Table of Contents

Notices

z/Architecture is a registered trademark of International Business Machines Corporation.

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 (IPL) 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 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 when the binary content is created using ASMA.

Concepts and lessons learned from the use of the IPL ELF ABI supplement will be incorporated into the new utility, `iplasma.py`. For details see “Appendix A”.

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 IPL function. The installed content may participate directly in the IPL function or be loaded by a boot loader that supports the loading.

boot 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 boot loader to which control is passed by the boot loader.

A list-directed IPL directory (LDID) constitutes a structure from which a bare-metal program may be installed. As the direct output of ASMA, ASMA places, in the context of the above definition, its output into the directory. The process described and documented herein takes the input files and installs the bare-metal program, or a boot 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 for all Hercules target architectures.
- Boot Loader Services available with the boot loader (all Hercules target architectures).
- 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` command line option.

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

Specifically **not supported** are:

- Other device types, including CKD

SATK provides a FBA boot loader program as part of the Guide. Each is created by ASMA and usable with the `iplasma.py` tool.

References

- *IPL ELF ABI Supplement*, in SATK directory `doc/iplelf`.
- *SATK User Guide*, in SATK as `doc/Guide.odt` or `doc/Guide.pdf`.
- *Boot Loader Services*, in SATK as `doc/BLS.odt` or `doc/BLS.pdf`.

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 LDID as opposed to an ELF executable file. The primary list-directed IPL file in the LDID identifies the binary image data and its starting absolute address in storage. Each file of binary image data in a LDID 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 and supplying of booted program communication. In some cases the IPL process itself is not readily built to load the desired program and instead an intermediate boot 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 bare-metal 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 (1 or more)	required	Bare-metal program / boot loader	
3	optional	Low storage initialization	
4	optional	Booted program 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 a previously loaded boot loader program.

Because the IPL function loads and uses memory, the implementation requires an approach to how memory will be utilized by the IPL 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. The 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 and yellow 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 of the IPL function to the first 16 megabytes of memory. This means that, initially, 24-bit CPU address mode is adequate for entering either a bare-metal program or a boot 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 the logical IPL records. Fixed-Block-Architecture (FBA) DASD, as emulated by Hercules, can only read or write entire sectors, the last sector being the only exception, 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. This record contains the remainder of the channel program used to load the bare-metal program, including when the bare-metal program is a boot loader.

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,
- 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 at which address the program image was assembled. If this is needed, 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 this case the `START` directive is not set to zero. Without a separate region, `iplasma.py` can not initialize the assigned storage by creating IPL Record 3. The bare-metal program must perform any required initialization.

Unless the program in the image file understands the requirement for processing more than one region within the image, only one `START` directive may be used in the assembly.

--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, 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 the `ASALOAD` and 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 LDID's as input. The input is used to install either:

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

onto an emulated medium.

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

Option	Optional	Default Filename	Bare-Metal Program	Boot Loader and Booted Program
`source`	no	PROGRAM.bin	the bare-metal program	the booted program
`--boot`	--	none	--	the boot loader
`--load`	yes	`--`	optional for -f image bare-metal program	optional for -f image booted program
`--psw`	yes	`IPLPSW.bin`	Bare-metal IPL record 0	The booted program's PSW
`--asa`	yes	`ASAREGN.bin`	Assigned storage area region	Ignored for a booted program
`--lpsw`	yes	`IPLPSW.bin`	--	Boot loader IPL record 0
`--lasa`	yes	`ASAREGN.bin`	--	Optional for boot loader
`--dtype`	no	none	required	required
`--medium`	no	none	required	required
`--recl`	yes	`--`	`--`	Defaults for `--dtype`

The LDID 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 a START directive containing a region name. For a bare-metal program, various files may be identified. The boot loader may override 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 input/output operations to occur, usually based upon the content of IPL Record 1, and 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 is the IPL PSW from the LDID or image file.

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

Bare-metal Program Content

The `iplasma.py` tool emulates the IPL function of a LDID 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.

Only one other file within the LDID is considered to be the loaded program. It must fit within the maximum length allowed by the IPL medium for a single program.

All bare-metal program content must load within the first 16M of memory. Otherwise, a boot 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.

Boot loaders use the LOD1 record to understand the expectations from the installation process. The installation process will ensure that the boot loader is loaded into memory and control is passed to it. However, there is no guarantee that the resident boot loader can perform the operations requested by the LOD1 record. Only the boot loader knows this. SATK supplied boot loaders will validate that the IPL medium and other options are supported by the boot loader.

LOD1 Record Content

LOD1 is the vehicle by which `iplasma.py` communicates information to a boot loader and the mechanism by which a boot loader communicates with a booted program. The `LOD1` Record should be loaded on a four-byte memory address boundary. Primarily this is because it contains address fields. The four-byte boundary is satisfied by the area reserved in real memory for the LOD1 record, X'240'-X'28F'.

All numeric fields are unsigned binary. Fields altered or provided by the boot loader following IPL are identified in green and must be initialized to binary zeros. Reserved fields are highlighted in gray and must also be initialized to binary zeros. Fields without a background color (white) are provided by `iplasma.py` when the LOD1 record is created.

Disp. (Dec)	Disp. (Hex)	Length	Memory Address	Description
+0	+0	4	`000240`	Record identification: EBCDIC `C'LOD1'` or `X'D3D6C4F1'`
+4	+4	1	`000244`	IPL Medium Information: • X'01' – Booted program is contained in an object deck • X'02' – Directed records contain two-byte length of content • X'04' – Card • X'08' – Tape • X'10' – FBA DASD (see field at +24) • X'20' – CKD DASD (see fields +28 - +34) • X'40' – ECKD DASD (to be documented) • X'80' – Eight byte directed record in use, rather than four byte default.
+5	+5	1	`000245`	Reserved, must be zero
+6	+6	2	`000246`	Maximum length of boot directed records 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	4	`000250`	Booted program entry address (from its `IPLPSW` or ASA region) Meaning of bits 0 and 31 of instruction address: • 00 – set addressing mode to 24-bits • 10 – set addressing mode to 31-bits • 01 – set addressing mode to 64 bits (requires changing architecture in dual mode z/Architecture®) • 11 – set addressing mode to 64-bits (z/Architecture only)
+20	+14	1	`000254`	Boot Loader Assembled Architecture: 1. `X'00'` – not supplied by boot loader 2. `X'01'` – S/360 in BC mode 3. `X'02'` – S/370 in BC mode 4. `X'03'` – S/370 in EC mode 5. `X'04'` – S/380

Disp. (Dec)	Disp. (Hex)	Length	Memory Address	Description
				6. X'05' – 370-XA 7. X'06' – ESA/370 8. X'07' – ESA/390 9. X'08' – ESA/390 on dual z/Architecture system 10. X'09' – z/Architecture on dual z/Architecture system 11. X'0A' – z/Architecture only system (not yet supported) 12. X'FF' – Invalid CPU operating environment
+21	+15	1	000255	Boot loader I/O architecture 1. B'0000xxxx' – I/O architecture not supplied 2. B'0001xxxx' – Channel based I/O in use 3. B'0010xxxx' – Channel subsystem in use 4. B'1xxxxxxx' – Invalid I/O architecture Boot loader I/O architecture mode: 1. B'xxxx0000' – I/O architecture mode not supplied 2. B'xxxx0001' – CCW Format 0 in use (24-bit I/O) 3. B'xxxx0010' – CCW Format 1 in use (31-bit I/O) 4. B'xxxx0100' – CCW Format 1 with MIDAW (64-bit I/O) 5. B'x1xxxxxx' – Invalid I/O architecture mode Valid combinations: • X'11' – Channel I/0 with 24-bit addressing • X'22' – Channel subsystem with 31-bit addressing • X'24' – Channel subsystem with 64-bit data addressing
+22	+16	1	000256	Boot loader addressing mode upon booted program entry: 1. X'00' – Addressing mode not supplied by boot loader 2. X'01' – Boot loader in 24-bit addressing mode 3. X'02' – Boot loader in 31-bit addressing mode 4. X'03' – Boot loader in 64-bit addressing mode 5. X'FF' – Invalid addressing mode
+23	+17	1	000257	Boot loader services: 1. B'00000000' – Boot loader services not available 2. B'xxxxxxxx' – Boot loader services available. Meaning defined by the boot loader.
+24	+18	4	000258	FBA DASD booted program starting physical sector
+28	+1C	2	00025C	CKD DASD booted program starting cylinder number
+30	+1E	2	00025E	CKD DASD booted program starting track (head) number
+32	+20	1	000260	CKD DASD booted program starting record number
+33	+21	1	000261	CKD DASD number of boot records per track
+34	+22	2	000262	CKD DASD maximum cylinder number
+36	+24	2	000264	CKD DASD maximum track (head) number
+38	+26	2	000266	Subchannel device number from IPL function
+40	+28	4	000268	IPL device identification from IPL function used by I/O instructions

Disp. (Dec)	Disp. (Hex)	Length	Memory Address	Description
+44	+2C	4	`00026C`	Boot loader input/output area address
+48	+30	4	`000270`	Boot loader services entry address. Valid if 000257 is not zero. All services, when available, are called using this address. Architecture change will alter this address.
+52	+34	1	`000274`	Current running architecture level. See Boot Loader Assembled Architecture for values.
+53	+35	3	`000275`	Reserved, must be zero
+56	+38	4	`000278`	Internal boot loader service return to caller address. Valid if 000257 is not zero.
+60	+3C	4	`00027C`	LOD1 Extension Format Controls provided by boot loader
+64	+44	8	`000280`	Booted program's load address override: • `000280` – `000283` – 24- or 31-bit address • `000280` – `000287` – 64-bit address
+72	+48	8	`000288`	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 booted programs use.

See “Appendix B” for an extended discussion of the motivation for some of these field definitions.

The LOD1 record replaces the capabilities supplied by the imported Python `__boot__.py` module previously used with `iplmed.py`.

Memory Usage and Program Entry

The following table describes the general usage of memory with and without a SATK supplied boot loader. Addresses are absolute. Source refers to the default ASMA region name found in the LDID.

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 does 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 iplasma.py	1, 4	IPL Record 0 (IPL0), IPL PSW and reads IPL Record 1 and transfers CCW control to it.
000000	0001FF	512 (X'200')	ASAREGN.bin	1, 4	IPL Record 3 (IPL3), assigned storage region
000200	00023F	64 (X'40')	none	5	Reserved for Hercules IPL parameter data
000240	EOP	varies	varies	6	Bare-metal program region or image (IPL2)
EOP+	EOP+X	varies	iplasma.py	2, 3, 4	IPL Record 1 (IPL1), reads the bare-metal program
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 ASMA 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. This record normally contains the additional CCW’s required to read 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 boot 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 boot 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` command. 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 replace the IPL PSW from IPL Record 0.

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` (ASMA region name IPLPSW)
- the bare-metal program (IPL Record 2), default region file `PROGRAM.bin` (ASMA region name `PROGRAM`), and last
- the assigned storage area initialization region (IPL Record 3), default region file `ASAREGN.bin` (ASMA region name `ASAREGN`).

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`.

Boot Loader Usage

This section describes how memory is used with a boot loader. The boot 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 boot 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 end

of boot loader input/output area, if used.

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

Start	End	Length	Source	Notes	Description
000000	000017	24 (X'018')	IPLPSW.bin	1, 4, 11	IPL Record 0 (IPL0)
000000	0001FF	512 (X'200')	ASAREGN.bin	1, 4	IPL Record 3 (IPL3)
000000	001FFF	8,192 (X'2000')	none	4	Reserved for boot loader program usage
000200	00023F	64 (X'40')	none	9	Reserved for Hercules IPL parameter data
000240	00028F	448 (X'1C0')	iplasma.py	4, 8, 10	Reserved for IPL Record 4 (LOD1)
000290	0003FF	368 (X'170')	none	13	Reserved for boot loader's use.
000400	0011FF	3,584 (X'E00')	PROGRAM.bin	3, 4, 6	IPL Record 2 (IPL2), the boot loader
001200	001FFF	3,584 (X'E00')	none	14	Reserved for hardware usage
002000	EOP	variable	PROGRAM.bin	2, 3, 7, 12	Booted program loaded by boot loader
EOP+	EOP+X	varies	iplasma.py	4	IPL Record 1 (IPL1)
EOP+X+1	EIO	varies		4, 6	Optional I/O area used by boot 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 boot records on the IPL medium.

Note 3: The sources for the boot loader and booted program may have the same name but always are found in different directories, inferred from the iplasma.py command-line arguments.

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 boot loader. FBA DASD requires an input/output area for support of boot records using the directed format. The Hercules FBA DASD driver does not support data chaining or skipping. 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 boot loader. The boot loader uses the IPL function to become resident and receive control. The same constraints apply to boot loader directed records making some record sizes 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 boot 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 boot 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 boot loader.

Note 12: An image file may also be used as the source of the booted program.

Note 13: The format of this area is identified by the LOD1 Extension Version Control field at LOD1 record offset +60 and is supplied by the boot loader.

Note 14: The hardware usage of this area is for store status save areas and features not used by a boot loader. Use by a boot loader has some “risk” but can occur with some safety. Know what you are doing.

Booted Program Entry

Entry of the booted program is triggered by the boot loader after all directed records have been read and loaded to memory.

Depending upon the LOD1 flags, before entering the booted program:

- the system has been placed in 64-bit architecture mode when option `--arch` has been specified.

All general registers are cleared to zero.

Entry to the booted program is achieved by a branch instruction. The booted program inherits the architecture, addressing mode and trap PSW’s of the boot loader. The booted program can inspect the LOD1 record in storage to determine many of these inherited characteristics.

Upon entry to the booted program, it must establish a base register and perform any self-relocation of address constants as required. Self relocation is required if a booted image file has been loaded at an address other than the one for which it was assembled. List-directed IPL regions are loaded at their assembled starting address.

CARD Deck Structure

Card decks are composed of 80-byte physical records accessed sequentially. Due to the 80 byte constraint, many physical records are required for logical IPL Record 1 when most other media can use just one.

The following table identifies how the different IPL records are managed for card decks.

Logical IPL Record	Description	Card Deck Usage	Physical Records
0	Initial IPL record	required	1
1	IPL channel program	required	multiple
2	Bare-metal program	required	multiple
3	Low-Storage Initialization	optional	multiple
4	Booted Program Info (LOD1)	optional	multiple
multiple	Directed records (booted program)	optional	1 per each directed record

An additional utility is provided for the creation of a deck for the purpose of loading. This utility supports concatenating multiple card image files together into one. The first file is expected to be the card deck loader. See “Appendix C – deck.py”.

IPL Channel Program

The IPL channel program reads all memory content. Both channel commands that read IPL memory content and those that read more channel commands themselves must coexist.

A card deck IPL will mix within the physical records both types of commands. The difference between the two types is the location into which data is read. Commands that read IPL memory content simply will place the content as dictated by the memory usage for the type of data. Commands that read other commands will utilize a pair of buffers for this purpose. IPL Record 0 will read the first set of commands that read data into the first buffer. The last two commands of this record will read the next set of channel commands into the other buffer and then transfers control in the channel to the other buffer. This cycling between buffers continues for the entire channel program.

Each of the channel programs are command chained to the next except for the last CCW in the entire program.

CPU control occurs with the usual loading of the IPL PSW at absolute location 0.

The following table illustrates the physical records constituting IPL Record 1 . IPL Record 0 reads the first card into Buffer A (A below). The first record then reads the second into Buffer B (B below), the CCW at position +64, and transfers channel program control to it, the CCW at position +72.

+0	+8	+16	+24	+32	+40	+48	+56	+64	+72
Read 80	Read 80	Read 80	Read 80	Read 80	Read 80	Read 80	Read 80	Read B	TIC B
Read 80	Read 80	Read 80	Read 80	Read 80	Read 80	Read 80	Read 80	Read A	TIC A
Read 80	Read 80	Read X, SIL	blank	blank	blank	blank	blank	blank	blank

Data records, IPL Record 2, are intermixed following each command record of IPL Record 1. The data records are shown in white. Each 80-byte CCW command card of IPL Record 1 reads a maximum of 640 bytes of memory content. The last IPL Record 1 physical record, because there are no additional physical records to be read, can read 800 bytes of memory content. The last command in red is not command chained. This completes the IPL channel program and causes the IPL PSW to be loaded. Blank areas of the card are shown in gray.

Intermixing the two types of card records results in the following. **Rd** means a Read CCW command. **TIC** refers to a Transfer in Channel CCW command. **A** and **B** refer to the two command containing buffers. Data records are identified by **D**.

Logical IPL Record 0 as one physical card record.

IPL PSW	Rd A	TIC A	blank

Here multiple physical card records show how Logical IPL Record 1 (records highlighted with a buffer color) and Logical IPL Record 2 (white records) are intermixed in the card IPL stream.

+0	+8	+16	+24	+32	+40	+48	+56	+64	+72
Rd D1	Rd D2	Rd D3	Rd D4	Rd D5	Rd D6	Rd D7	Rd D8	Rd B	TIC B
D1									
D2									
D3									
D4									
D5									
D6									
D7									
D8									
Rd D9	Rd D10	Rd D11	Rd D12	Rd D13	Rd D14	Rd D15	Rd D16	Rd A, SIL	TIC A
D9									
D10									
D11									
D12									
D13									
D14									
D15									
D16									
Rd D17	Rd D18	Rd D19, SIL	blank (as last command record, all 10 CCW locations may contain a Rd Dx CCW)						
D17									
D18									
D19								blank	

All of the command containing records can not be read together because the same two areas of memory are reused. However, this strategy essentially provides unlimited IPL program capacity when using an emulated card deck.

Logical IPL Records 3 and 4 can be added to this stream of intermixed CCW commands and IPL data.

Non-Hercules Use

While most SATK loaders are designed for use by other SATK programs on Hercules or simh in the future, the loaders designed for card deck usage have another use case: environments outside of Hercules. In those environments, the SATK card deck loader can function as a drop in replacement for proprietary loaders that fail to meet the needs of a load process for whatever reason.

As in the Hercules use case, such loaders are built using ASMA in particular and SATK generally. But the use of the loader itself can be somewhere else. In this case the loader is expected to be moved, with or without additional files, to the target environment. Movement of most files for use in such environments requires the files to be treated as binary data without line terminations during the transfer.

FBA DASD Structure

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

The following two tables identifies how the FBA sectors are used and by which ASMA output type: image, ld, or object. Object output is only supported with a boot loader.

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

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

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

Sectors	Sector #	image	ld	object	boot	Description
1	0	N	N	N	Y	IPL Records 0 and 1 combined into a single sector
1	1	Y	Y	Y	Y	Reserved for volume descriptor information
1	2	N	N	N	O	IPL Record 3, assigned storage area initialization
1	3	N	N	N	Y	IPL Record 4, the LOD1 record
variable	4+	N	N	N	Y	IPL Record 2, boot loader
variable	n+	Y	Y	Y	Y	Booted program composed of boot loader directed 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 boot loader) and a booted program, if any. Refer to the FBA allocation map for actual sector usage.

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` file path and name as was used with the ASMA `--gldipl` option when the bare-metal program was assembled. When a boot loader is used, multiple regions may be loaded.
- `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 loaded into memory. This option defaults to an address of 0. Provides the location for IPL Record 2.

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.

Boot Loader Input Command-Line Options

The `--boot` option causes the bare-metal program to be booted by the specified boot 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 boot loaders exist for use with these options. Such are planned for future development.

-a/--am MODE

Address mode requested when entering a booted program. MODE may be 24, 31, or 64. Defaults to 24. MODE must be consistent with the boot loader’s run-time capabilities.

-b/--boot FILEPATH

Identifies the list-directed IPL control file of the boot loader. boot loaders are required to be created using the ASMA `--gldipl` option from the `IPLPSW.bin`, `ASALOAD.bin` and `PROGRAM.bin` binary region file names. The list-directed IPL directory must contain only one program region. Multiple regions can not be loaded by the IPL function.

If specified the boot loader will always be used.

A boot 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 2 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 boot program was assembled.

--lasa FILENAME

Specify the boot loader's ASA region if it differs from the default.

--lpsw FILENAME

Specify the boot loader's PSW region if it differs from the default.

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` See note 1.
CKD DASD	`CKD` -> `3330`	`2305`, `2311`, `2314`, `3330`, `3340`, `3350`, `3380`, `3390`, `9345`
FBA DASD	`FBA` -> `3310`	`0671`, `0671-04`, `3310`, `3370`, `3370-2`, `9313`, `9332`, `9332-600`, `9335`, `9336`, `9336-20`
Tape	`TAPE` -> `3420`	`3410`, `3420`, `3422`, `3430`, `3480`, `3490`, `3590`, `8809`, `9347`

Note 1: The card medium uses two separate devices: a card reading device, `RDR`, and a punching device, `PUN`. Because `iplasma.py` is "punching" an output deck, it uses a device type that "punches", a 3525.

Currently only FBA DASD device types are fully supported. Cards are supported for a bare-metal program but without a booted program.

-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.

-o/--owner NAME

Adds the NAME as the volume's owner when a volume label is created for the volume (see `--volser`). Spaces are not allowed in the NAME. Defaults to SATK. Ignored when a volume label is not created.

-r/--recl SIZE

Describes the length of the directed boot records in bytes (including the four-byte or six-byte header). Must be compatible with the record sizes supported by the device and selected boot loader. If omitted, a default record size for the supported device will be used. For card unit record devices, the default is 80. For all other devices the default is 512.

-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`.

-v/--volser ID

When specified a volume label is created for the volume with the identified volume serial identification. Ignored if the volume is not a DASD volume. The volume label contains no information about a Volume Table of Contents. It only identifies the volume.

Appendix A - 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 just 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 boot loader records. In essence, the tooling for sequentially accessed volumes is different than the tooling required to create directly accessed volumes.

Reusable Boot Loaders

As originally conceived, the IPL ELF ABI Supplement expected a boot 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 boot loader.

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

Media Preparation Data

The *IPL ELF ABI Supplement* allowed 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 boot loader.

The use of IPL media by the ASMA process standardizes the use of preparation data by the boot loader. A new standard record, `LOD1`, is used to contain boot loader specific information. The `LOD1` record is considered part of the set of logical 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 boot loader and the nature of the IPL medium. If the `LOD1` record only contains information related to the program brought into memory by the boot loader, then the boot loader may read it. If, however, the `LOD1` record contains information related to the medium required by the boot loader when accessing the medium, then the IPL function must load it for the boot loader. Without this information, then potentially the boot 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 boot loader or program must actually inform the DASD of the extent before it can read from the device. The values supplied in the `LOD1` 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 boot 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 boot 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.

Boot Loader Interface

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

This assembled interface proved to be overkill in practice after the ability to reuse a boot loader became an option.

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

Stream vs. Directed Boot Program Records

Two types of sequentially accessed boot loader records were supported by `iplmed.py` depending upon the boot 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 record's address as the starting point for the record's 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 ASMA created boot loaders standardize on the directed IPL record format.

Some boot 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 boot loaders supporting card decks or emulated tape volumes.

Appendix B – How to Manage PSW's

IPL of a List-Directed IPL Directory

The list directed IPL is the most complex case. Multiple sources of the IPL PSW can exist when multiple assembled regions all start at address 0. The `--psw` and `--asa` command-line options assist, but their purpose is to identify these regions when they do not follow the default naming convention. The region names supplied by the `--psw` and `--asa` may not actually exist in the directory. The names differentiate the regions with special roles vs. those that are considered just part of the bare-metal program.

IPL Record 2 contains the IPL'd program or boot loader. Unlike legacy IPL processes, multiple IPL Record 2's may be brought into memory by the `iplasma.py` generated IPL sequence of CCW's. There are potentially three regions that could all be assembled at address 0. Each region must reside in a single IPL record. The medium may limit this size. Hence it is possible that some IPL'd programs may work fine on one medium and not others.

Boot Loaders must be contained in a LDID. Other bare-metal programs that are loaded by IPL in LDID's, must follow these same rules. It is the same process that brings both types of bare-metal programs into memory. The primary difference is the role of the bare-metal program. A boot loader resides in low memory (at addresses X'400' to X'1200', or 3,072 bytes in length).

What differs between the two types of bare-metal programs is how they are identified to `iplasma.py`. The generic program (not a boot loader) is identified using the `source` positional argument, the last argument in the command-line. Whereas, a boot loader's directory is identified by the `--boot` command-line option. In either case, the `--psw` and `--asa` command-line options apply to the program loaded by the IPL function.

The assumption is made that the initial PSW **or** the initialized assign storage area are assembled respectively into its own region. Operative word is "or". One or the other, but not both. The assembled program will use an assembler PSW directive when assembling just the IPL PSW. When assembling an initialized assign storage area, the program will use the `ASALOAD` and `ASAIPL` sequence of macros. The ASAIPL macro supplies the IPL PSW for the assigned storage area (which starts at address 0).

The rules for a list-directed load directory:

- One region containing the IPL PSW starting at assembled address X'0': either a PSW region (used in IPL Record 0) or an assigned storage area region (in IPL Record 3).
- If an assigned storage area region is not supplied, a PSW region is required, and
- One or more regions (in one or more IPL Record 2's respectively) containing the IPL'd program or boot loader starting with an address at or beyond address X'200'. Note: SATK supplied boot loaders will start at address X'400'.

When the bare-metal program is a boot loader, IPL Record 4 (the LOD1 record) will also be created by `iplasma.py` and loaded at address X'240'.

IPL of an Image File

An image file is treated as two regions. The first eight bytes of the image are assumed to be the IPL PSW, the logical equivalent to a separate region. The image file itself, including its initial eight bytes, will be loaded at the address specified by the `--load` command-line option and is treated as a single region. When dealing with an image file only one region should be assembled by ASMA.

Multiple regions can be assembled with an image file, but the additional regions must be relocated by the program itself to their assembled locations after the image file resides in memory. From the IPL of an image file, such relocation is not a factor. The entire image file must be of a size that it can be loaded by IPL Record 2 when placed on the IPL medium.

The image file can be loaded at address 0 allowing the IPL PSW (this time from IPL Record 2) and the assigned storage area to be initialized. There is no practical difference then between use of multiple regions (in a list-directed IPL directory) or a single image file in terms of what is possible when IPL'd. Except of course, the image file can not contain a boot loader and areas reserved for the system must be present within the image. This latter exception increases the size of the required IPL Record 2 and is avoided by use of a list-directed IPL directory.

Boot Loaded List-Directed IPL Directory

The only PSW that matters is that of the one used to enter the booted program. This PSW, identified by being loaded at address X'0' in the control file is placed into LOD1 for use in entering the booted program by the boot loader. If the region is longer than 8 bytes, only the initial 8 are used for the entry PSW. Only one such region is allowed. All other regions, of which there can be multiple, are loaded by the boot loader using directed boot records. No size limit exists.

The only requirement of the addresses is that they do not overlap with:

- the boot loader,
- the **preceding** memory areas reserved for other uses, or
- any of the I/O area used by the loader.

Whether trap PSW's are initialized before entry is dictated by the LOD1 record (`--traps`).

These three mechanisms logically replace IPL Record 0 (the IPL PSW), IPL Record 2 (program loading) and IPL Record 3 (assigned storage initialization).

Boot Loaded Image File

As with the bare-metal loading, the first 8 bytes of the image file are treated as the file's entry

PSW. This PSW is placed within the LOD1 record. The entire image file is loaded at its specified load address (`--load`) which must not overlay either the boot loader and its preceding addresses or the I/O area used by the loader.

Booted Program Environment Management

What is the role of the boot loader? Obviously it loads the “real” bare-metal program. But that is useless if control is not passed to the booted program. Which, in essence, means passing an environment to the booted program.

There are two ways to pass control:

- use a branch type instruction, or
- load a new PSW.

Either effects the instruction address of the current PSW. In the first case, branching, the instruction address is changed based upon the instruction’s operand. In the second case, the instruction address is changed by the fact that an entire new PSW is introduced as the current PSW.

The LOD1 record is established with the concept that the booted program is entered by means of a branch type instruction.

This is an arms-length approach to boot loading. The boot loader knows nothing about the booted program’s environment and, vice versa, the booted program knows nothing about the boot loader’s environment. The only link between the two is the transfer of control from the boot loader to the booted program.

This arms-length connection between the boot loader and booted program represents reality. Without a mechanism that allows the boot loader to communicate information about its execution environment, the booted program can not know what to expect and what it might need to do to change that environment to its liking.

The LOD1 record provides the vehicle for such communication. It has a standardized structure and a fixed location. It is created by `iplasma.py` which understands some aspects of the IPL function. After loading, the boot loader knows its functionality and can communicate it via LOD1 to the booted program. In turn the booted program can recognize whether it can operate within the boot loader’s environment or needs to change it.

By its nature as a program, the boot loader has an environment in which it operates. The `--ccw1` option changes the boot **loader**’s environment to using 31-bit addressing. The `--zarch` option explicitly is intended to adjust the **booted** programs environment, but will change the boot loader’s environment as well.

One of the longer term goals for the boot loader is to provide some simple services to the booted program. In this role the two environments are inextricably linked and the booted program must operate in a compatible environment. No sophisticated interrupt driven connection is assumed here. Just simple sub-routines provided by the loader that the booted program might use.

Appendix C – deck.py

The `deck.py` utility found in the tools directory of SATK is used to concatenate multiple emulated card deck files into a single file suitable for IPL. This means that the first file must be a bare-metal program, the boot loader, followed by the records read by the boot loader, the booted program, in whatever format is expected by the boot loader.

When the boot loader is supplied by SATK and the booted program is placed in directed records, `iplasma.py`, produces the appropriate card deck. `deck.py` is used where the booted program has some other content, for example, an absolute object module of `TXT` records. ASMA produces such a module as output.

Two output formats are provided by `deck.py`:

- Emulated card deck or
- Virtual AWS tape file.

Both formats use the same channel programs in the same ways. Where an AWS virtual tape file is required, `deck.py` may be used to convert one or more emulated card decks into a single AWS virtual tape file.

Some operating systems may provide utilities that can be used for a similar purpose, namely, concatenating multiple binary files together. `deck.py` has the added benefit that only files containing emulated card decks are accepted for concatenation: a file of exactly 80-byte cards without line terminations.

Additionally the file content can be analyzed for its content of ASCII, EBCDIC, and other characters. ASCII and EBCDIC character sets overlap so these values may add up to values larger than the total number of bytes.

The general command-line format for the tool is:

```
deck.py  [ options ... ] source ...
```

The `source` positional arguments name each of the input emulated card deck files. This means that each file must be an integral of 80-bytes. Only full cards are accepted. The environment variable `DECKS`, when present, is used to locate each file identified by a relative file path or just by the file's name. Otherwise `source` must contain a fully qualified path.

If any file is rejected because it has an incomplete 80-byte record, the output file is suppressed.

DECKS Environment Variable

The optional environment variable `DECKS` identifies the sequence of directories searched for when locating input files.

Command Line Options

--boot FILENAME

The `--boot` option allows a boot loader to be identified. The boot loader must be an emulated card deck. The boot loader can be created using `iplasma.py` as a single bare-metal program. A boot loader is always placed as the first of the concatenated input files onto the output medium. The `DECKS` environment variable, if present, is applied to the `FILENAME`. Otherwise `FILENAME` must contain a fully qualified path.

If omitted, the sequence of `source` positional arguments dictates the sequence. When present, the file identified by `--boot` is always first, followed by the files in the sequence of the `source` positional arguments.

Unless the intent is to include the boot loader more than once, it should not be included in the list of `source` arguments.

-c/--card FILEPATH

Identifies the created output emulated card deck file. `FILEPATH` must be a fully qualified file path. If neither `--tape` nor `--card` are specified, no output file is created. When used, `--tape` may not be present.

--dump

Produces a hexadecimal dump of input files.

-t/--tape FILEPATH

Identifies the created output virtual AWS tape file. `FILEPATH` must be a fully qualified file path. If neither `--tape` nor `--card` are specified, no output file is created. When used, `--card` may not be present.

--tm NUMBER

Specifies the number of tape marks to append to the created tape when `--tape` is used. Defaults to 0. Ignored if --`tape` is not used.