)
EOP+X+1	EIO	varies		4, 6	Optional I/O area used by bootstrap loader

Note 1: If low-storage is not initialized with an area created by the SATK ASALOAD macro, a region containing the IPL PSW, named by default IPLPSW, is required.

Note 2: The booted program is loaded from the bootstrap records on the IPL medium.

Note 3: The sources for the bootstrap loader and booted program may have the same name but always are found in different directories, inferred from the command-line arguments of the IPL medium creation tool.

Note 4: This area is available for use by the booted program upon entry.

Note 6: The entire area from X'000240'-X'001FFF' is available for use of the bootstrap loader. FBA DASD requires an input/output area for support of bootstrap records using the directed format. The Hercules FBA DASD driver does not support data chaining or skipping. It therefore can not use self-modifying channel programs as is possible with other emulated devices. The input/output area location is provided by the LOD1 record.

Note 7: IPL medium physical record constraints identified in Note 6 of the section “Bare-Metal Program Usage” apply to the size of a bootstrap loader. The bootstrap loader uses the IPL function to become resident and receive control. The same constraints apply to bootstrap loader directed records making some record sized invalid for some devices.

Note 8: When using an FBA DASD device, the Hercules FBA DASD driver requires reading (and writing) of entire sectors. This results from the lack of data chaining support within the driver. It is expected that the sector containing the LOD1 record on the FBA DASD device will be read into memory starting at X'200' and that the actual LOD1 record content will start at byte 64 of the sector. This positions the content in memory at its expected starting location.

Note 9: Hercules IPL parameter data resides in the 16 32-bit registers and if present is visible to the program receiving control following the IPL function. The data must be saved by the bootstrap loader program before any register content is altered, that is before a base register is established.

Note 10: Some content of the `LOD1` record may be useful to the bare-metal program, particularly in cases where IPL device attributes are provided. This eliminates the need for the bare-metal program to figure out the nature of the IPL device. It is for this reason the `LOD1` record is provided at a standard memory location.

Note 11: When a bootstrap loader supports architecture change, a 128-bit PSW may be defined within the booted program's list-directed IPL directory. It is not used for IPL Record 0, but will be placed in the `LOD1` record and used by the bootstrap loader.

Booted Program Entry

To be supplied

FBA DASD Structure

FBA DASD volumes are composed of 512-byte sectors. All data is written to and from the sectors.

Sectors usage is as follows when a boot strap loader is not used:

Sectors	image	ld	Description
1	Y	Y	IPL Records 0 and 1 combined into a single sector
1	Y	Y	Reserved for volume descriptor information
1	N	O	IPL Record 3, assigned storage area initialization
variable	Y	Y	IPL Record 2, bare-metal program

Sectors usage is as follows when a boot strap loader is used:

Sectors	image	ld	object	boot	Description
1	N	N	N	Y	IPL Records 0 and 1 combined into a single sector
1	Y	Y	Y	Y	Reserved for volume descriptor information
1	N	N	N	O	IPL Record 3, assigned storage area initialization
1	N	N	N	O	IPL Record 4, the LOD1 record
variable	N	N	N	Y	IPL Record 2, bootstrap loader
variable	Y	Y	Y	Y	Booted program bootstrap loader records

Actual starting sector numbers will vary based upon the presence of optional volume content and sectors required to contain a bare-metal program (or bootstrap loader) and a booted program, if any.

iplasma.py

The tool at `tools/iplasma.py` in the `SATK` directory prepares an IPL medium for use with any platform that supports use of the emulated volume formats. The Hercules emulator does.

The general command-line format for the tool is:

```
iplasma.py  [ options ... ] source
```

The `source` positional argument is the path and file name of the input file. The file is expected to have been created by the ASMA tool, `asma.py`. See `--format` option for how to identify the type of input being processed.

-f/--format

The input format of the `source` positional argument. Three input formats are recognized:

- `image` – the source option is an image file created by ASMA. Identify the same file for the `source` option as was used with the ASMA `--image` option when the bare-metal program was assembled.
- `ld` – the source option is a list-directed IPL control file created by ASMA. Identify the same file for the `source` option as was used with the ASMA `--gldipl` option when the bare-metal program was assembled.
- `object` – the source option is an absolute load deck created by ASMA. Identify the same file for the `source` option as was used with the ASMA `--object` option when the bare-metal program was assembled. This option also requires the `--boot` option.

Defaults to `image`.

-v/--verbose

Enable verbose messaging during processing.

Image File Input Related Options

-l/--load ADDRESS

Identifies where the image file is to loaded into memory. This option defaults to an address of 0. Provides the location for IPL Record 1.

List-Directed IPL Input Related Options

The bare-metal program is defined by all program regions defined in the identified list-directed IPL directory control file that are **not** identified by the `--noload`, `--asa` or `--psw` options.

-n/--noload FILENAME

Identify any list-directed IPL region to be ignored during creation of the IPL medium. Multiple occurrences of this option are supported. To forcibly ignore the default `--asa` region or `--psw` region, the region name must be identified with this option.

--psw FILENAME|bc|ec

Identifies the format or region in the list-directed IPL directory containing the bare-metal program's IPL PSW. A region overrides an IPL PSW found in the `--asa` region. Defaults to `IPLPSW.bin`. Ignored if the binary region file does not exist in the directory or is included in the `--noload` option.

If the characters `bc` or `ec` are used, a 64-bit PSW is created in the requested format using the starting address of the bare-metal program as the PSW instruction address. In this case, `ec` is the default.

The PSW resulting from this option is used in the creation of IPL Record 0.

See the section “Bare-Metal Program Entry” for details.

--asa FILENAME

Identifies the region in the list-directed IPL directory containing the assigned storage area initialization, including the bare-metal program's IPL PSW if not overridden by the `--psw` option. Defaults to `ASALOAD.bin`. Ignored if the binary region file does not exist in the directory or if included in the `--noload` option. The content of this file is used for creation of IPL Record 3.

Bootstrap Loader Input Command-Line Options

The `--boot` option causes the bare-metal program to be booted by the specified bootstrap loader program rather than being directly brought into memory by the IPL function. This option is required if the bare-metal program is identified by the `--object` option.

Presently no SATK standard bootstrap loaders exist for use with these options. Such are planned for future development.

-b/--boot FILEPATH

Identifies the list-directed IPL control file of the bootstrap loader. Bootstrap loaders are required to be created using the ASMA `--gldipl` option from the `IPLPSW.bin`,

`ASALOAD.bin` and `PROGRAM.bin` binary region file names.

If SATK provides a bootstrap loader for the requested device and bare-metal program format, it will be used when the option is omitted and a bootstrap loader is required.

If specified the bootstrap loader will always be used.

A bootstrap loader is required if the bare-metal program resides at any location higher than X'FFFFFF' or the sequences required to load the bare-metal program exceeds the IPL medium's maximum IPL record 1 length. Bare-metal content residing above X'7FFFFFFF' must be relocated after loading.

Use the same value for this argument as was specified for the ASMA `--gldipl` option when the bootstrap program was assembled.

--arch ACTION

If supported by the bootstrap loader, causes the bootstrap loader to change architectures before entering the bare-metal program if action is `change`. If action is `64` also causes entry to the bare-metal program in 64-bit addressing mode. This option influences the LOD1 bootstrap loader flags.

--traps

If supported by the bootstrap loader, causes the bootstrap loader to set new 64-bit PSW traps before entering bare-metal program.

Depending upon the presence or absence of the `--arch` option, 128-bit trap PSW's will also be established.

This option influences the LOD1 bootstrap loader flags.

Output IPL Medium Command-Line Options

-d/--dtype DTYPE

Describes the device type being created. An explicit `DTYPE` value may be used or a generic value that implies a specific value.

Medium	Generic	Supported DTYPE Values
Cards	`CARD` -> `3525`	`3525`
CKD DASD	`CKD` -> `3330`	`2305`, `2311`, `2314`, `3330`, `3340`, `3350`, `3380`, `3390`, `9345`
FBA DASD	`FBA` -> `3310`	`0671`, `0671-04`, `3310`, `3370`, `3370-2`, `9332`, `9332-600`, `9313`, `9335`, `9336`, `9336-20`
Tape	`TAPE` -> `3420`	`3410`, `3420`, `3422`, `3430`, `3480`, `3490`, `3590`, `8809`, `9347`

Currently only FBA DASD device types are supported.

-m/--medium FILEPATH

Specifies the file and path to the emulated IPL medium created by the tool. All emulated media types are supported by the Hercules emulator. Depending upon the format, the medium may be supported directly by other platforms.

-r/--recl SIZE

Describes the length of the directed bootstrap records in bytes (including the four-byte address field). Must be compatible with the record sizes supported by the device and selected bootstrap loader. If omitted, a default record size for the supported device will be used.

-s/--size OPTION

Defines the sizing of an output CKD DASD or FBA DASD volume. The option is ignored for other device types. In each case all of the content required to IPL or boot the bare-metal program is written. And in each case the DASD volume will have the standard attributes of the `--dtype` device type. What differs is the physical file system size of the emulated volume and the response to certain channel commands, for example, the `READ DEVICE CHARACTERISTICS` command.

Three values are supported.

- `mini` - The smallest DASD volume supported by Hercules is created. `mini` is the default.
- `comp` – The smallest DASD volume required to allow Hercules compression of the volume is created. The output volume is eligible for compression by a Hercules utility. The emulated volume as created by `iplasma.py` is itself not compressed.
- `std` – A full size DASD volume as specified by the `--dtype` options is created. A `std` volume is also compression eligible.

The size relationship between the different options is:

`mini` **<=** `comp` **<=** `std`.