

LibMCPU MELF ABI

Формат объектов и исполняемых файлов MСPU, контракт компоновщика (linker) и загрузчика (loader)

Самостоятельное справочное руководство; нумерация разделов начинается с 1

Нормативное техническое руководство LibMCPU. Документ описывает текущий архитектурный контракт и намеренно не привязан к номеру выпуска пакета.

Статус документа. Документ является самостоятельным описанием текущего MELF-контракта LibMCPU и намеренно не связан с номером выпуска пакета. Он фиксирует статический ABI v1, исторический explicit-preload ABI v2, frozen normal dynamic ABI v3 с SONAME/NEEDED, lifecycle, STACK/HEAP, MCPU static archives и partial ET_REL linking. HLL PROGRAM/MODEL classification и automatic CRT insertion описаны как linker layer над MELF container ABI.

MELF — не системный ELF. Формат намеренно использует сигнатуру `_ELF`, частное значение `e_machine=MCPU` и специфические для MCPU секции и заголовков сегментов (Program Headers). Ядро host-системы и системный `ld.so` не участвуют в запуске. `mcpu-ld` и `mcpu-run` являются частями одного целевого ABI.

Оглавление

Оглавление	1
1 Назначение и границы формата	3
2 Идентификация, класс и <code>e_machine</code>	3
2.1 Поле <code>e_ident</code>	3
2.2 <code>e_machine</code> и <code>e_flags</code>	3
3 Типы файлов и версии ABI	3
4 Заголовок MELF64	3
5 Архитектурные адреса внутри MELF	4
6 Секции и флаги секций	4
7 Объектная модель ET_REL	5
8 Таблицы символов ABI	5
8.1 <code>st_value</code> в ET_REL	5
8.2 Статическое разрешение символов	5
8.3 Lifecycle-символы	5
9 RELA ABI	5
10 Частичная компоновка <code>mcpu-ld -r</code>	6
11 Статические библиотеки <code>lib*.a</code>	6
12 Метаданные профиля целевой архитектуры	6
13 Метаданные lifecycle	7
14 Таблица заголовков сегментов (Program Headers)	7
15 Статический ET_EXEC ABI v1	7
16 Точки входа lifecycle исполняемого файла	8
16.1 С секцией <code>.init</code>	8
16.2 Без секции <code>.init</code>	8
16.3 FINI	8
16.4 Связь MELF lifecycle с HLL Runtime ABI	8
17 Постоянная область запуска <code>McpuStartup</code>	8
18 Исторический MELF ABI v2	8
19 Frozen MELF ABI v3	9
20 SONAME, NEEDED и link-time <code>-L/-l</code>	9
21 Поиск библиотек и runtime root	9
22 Рекурсивная загрузка NEEDED	9
23 <code>.dynsym/.dynstr</code> и runtime RELA	9
24 Детерминированный runtime symbol scope	10
25 Контракт ET_DYN	10
26 LOCAL relocation внутри ET_DYN	10
27 Размещение образа процесса и единственный DATA guard	10
28 BSS и соотношение <code>filesz/memsz</code>	10
29 Дескриптор STACK и командная строка	11
30 Дескриптор HEAP, <code>_end</code> и <code>SYS_brk</code>	11
31 Дисциплина проверки загрузчика	11

32	Режимы команды mcpu-ld	11
33	Инструменты анализа и безопасного редактирования MELF	12
33.1	Read-only inspection	12
33.2	mcpu-strip	12
33.3	mcpu-objcopy	12
33.4	Non-ALLOC debug metadata и frozen ABI v3	12
34	Правила совместимости	12
1	Размеры сериализованных записей	13
2	Канонические имена секций	13
3	Нормативная памятка frozen dynamic ABI v3	13

1. Назначение и границы формата

MELF хранит три вида файлов MCPU:

- перемещаемый объект (**relocatable object** (*перемещаемый объект*)) ET_REL;
- исполняемый образ (**executable** (*исполняемый файл*)) ET_EXEC;
- разделяемый объект (**shared object** (*разделяемый объект*)) ET_DYN, явно загружаемый при запуске программы.

От ELF заимствованы общие идеи заголовков Ehdr, Shdr и Phdr, таблиц символов и строк, а также relocation-записей RELA. Архитектурные пространства памяти, точки входа lifecycle и метаданные профиля целевой архитектуры определяются самим MCPU ABI.

Текущая реализация использует только MELF64. Имена и структуры MELF32 зарезервированы, но ABI v1, ABI v2 и ABI v3 их не принимают. С-структуры из `mcpu-elf.h` описывают логические поля, но не являются готовой сериализацией на диске (on-disk serialization): код записи и чтения явно кодирует и декодирует многобайтовые поля с учётом EI_DATA.

PROGRAM образует отдельную область с собственным порядком байтов: машинные поля MCPU внутри PROGRAM всегда записываются MSB-first независимо от EI_DATA контейнера.

2. Идентификация, класс и e_machine

2.1. Поле e_ident

Поле	Значение	Смысл
MAG0..3	'_', 'E', 'L', 'F'	сигнатура MELF; это не обычный 0x7fELF
EI_CLASS	2	MELF_CLASS_64
EI_DATA	1 или 2	порядок байтов LSB/MSB для метаданных и объектов данных целевой архитектуры
EI_VERSION	1	текущая версия формата
EI_OSABI	64	MELF_OSABI_MCPU
EI_ABIVERSION	1, 2 или 3	статический ABI v1 / исторический dynamic ABI v2 / frozen dynamic ABI v3

2.2. e_machine и e_flags

`e_machine=0x4d43` (ASCII "MC"). Это частное значение MELF, а не официальное значение системного ELF. В текущем ABI `e_flags=0`. Неизвестные биты загрузчик (loader) обязан отвергать, а не молча игнорировать.

3. Типы файлов и версии ABI

В MELF номер ABI относится не к версии инструмента, а к конкретному контракту файла. Текущий toolchain продолжает читать исторические ABI v1/v2 и выпускает ABI v1 для ET_REL/статического ET_EXEC и frozen ABI v3 для новых dynamic ET_EXEC/ET_DYN.

Режим формирования	e_type	ABI	Назначение
<code>mcpu-as</code>	ET_REL	1	перемещаемый входной объект
<code>mcpu-ld-r</code>	ET_REL	1	частично скомпонованный перемещаемый объект
<code>mcpu-ld</code>	ET_EXEC	1	полностью статический исполняемый файл
исторический dynamic output	ET_EXEC / ET_DYN	2	explicit-preload startup ABI, сохраняемый загрузчиком для совместимости
<code>mcpu-ld -dynamic</code>	ET_EXEC	3	главный startup-dynamic исполняемый файл с <code>.dynamic</code>
<code>mcpu-ld -shared</code>	ET_DYN	3	shared object с обязательным SONAME и возможными NEEDED
ET_CORE	зарезервировано	—	пока не поддерживается

ABI v3 является frozen normal dynamic ABI. Его несовместимое изменение требует нового номера ABI. ABI v3 не меняет CPU ISA, CALL ABI, PTR64 или базовые типы RELA; он расширяет startup dynamic-link contract метаданными `.dynamic`, SONAME/NEEDED, поиском библиотек и детерминированным runtime symbol scope.

4. Заголовок MELF64

Сериализованный MELF64_Ehdr имеет размер 88 байт. Поля точек входа lifecycle содержат полные архитектурные PTR64 (PROGRAM, offset), а не смещения внутри файла.

Поле	Размер	Смысл
e_ident	16	сигнатура, класс, порядок данных, версия, OSABI, версия ABI
e_type	2	ET_REL / ET_EXEC / ET_DYN
e_machine	2	MCPU
e_version	4	MELF_EV_CURRENT
e_entry	8	обязательная точка входа исполняемого файла; всегда _start
e_init_entry	8	_start, если существует .init; иначе 0
e_life_entry	8	обязательный _life
e_fini_entry	8	необязательный _fini; иначе 0
e_phoff	8	смещение в файле до таблицы заголовков сегментов (Program Headers)
e_shoff	8	смещение в файле до таблицы заголовков секций (Section Headers)
e_flags	4	в текущем ABI равен 0
e_ehsize	2	88
e_phentsize	2	64 для MELF64
e_phnum	2	число записей заголовков сегментов
e_shentsize	2	64 для MELF64
e_shnum	2	число записей заголовков секций
e_shstrndx	2	индекс секции .shstrtab

У ET_REL окончательных адресов lifecycle ещё нет, поэтому четыре поля точек входа равны нулю. ET_DYN не является самостоятельной программой и также не имеет исполняемых точек входа lifecycle.

5. Архитектурные адреса внутри MELF

Для окончательно размещённых загружаемых секций (allocated sections) поле sh_addr содержит полный PTR64 (segment, final_offset). Используется следующая таблица селекторов (selectors):

Пространство	SEG	Секции / роль
DATA	0x0	.data, .bss
STACK	0x1	только дескриптор времени выполнения; полезные данные объекта отсутствуют
PROGRAM	0x2	.init, .text, .fini
CONTROL	0x3	.control
INPUT	0x4	.input
OUTPUT	0x5	.output
TRACE	0x6	.trace
HEAP	0x7	только дескриптор времени выполнения; данных ассемблерного объекта нет

В ET_REL символы и секции используют смещения относительно своей секции и ещё не имеют окончательной базы процесса. В ET_DYN адреса также остаются относительными к модулю; окончательные базы PROGRAM и DATA назначает компоновщик запуска (startup linker).

6. Секции и флаги секций

MELF использует типы секций, близкие к ELF: SHT_NULL, SHT_PROGBITS, SHT_SYMTAB, SHT_STRTAB, SHT_RELA, SHT_NOBITS, SHT_DYNSYM и SHT_DYNAMIC.

Кроме того, определены два специфических для MCPU типа:

Тип секции	Числовое значение
SHT_MCPU_TARGET	0x70000000
SHT_MCPU_LIFECYCLE	0x70000001

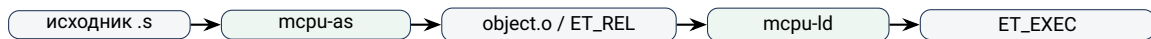
Загружаемые секции получают обычные признаки R/W/X через SHF_WRITE/ALLOC/EXECINSTR; тег архитектурного сегмента MCPU кодируется в процессорных битах флагов.

Секция	Тип	Пространство	Атрибуты
.init	PROGBITS	PROGRAM	ALLOC + EXEC
.text	PROGBITS	PROGRAM	ALLOC + EXEC
.fini	PROGBITS	PROGRAM	ALLOC + EXEC
.data	PROGBITS	DATA	ALLOC + WRITE
.bss	NOBITS	DATA	ALLOC + WRITE
.control	PROGBITS	CONTROL	ALLOC, только чтение для guest-программы
.input	PROGBITS	INPUT	ALLOC, только чтение для guest-программы
.output	PROGBITS	OUTPUT	ALLOC + WRITE
.trace	PROGBITS	TRACE	ALLOC + WRITE

`sh_addralign` задаёт требование выравнивания `guest`-адреса. Значения 0/1 означают отсутствие специального требования, остальные допустимые значения являются степенями двойки. `SHT_NOBITS` занимает архитектурную память, но не содержит данных в файле.

7. Объектная модель ET_REL

Ассемблер формирует символы со смещениями относительно секций и явные записи RELA. Типовой путь выглядит так:



В каждом `ET_REL`, включая результат `mcpu-ld -r`, имеется ровно одна `.mcpu.target`. Окончательная `.mcpu.lifecycle` ещё отсутствует: формирование `lifecycle` принадлежит компоновщику (linker), а в перемещаемом объекте представлено каноническими символами `_start`, `_life` и необязательным `_fini`.

8. Таблицы символов ABI

Запись MELF64 в таблице символов имеет размер 24 байта. Символ 0 — полностью нулевая запись неопределённого символа. В `st_info` привязка (binding) хранится в старшей тетраде, тип — в младшей.

Значения binding: LOCAL=0, GLOBAL=1, WEAK=2. Типы символов: NOTYPE, OBJECT, FUNC, SECTION, FILE. Для обычных символов времени запуска используется visibility DEFAULT; в подстановке символов при запуске (interposition) участвуют только имена DEFAULT GLOBAL/WEAK.

8.1. st_value в ET_REL

Для определённого символа `st_shndx` содержит индекс секции, а `st_value` — байтовое смещение от её начала. Для неопределённого символа используется `SHN_UNDEF` и нулевое значение. Директива `.equ` остаётся механизмом абсолютных выражений времени ассемблирования и не обязана создавать запись MELF symbol table.

8.2. Статическое разрешение символов

LOCAL никогда не разрешается через границу object file. Сильный GLOBAL имеет приоритет над WEAK. Несколько сильных GLOBAL-определений являются ошибкой. Если сильное определение отсутствует, несколько WEAK-определений выбираются детерминированно по порядку входных файлов компоновщика. Неопределённый сильный GLOBAL после окончательной компоновки является ошибкой. При `mcpu-ld -r` такой символ может оставаться неопределённым и переносится вместе с RELA в новый `ET_REL`.

8.3. Lifecycle-символы

`_start` и `_life` обязательны как GLOBAL/FUNC для каждого исполняемого файла; `_fini` — необязательный GLOBAL/FUNC. Lifecycle-символы не могут быть WEAK. Псевдометка исходного текста `_start_life` существует только в синтаксисе ассемблера и эквивалентна двум символам `_start` и `_life` с одним смещением в секции `.text`, когда `.init` отсутствует.

9. RELA ABI

Запись MELF64 RELA имеет размер 24 байта: `r_offset`, `r_info` и знаковый 64-битный `r_addend`. Базовое выражение relocation: `S+A`. Все текущие MCPU relocation исправляют ровно 8 байт.

Relocation	Где находится поле	Цель	Записываемое значение
R_MCPU_CODE64	PROGRAM	PROGRAM	окончательное необработанное байтовое смещение PROGRAM
R_MCPU_DATA64	PROGRAM	DATA/.bss	окончательное необработанное байтовое смещение DATA
R_MCPU_PTR64_CODE	PROGRAM	загружаемый сегмент	PTR64(segment,offset) в порядке байтов машинного кода
R_MCPU_PTR64_DATA	секция данных	загружаемый сегмент	PTR64 в представлении данных целевой архитектуры
R_MCPU_CPTR64_DATA	секция данных	только PROGRAM	PTR64(PROGRAM,offset)

В ET_REL поле `r_offset` является байтовым смещением внутри целевой секции, указанной через `sh_info`; `sh_link` указывает на таблицу символов. Все 8 байт исправляемого поля обязаны помещаться в сохранённой секции. Relocation непосредственно в область NOBITS запрещена: в файле нет байтов, которые можно исправить.

Relocation неопределённого WEAK-указателя с нулевым `addend` может дать канонический полностью нулевой NULL. Для необработанных CODE64/DATA64 такое молчаливое преобразование в zero запрещено, потому что `offset 0` является допустимым архитектурным адресом.

10. Частичная компоновка `mcru-ld -r`

Режим `mcru-ld -r` объединяет несколько MELF ET_REL в один новый ET_REL. Это именно **partial link** (частичная компоновка), а не построение исполняемого образа.

Для каждой одноимённой секции вклады входных объектов конкатенируются в порядке входа с соблюдением их `sh_addralign`. Значения определённых символов и `r_offset` пересчитываются в координаты объединённой секции. LOCAL-символы разных объектов остаются раздельными даже при одинаковом имени. Для GLOBAL/WEAK действует обычное правило силы: сильный GLOBAL заменяет WEAK; два сильных GLOBAL одного имени являются ошибкой.

RELA при `-r` не превращаются в окончательные архитектурные адреса. Они переносятся в выходной объект с новым `section-relative offset` и ссылкой на канонический символ результирующего ET_REL. Неопределённые GLOBAL/WEAK допускаются и могут быть разрешены последующей окончательной компоновкой.

```
mcru-ld -r -o combined.o a.o b.o c.o
mcru-ld --stack-size=64K -o program main.o combined.o
```

STACK/HEAP, lifecycle-финализация, Program Headers, SONAME/NEEDED и runtime image в режиме `-r` не создаются. Поэтому `-r` взаимоисключён с `--dynamic`/`--shared`; опции размеров STACK/HEAP и `--soname` для него недопустимы.

11. Статические библиотеки `lib*.a`

Статическая библиотека MCPU не является новым типом MELF-файла. Это обычный UNIX `a`-контейнер, чьи `ordinary members` являются независимыми текущими MELF ET_REL. Утилита `mcru-ar` создаёт `deterministic archives` и при модификаторе `s` записывает настоящий System-V/GNU-style symbol index.

```
mcru-ar rcs libmodel.a model.o solver.o helpers.o
mcru-ar t libmodel.a
```

При линковке из архива извлекается только `member`, определяющий текущий неразрешённый сильный GLOBAL. Неопределённый WEAK сам по себе `member` не вытягивает. После каждого извлечения текущий архив повторно сканируется до устойчивого состояния, поэтому новый `member` может породить ссылку на `member`, расположенный раньше в том же архиве. Уже пройденный отдельный `-l archive` автоматически после следующей библиотеки не пересматривается; повторное `-la` имеет реальный смысл.

Для ABI-v1 и для `-r -lfoo` ищет только `libfoo.a`. Для ABI-v3 `--dynamic`/`--shared` поиск является `directory-major`: в каждом каталоге сначала проверяется `libfoo.so`, затем `libfoo.a`. Статически извлечённые `members` становятся частью результирующего MELF и не создают DT_NEEDED.

12. Метаданные профиля целевой архитектуры

`.mcru.target` обязательна в ET_REL и ET_EXEC; модули ABI v2/v3 несут тот же профиль совместимости. Запись имеет размер 64 байта, не является ALLOC и выровнена на 8.

Поле	Смещение	Смысл
mt_version	0	версия структуры профиля, сейчас 1
mt_size	4	размер записи, сейчас 64
mt_isa_revision	8	ревизия ISA, сейчас 1
mt_abi_revision	12	ревизия программного ABI, сейчас 1
mt_program_model	16	профиль ILP32/LP64/LLP64
mt_data_byte_order	20	порядок байтов объектов данных; согласован с EI_DATA
mt_data_word_order	24	порядок слов LibMPPU
mt_integer_model	28	дополнительный код (дополнительный код)
mt_address_model	32	PTR64 SEG4:OFF60
mt_register_model	36	REG24 / регистровый пул 512 KiB
mt_real_model	40	LibMPPU
mt_complex_model	44	LibMPPU
mt_max_align	48	максимальное обычное выравнивание ABI = 16
mt_flags	52	пока 0
mt_required_features	56	пока 0

Компоновщик требует совместимости профилей целевой архитектуры. Загрузчик повторно проверяет окончательный профиль до создания guest-областей памяти. Среда выполнения не исправляет несовместимый профиль и не конвертирует порядок байтов данных автоматически.

13. Метаданные lifecycle

Окончательный ET_EXEC содержит `.mcpu.lifecycle` фиксированного размера 32 байта. Запись описывает **семантику жизненного цикла**, но не дублирует адреса точек входа: единственным нормативным источником адресов остаётся Ehdr.

Флаги ABI v1 фиксируют: START обязателен, LIFE обязателен, INIT и FINI необязательны; `e_entry` указывает на START; без INIT START совпадает с LIFE; LIFE завершается END и может быть продолжен; штатный EXIT переходит в FINI; область запуска (startup area) остаётся действительной всё время жизни программы.

Метаданные lifecycle намеренно не содержат копий указателей на `_start/_life/_fini`. Благодаря этому в образе нет двух независимо перемещаемых источников истины. Загрузчик проверяет согласованность Ehdr и записи lifecycle, после чего использует числовые поля точек входа из Ehdr.

14. Таблица заголовков сегментов (Program Headers)

Запись MELF64 Phdr имеет размер 64 байта и расширяет привычную ELF-подобную форму полем `p_maxsz`. `p_paddr=0`: отдельного пространства физических адресов у текущего MCPU ABI нет. `p_align` относится к выравниванию guest-адреса и файла, а не к размеру страницы host-системы.

Пространство	p_type	Флаги	Правило размеров
PROGRAM	PT_LOAD	R X	filesz=memsz=maxsz
DATA	PT_LOAD	R W	filesz <= memsz; maxsz=memsz
CONTROL	PT_LOAD	R	сохранённый и runtime-размеры равны
INPUT	PT_LOAD	R	сохранённый и runtime-размеры равны
OUTPUT	PT_LOAD	R W	сохранённый и runtime-размеры равны
TRACE	PT_LOAD	R W	сохранённый и runtime-размеры равны
STACK	PT_MCPU_STACK	R W	filesz=0; memsz=maxsz
HEAP	PT_MCPU_HEAP	R W	filesz=0; memsz <= maxsz

Для пространств, загружаемых из образа, `p_vaddr=PTR64(segment, 0)` в координатах окончательного образа исполняемого файла или модуля. Дескриптор STACK обязателен в ET_EXEC; дескриптор HEAP необязателен. Округление host-мmap к размеру страницы не является частью ABI.

15. Статический ET_EXEC ABI v1

Обычный `mcpu -ld` завершает всю работу времени компоновки до запуска:

1. проверяет профиль целевой архитектуры каждого ET_REL;
2. размещает вклады секций с учётом `sh_addralign`;

3. создаёт единственную 16-байтовую защитную нулевую область DATA (NULL guard) по смещениям 0..15;
4. разрешает символы GLOBAL/WEAK/UNDEF;
5. применяет все RELA с проверкой диапазона и сегмента;
6. проверяет обязательные определения lifecycle;
7. назначает окончательные PTR64-адреса секций и символов;
8. формирует окончательные заголовки сегментов (Program Headers) и метаданные.

Исполняемый файл ABI v1 не нуждается в поиске символов при запуске. Любая попытка использовать `--ldpreload` или `MCPU_LD_PRELOAD` с ABI v1 является ошибкой, а не поводом неявно превратить программу в dynamic ABI.

16. Точки входа lifecycle исполняемого файла

16.1. С секцией .init

`_start` принадлежит `.init`; поля `e_entry` и `e_init_entry` указывают на `_start`. `_life` принадлежит `.text`, а `e_life_entry` указывает на `_life`. После END в INIT супервизор начинает повторяющиеся фазы LIFE.

16.2. Без секции .init

`_start` и `_life` обязаны находиться по одному адресу в `.text`. Псевдометка `_start_life`: создаёт именно эту пару символов. В окончательном исполняемом файле `e_init_entry=0` и `e_entry==e_life_entry`.

16.3. FINI

Если существует `.fini/_fini`, поле `e_fini_entry` указывает на него; иначе оно равно 0. Штатный EXIT после LIFE входит в FINI. Диагностическая остановка по внешнему лимиту шагов не имитирует EXIT и не обязана запускать FINI.

16.4. Связь MELF lifecycle с HLL Runtime ABI

Сам MELF container определяет машинные entry points `_start`, `_life` и optional `_fini`. Поверх этого container contract существующий `mcpu - ld` реализует HLL complete-program layer: strong GLOBAL/FUNC `main+fini` в одном defining ET_REL классифицирует PROGRAM, а `init+life+fini` — MODEL. Linker автоматически добавляет соответствующий CRT из `<MCPU-root>/lib/mcpu/`; отсутствие обязательного CRT является fatal error. Raw low-level objects с собственными `_start/_life[_fini]` продолжают линковаться без automatic CRT, но смешивать оба режима запрещено.

Это не меняет wire format MELF: HLL names являются linker policy над обычной symbol table, а окончательный ET_EXEC по-прежнему исполняется через нормативные lifecycle entry points. Loader не классифицирует HLL source form.

Полный контракт HLL + CRT + MCPU Libc, включая `McpuStartup`, PRE_RUNTIME/RUNNING, internal Libc hooks и MODEL return namespace, приведён в `libmcpu-bytecode-architecture - ru.pdf`.

17. Постоянная область запуска McpuStartup

До первой guest-инструкции загрузчик строит в верхней части STACK структуру `McpuStartup`. Она имеет поле версии, поэтому имя C-типа намеренно не содержит суффикс V1.

Смещение	Поле	Размер	Смысл
0x00	version	4	MCPU_STARTUP_VERSION_1
0x04	size	4	32
0x08	argc	8	число аргументов guest-программы
0x10	argv	8	PTR64(STACK, массив argv)
0x18	зарезервировано	8	0

После заголовка располагается массив `argv[argc+1]` из PTR64, завершающий NULL, затем строки аргументов, завершённые NUL, и нулевое дополнение. `startup_base` выровнен на 16 байт. Начальное состояние: `PC=e_entry`, `SP=startup_base`, `FP=stack_size`. Искусственного CALL-frame для `_start` нет.

18. Исторический MELF ABI v2

ABI v2 ввёл ET_DYN, `.dynsym/ .dynstr` и сохранённые runtime RELA, но не имел SONAME/NEEDED. Полный набор shared objects задавался явным `mcpu - run - -ldpreload=a.so:b.so`; вся startup-компоновка завершалась до первой инструкции `_start`. ABI v2 остаётся читаемым для совместимости, тогда как текущие `mcpu - ld - -dynamic - -shared` выпускают ABI v3.

19. Frozen MELF ABI v3

ABI v3 сохраняет image/symbol/RELA-модель ABI v2 и добавляет нормальную модель зависимостей. Каждый dynamic ABI-v3 ET_EXEC/ET_DYN содержит ровно одну не-ALLOC секцию .dynamic типа SHT_DYNAMIC, связанную через sh_link с .dynstr.

Запись MELF64_Dyn имеет размер 16 байт: знаковый 64-битный d_tag и беззнаковый 64-битный d_val. Текущие теги:

Тег	Значение	Смысл
DT_NULL	0	обязательный единственный терминатор canonical stream
DT_NEEDED	1	offset имени требуемого SONAME в .dynstr
DT_SONAME	14	offset собственного SONAME shared object в .dynstr

После DT_NULL записей быть не должно. ABI-v3 ET_DYN, используемый как normal dependency, имеет ровно один SONAME. ET_EXEC не имеет собственного SONAME. Оба типа могут иметь ноль или более NEEDED в записанном порядке.

20. SONAME, NEEDED и link-time -L/-I

Имена development link, runtime identity и physical file разделены. Типичная установка:

```
libfoo.so -> libfoo.so.1
libfoo.so.1 -> libfoo.so.1.2.3
libfoo.so.1.2.3 # DT_SONAME = libfoo.so.1
```

При -lfoo linker находит development name libfoo.so, полностью валидирует выбранный ABI-v3 ET_DYN, читает его SONAME и записывает в потребитель именно DT_NEEDED[libfoo.so.1]. Абсолютный путь, имя symlink и physical filename в NEEDED не попадают.

ABI-v3 dynamic output в каждом search directory предпочитает libfoo.so, затем libfoo.a. Если выбран archive, нужные ET_REL members встраиваются статически и никакого NEEDED для них нет. ABI-v1 и -r shared candidate не рассматривают.

21. Поиск библиотек и runtime root

mcpu-ld и mcpu-run используют общую relocatable root/config policy. Базовый system directory равен <runtime-root>/lib; конфигурационная цепочка в порядке роста приоритета:

```
<runtime-root>/etc/mcpu-ld.conf
/etc/mcpu/mcpu-ld.conf
$HOME/.mcpu/etc/mcpu-ld.conf
```

Переменная конфигурации MCPU_LD_SYSTEM_LIBRARY_PATH заменяет system path; пустое значение отключает его. -no-config не читает config layers, но сохраняет executable-derived default. -sys-root=PATH подразумевает -no-config и использует PATH/lib. Для linker явные -L идут раньше system path; для runtime явный --library-path также имеет более высокий search priority.

22. Рекурсивная загрузка NEEDED

mcpu-run сначала читает прямые DT_NEEDED главного ET_EXEC, ищет каждый SONAME по runtime search policy, валидирует ABI-v3 ET_DYN и требует точного совпадения внутреннего DT_SONAME с запрошенным именем. Затем так же обрабатываются транзитивные NEEDED.

Граф зависимостей строится breadth-first и дедулицируется по SONAME. Direct dependencies предшествуют transitive; siblings сохраняют порядок NEEDED слева направо. Цикл вида A->B->A допустим и завершается через предварительную регистрацию SONAME.

23. .dysym/.dynstr и runtime RELA

.dysym/.dynstr являются частью startup-link ABI, в отличие от developer/debug .symtab/.strtab. DEFAULT GLOBAL/WEAK участвуют в выборе определения; LOCAL никогда не входит в interposition namespace. Lifecycle-имена _start/_life/_fini принадлежат главному executable и не являются runtime-preemptible symbols.

Runtime SHT_RELA использует тот же MELF64_Rel. sh_link указывает на .dynsym, sh_info — на allocated section с исправляемым полем, а r_offset остаётся offset относительно output section. Все runtime RELA выполняются до _start; lazy binding отсутствует.

24. Детерминированный runtime symbol scope

Для одного имени сначала сравнивается binding strength: GLOBAL всегда сильнее WEAK. Среди определений одинаковой силы действует scope:

1. explicit preload modules — наивысший scope, внутри списка правый элемент имеет больший приоритет;
2. главный ET_EXEC;
3. normal DT_NEEDED graph в breadth-first, left-to-right порядке.

Таким образом direct dependency выигрывает у конфликтующей transitive dependency той же силы, а первый sibling в NEEDED order выигрывает у следующего. Если GLOBAL нет вообще, тот же scope применяется к WEAK.

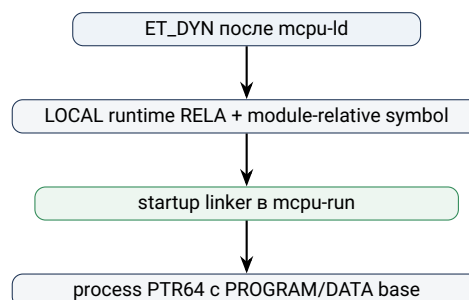
MCPU_LD_PRELOAD является environment counterpart для -ldpreload. Environment entries добавляются первыми, command-line entries — после них; существующее rightmost-wins правило поэтому естественно делает command line сильнее environment. Эта переменная не является частью mcpu-ld.conf и не подавляется -no-config.

25. Контракт ET_DYN

ABI-v2/v3 ET_DYN содержит только library .text/.data/.bss плюс non-ALLOC metadata. Он не владеет .init, .fini, .control, .input, .output, .trace, lifecycle definitions, PT_MCPU_STACK или PT_MCPU_HEAP. STACK, HEAP и model spaces принадлежат главному ET_EXEC. Для ABI v3 normal shared object SONAME обязателен.

26. LOCAL relocation внутри ET_DYN

LOCAL-символ остаётся частным и неподменяемым, но окончательная process base модуля заранее неизвестна. Поэтому relocation внутри ET_DYN, которой нужен окончательный PROGRAM/DATA address, сохраняется как LOCAL runtime RELA. Startup linker разрешает её строго внутри исходного модуля и добавляет runtime base соответствующего сегмента. Эта коррекция является нормативной частью runtime relocation contract и полностью сохраняется в ABI v3.



27. Размещение образа процесса и единственный DATA guard

У MCPU нет единого ELF-подобного смещения загрузки (load bias): PTR64 имеет независимые селекторы сегментов. Компонент запуска назначает отдельную базу для каждого архитектурного сегмента.

Вклад	База
main PROGRAM	0
lib1 PROGRAM	align_up(main_program_end, lib1_align)
lib2 PROGRAM	align_up(lib1_program_end, lib2_align)
main DATA	0
первая DATA-библиотека	после окончательного DATA main; минимум 16, если DATA main отсутствует

В ET_DYN нет собственного DATA NULL guard. Во всём образе процесса существует ровно одна нулевая область offsets 0..15. Поэтому локальный символ библиотеки с относительным к модулю DATA offset 0 после размещения при запуске закономерно получает process offset 16 или больше.

28. BSS и соотношение filesz/memsz

.bss имеет тип SHT_NOBITS. Она участвует в расчёте адресов и увеличивает DATA p_memsz, но не увеличивает объём данных

в файле `p_filesz`. Статический и `startup`-загрузчики сначала получают изначально обнулённую память, затем копируют сохранённые байты, поэтому NOBITS детерминированно начинается с нулей.

Для PROGRAM хранится только машинный код, поэтому `filesz=memsz`. Для DATA обычно `filesz<memsz`. LOCAL-ссылки из ET_DYN в DATA/.bss всё равно требуют перебазирувания по базе модуля, даже если целевые байты относятся к NOBITS.

29. Дескриптор STACK и командная строка

Размер STACK является частью ABI исполняемого файла. Он задаётся компоновщиком через `-stack-size=N` и записывается в PT_MCPU_STACK; `mcru-run` не может его переопределить. Разделяемые объекты ABI v2 не задают политику STACK.

Предпочтительный синтаксис запуска использует явный разделитель:

```
mcru-run [runtime-options] -- executable [program-args...]
```

Все параметры после `--` передаются в `guest argv` без повторного разбора программой `mcru-run`; текстовое имя исполняемого файла становится `argv[0]`. Старая форма без разделителя сохранена для совместимости.

30. Дескриптор HEAP_end и SYS_brk

Необязательный PT_MCPU_HEAP хранит начальный подготовленный размер в `p_memsz` и неизменяемый предел в `p_maxsz`; оба значения не меньше 16, причём `max >= initial`. Определяемый компоновщиком символ `_end=PTR64(HEAP, 16)` является частным для исполняемого файла, неподменяемым и задаёт начальный и минимальный `program break`.

SYS_brk имеет номер 10. NULL-аргумент запрашивает текущее значение; HEAP PTR64 задаёт абсолютный новый break. В отличие от общего `stable syscall rule -MCPU_errno`, SYS_brk возвращает фактический текущий HEAP PTR64. При отклонённом запросе break не меняется. Shrink зануляет освобождённый диапазон, но не обязан уменьшать `host`-отображение памяти; последующий рост остаётся детерминированным. Guest PTR64 остаётся корректным при перемещении `host backing`, поскольку содержит `segment+offset`, а не `host-адрес`.

31. Дисциплина проверки загрузчика

Загрузчик сначала полностью проверяет структуру файла и только затем создаёт `guest`-области памяти. В частности проверяются:

- сигнатура, класс, порядок данных, версия, OSABI, ABI/machine, размеры заголовков и границы таблиц;
- обязательные метаданные `target/lifecycle` и зарезервированные поля;
- уникальность заголовков сегментов (Program Headers), права доступа, соотношения размеров, выравнивание и границы OFF60;
- соответствие каждой ALLOC-секции её архитектурному сегменту;
- `selector/range` точек входа `lifecycle`;
- нулевые байты DATA guard в статическом исполняемом образе;
- для ABI v2/v3 — структура `dynsym/RELA`, диапазоны символов, границы исправляемых полей и семантика целевого сегмента; для ABI v3 дополнительно `canonical .dynamic`, SONAME/NEEDED и соответствие `identity` загруженного ET_DYN запрошенному SONAME.

Загрузчик проверяет, но не «чинит». Он не выводит `lifecycle` из названий секций, не создаёт недостающий guard, не меняет порядок байтов PROGRAM, не увеличивает STACK, не ослабляет HEAP limit и не переинтерпретирует несовместимый профиль целевой архитектуры. Компоновщик запуска ABI v2/v3 выполняет только явно определённые сохранённые `relocation`, `dependency discovery` и разрешение символов.

32. Режимы команды mcru-ld

```
# partial ET_REL
mcru-ld -r -o combined.o a.o b.o

# статический ABI v1
mcru-ld --stack-size=4096 --heap-size=64K --heap-limit=1M \
  -L$HOME/.mcru/lib -lmodel -o app main.o

# главный исполняемый файл frozen ABI v3
mcru-ld --dynamic --stack-size=64K --no-heap \
  -L$HOME/.mcru/lib -lrk4 -o model pendulum.o
```

```
# shared object ABI v3
mcpu-ld --shared --soname=librk4.so.1 \
        -o librk4.so.1.0.0 rk4.o
```

STACK/HEAP принадлежат только ET_EXEC. - - soname допустим только для - - shared. - r не создаёт executable/runtime metadata и использует для - l только static archives. Суффиксы размера K/KiB, M/MiB, G/GiB используют двоичные множители.

33. Инструменты анализа и безопасного редактирования MELF

Binary utilities работают поверх того же MELF contract и не создают отдельного container ABI.

33.1. Read-only inspection

mcpu-readelf показывает file/program/section headers, symbol tables, RELA, .dynamic и MCPU-specific metadata. mcpu-nm выводит .symtab или .dynsym и умеет читать static archives. mcpu-objdump объединяет section/relocation views с instruction disassembly; decoder общий с CPU runtime через __cpu_decode_at(), поэтому ISA не дублируется отдельным parser-ом.

33.2. mcpu-strip

Текущая policy намеренно консервативна. В ET_REL ordinary .symtab/.strtab сохраняются, потому что они являются частью последующей link semantics. В ET_EXEC/ET_DYN ordinary non-ALLOC SHT_SYMTAB может быть удалена только если retained sections от неё не зависят; .dynsym/.dynstr/.dynamic, RELA, MCPU metadata и allocated image всегда сохраняются. Современные final-image writers уже компактны, поэтому для обычного ET_EXEC/ET_DYN strip часто является byte-for-byte no-op.

33.3. mcpu-objcopy

mcpu-objcopy выполняет только MELF-to-MELF structural copy/edit. Safe interface умеет выбирать/удалять sections и dump/add/update/rename для metadata, но запрещает разрушать allocated image, frozen dynamic metadata, runtime relocations и ET_REL link-critical dependencies. Новая section через - - add-section является non-ALLOC SHT_PROGBITS. Metadata tail может быть repacked, но program headers и runtime-visible PT_LOAD bytes должны оставаться неизменными.

Flat binary, Intel HEX, S-record, VMA/LMA и firmware-image conversion намеренно отсутствуют: виртуальная MCPU исполняет MELF напрямую.

33.4. Non-ALLOC debug metadata и frozen ABI v3

Будущий debug format не обязан менять MELF ABI v3, если его sections остаются non-ALLOC, не входят в PT_LOAD, не требуются loader/runtime для исполнения и могут быть полностью удалены без изменения guest-visible поведения. Версия такого debug format должна быть самостоятельной и не кодироваться повышением EI_ABIVERSION MELF.

В ET_REL debug sections при необходимости могут иметь link-time RELA к code/data symbols; такие relocations являются metadata link-editor contract и не должны становиться обязательными runtime relocations final ET_EXEC/ET_DYN.

34. Правила совместимости

1. Статический ET_EXEC ABI v1 и ET_REL ABI v1 остаются основой статического/разработческого пути; ABI-v1 executable отвергает preload.
2. Исторический ABI v2 остаётся читаемым и использует explicit-preload startup model без SONAME/NEEDED.
3. Frozen ABI v3 является текущим normal dynamic ABI. Несовместимое изменение .dynamic, NEEDED/SONAME, runtime scope или retained-relocation contract требует нового номера ABI.
4. Static lib*.a и mcpu-ld - r являются link-time механизмами над ABI-v1 ET_REL и не требуют ABI v4.
5. Ревизия ISA и CALL ABI остаются равны 1 независимо от версии контейнерного dynamic ABI.
6. Будущие расширения должны использовать явные version/feature поля, а не менять смысл зарезервированных битов или записей молча. Non-ALLOC debug metadata с самостоятельной версией формата не требует нового MELF ABI, пока loader/runtime semantics не зависят от неё.

1. Размеры сериализованных записей

Запись	Байт	Назначение
MElf64_Ehdr	88	заголовок файла
MElf64_Shdr	64	заголовок секции
MElf64_Sym	24	запись таблицы символов
MElf64_Rela	24	запись RELA
MElf64_Dyn	16	запись <code>.dynamic</code> ABI v3
MElf64_Phdr	64	заголовок сегмента (Program Header) с <code>p_maxsz</code>
MElf_McpuTarget	64	метаданные профиля целевой архитектуры
MElf_McpuLifecycle	32	метаданные lifecycle
McpuStartup	32	заголовок области запуска (startup) в STACK guest-программы; это не сериализованные метаданные MELF

2. Канонические имена секций

<code>.init, .text, .fini</code>	Секции машинного кода PROGRAM.
<code>.data, .bss</code>	Объекты DATA; <code>.bss</code> имеет тип NOBITS.
<code>.control, .input, .output, .trace</code>	Модельные пространства, принадлежащие ET_EXEC.
<code>.symtab, .strtab</code>	Информация о символах для разработки и статической компоновки.
<code>.dynsym, .dynstr</code>	Runtime symbol/string tables ABI v2/v3.
<code>.dynamic</code>	Non-ALLOC dependency metadata ABI v3: SONAME/NEEDED=NULL.
<code>.rela.init, .rela.text, .rela.fini</code>	Секции relocation для PROGRAM.
RELA секций данных	<code>.rela.data, .rela.control, .rela.input, .rela.output, .rela.trace</code> .
<code>.mcpu.target</code>	Обязательный профиль целевой архитектуры.
<code>.mcpu.lifecycle</code>	Окончательный lifecycle-контракт исполняемого файла.
<code>.shstrtab</code>	Таблица строк с именами секций.

3. Нормативная памятка frozen dynamic ABI v3

- New dynamic output использует ABI v3; ABI v2 поддерживается как исторический explicit-preload input format.
- ABI-v3 ET_DYN имеет обязательный SONAME и не владеет lifecycle, model spaces, STACK или HEAP.
- ABI-v3 ET_EXEC/ET_DYN имеют canonical non-ALLOC `.dynamic`; NEEDED содержит SONAME, а не pathname или development symlink name.
- Normal dependencies обнаруживаются рекурсивно по SONAME, deduplicate/cycle handling выполняются до guest execution.
- GLOBAL сильнее WEAK; explicit preload имеет наивысший equal-strength scope и rightmost priority; затем main executable; затем breadth-first/left-to-right NEEDED graph.
- LOCAL никогда не участвует в name interposition; нуждающиеся в process base LOCAL ET_DYN relocations сохраняются и rebased внутри исходного модуля.
- Все runtime relocation и symbol resolution заканчиваются до `_start`; PLT/GOT, lazy binding, `dlopen()`, `dlsym()` и `dlclose()` отсутствуют.
- Во всём process DATA существует один NULL guard; отдельный ET_DYN guard не создаётся.
- Static archives и `mcpu - ld - r` работают на link-time поверх ET_REL и не меняют frozen ABI v3.