Maple Engine Instruction Format

Contents

1. Introduction

The code generator for Maple Engine uses Maple IR output from Maple compiler as the input to generate executable code for Maple Engine. The code is generated in assembly and assembled into ELF binary and executed by the Maple Engine.

The generated code is comprised of various ELF sections that contains Maple Engine Instructions, memory layout and meta data for classes and objects. This document describes the Maple Engine instruction set and format. For information on the contents of the generated image for execution by Maple Engine, including text and data sections, memory layout and meta data, please refer to documents “Metadata in .so Files” and “So Importer”.

2. Maple Engine Opcodes

Maple Engine Instructions contains an 8 bit opcode. Maple Engine opcode is logically independent of Maple IR opcode although it currently reuses the MapleIR opcode set (opcodes up to and including OP_addrofpc (154), are the same as MapleIR opcodes) and extends it with additional opcodes specific for the Maple Engine.

All Maple Engine instructions up to and include opcode OP_addroffpc (154) are 4 byte instructions except:

- OP_ireadpcoff 4 byte instruction + 4 byte integer relative offset to target symbol
- OP_addroffpc 4 byte instruction + 4 byte integer relative offset to target symbol
- OP_rangegoto 4 byte instruction + 4 byte tagoffset +
  jump table of vector <uint16 jmptbl idx, uint16 offset>
- OP_javatry 4 byte instruction + vector of offsets to catches
- OP_javacatch 4 byte instruction + vector of exception types
- OP_addroffunc 4 bytes instruction + 8 byte address

Additionally, xxx32/xxx64 indicate 4 byte instructions followed by additional 32/64 bit of data

- OP_addrof32
- OP_constval64
- OP_iassignoff32
- OP_brtrue32
- OP_brfalse32
- OP_goto32

3. Maple Engine Instruction Format

The base MapleRE instruction format, mre_instr_t, is a 4 byte overlay on Maple IR base_node_t to support packing instruction parameters into the instruction word to reduce image size.

```
// MapleIR base node
struct base_node_t {    // 4 bytes
  Opcode op : 8;
  PrimType ptyp : 8;
  uint8 typeflag;           // a flag to speed up type related operations in the VM
  uint8 numopnds : 8;    // only used for N-ary operators, switch and rangegoto
};


// MapleRE base instruction format
struct mre_instr_t {      // 4 bytes
  RE_Opcode op : 8;       // Runtime Engine opcode
  PrimType ptyp : 8;
```

```
  union {
    int16 value;
    int16 offset;            // ireadoff, iassignoff
    int16 frameIdx;          // regread, regassign, addrof
    uint16 numCases;         // javatry numcatches, javacatch numextypes, rangegoto numcases
    struct {
      uint8 intrinsicId;
      uint8 numopnds;
    } intrinsic;             // intrinsiccall
    struct {
      uint8 opPtyp;
      uint8 numopnds;
    } type;                  // cmp*, eq, ge, gt, le, lt, ne, ceil, floor, retype, round, trunc, cvt
    union {
      int8   i8;
      uint8  u8;
      int16  i16;
      uint16 u16;
    } constval;              // constval
    struct {
      uint8 boffset;
      uint8 bsize;
    } extractbits;           // extractbits, zext, sext
  } param;                   // instruction parameters

  base_node_t *bn() {  // for accessing base_node_t instruction overlay
    return reinterpret_cast<base_node_t *>(this);
  }
};
```

4. IR Lowering for Maple Engine

- iread/iassign
  - lower to OP_ireadoff/OP_iassignoff/OP_ireadoff32/OP_iassignoff32
- dread/dassign
  - Target symbol is formal arg or local var:
    Generate dread/dassign mre_instr_t that includes 16 bit frame slot index to the formal arg or local var.
  - Target symbol is extern, or global defined within module:
    Lower to address of symobl plus iread, i.e. ireadpcoff/addroffpc + iread, depending on whether symbol is extern or global.
- addrof
  - The addrof instruction is an expr (operand) to MPL_CLINIT_CHECK:
    Lower to addroffpc with _PTR prepended to sym name
  - Target symbol is extern and in set of class info objects for arrays predefined in libmaplert.so:

Lower to RE_addrof32 with 32 bit PC relative offset to GOT entry address (sym@GOTPCREL)
    - Target symbol is extern and neither of above:
      Lower to OP_ireadpcoff with 32 bit pc relative offset to symbol with _PTR prefix prepended (which will be resolved by maple linker at load tim)
    - Target symbol is formal arg or local var:
      Generate 4 byte addrof mre_instr_t that includes 16 bit frame slot index to the formal arg or local var
    - Target symbol is global defined within module:
      Lower to addroffpc with 32 bit int relative offset to symbol
- Call
    - Function to call is in intrinsic call list:
      Lower to intrinsiccall.
    - Function not in intrinsic call list:
      Lower to addroffunc + icall
    - For Java only:
        - Function to call is MCC_CallFastNative:
          Lower to icall that calls target native func directly.
        - Function to call is MCC_CallSlowNative0-9 and ext:
          Lower to MCCallSlowNative intrinsiccall to call target native func directly.

5. Instruction Specifics

5.1 Rangegoto

IR Syntax:
rangegoto (<opnd0> <tag-offset> {
  <intconst0>: goto <label0>
  <intconst1>: goto <label1>
    ...
  <intconstn>: goto <labeln> }

<opnd0> is of integer type. After the 4B instruction word come the int32 <tag-offset> word followed by a table of pairs of uint16 integer idx value (starting from 0) and uint16 offset. The number of jump table entries is in the 4B instruction word at param.numCases.

In searching the table for a match during execution, the evaluated value of <opnd0> minus <tag-offset> is used as index into the jump table for corresponding entry. There must be no gap in the constant integer values specified, and a match is guaranteed within the range of specified constant integer values, which means the code generator can omit generation of out-of-range checks

```
.byte OP_rangegoto, 0x0, 0x3a, 0x0
.byte 0x41, 0x0, 0x0, 0x0
.byte 0x0, 0x0
.2byte mirbin_label_28903_2-.
.byte 0x1, 0x0
```

```
.2byte mirbin_label_28903_2-.
.byte 0x2, 0x0
.2byte mirbin_label_28903_2-.
.byte 0x3, 0x0
.2byte mirbin_label_28903_2-.
….
….
```

5.2 Mapping MapleIR formal args and auto vars to Maple Engine Stack Frame Index

Formal arguments and auto variables in MapleIR functions are referenced in MapleIR instructions as registers using reg idx or var names using symbol idx. For Maple Engine, these are mapped to idx (frameidx) in the function's stack frame as maintained by Maple Engine.

References to MapleIR pseudo-registers and var symbols in formal args are mapped to +ve numbers starting from 1, and auto vars are mapped to -ve numbers starting from -2. %%retval0 and %%thrownval are special cases that map to 0 and -1.

6. OpCode Listing

Please refer to “maple engine opcodes” spreadsheet.

7. Appendix

7.1 Debug Trace

Debug tracing output can be enabled by setting the environment variable, MAPLE_ENGINE_DEBUG to trace just Maple Engine instructions, methods and functions, or everything.

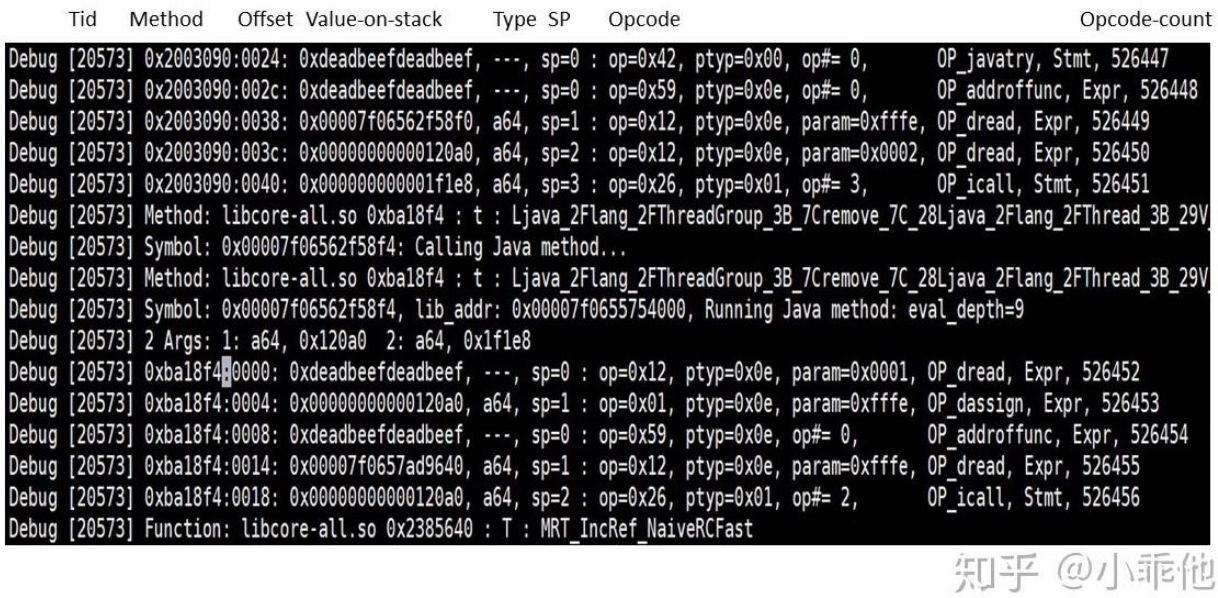
1. export MAPLE_ENGINE_DEBUG=all
2. export MAPLE_ENGINE_DEBUG=method
3. export MAPLE_ENGINE_DEBUG=instruction

Using output from setting 1 (all), tools are available to help trace the high level language (Java in this example) source code (from which the Maple Engine code was generated from) being executed:

7.1.1 engine-backtrace.sh

Usage: ./engine-backtrace.sh <filename>

- Using debug build of Maple Engine, set environment variable MAPLE_ENGINE_DEBUG=all, then run test app and capture output to file. The captured output will be as below:



- Run eng-backtrace.sh script on test output
  ./engine-backtrace.sh <output.txt>
- This will generate a file of the high level language (Java in this example) backtrace per thread, e.g. test.log.tid-<xxxxxx>.backtrace:

```
Ljava_2Futil_2Fzip_2FZipFile_3B_7C_3Cclinit_3E_7C_28_29V { // libcore-all.so 0x81202c
}
Ljava_2Futil_2Fzip_2FInflater_3B_7C_3Cclinit_3E_7C_28_29V { // libcore-all.so 0x8be710
}
Ljava_2Futil_2Fzip_2FDeflater_3B_7C_3Cclinit_3E_7C_28_29V { // libcore-all.so 0x8203cc
}
Ljava_2Fio_2FFileDescriptor_3B_7C_3Cclinit_3E_7C_28_29V { // libcore-all.so 0xa857a8
. Ljava_2Fio_2FFileDescriptor_3B_7CdupFd_7C_28I_29Ljava_2Fio_2FFileDescriptor_3B { // libcore-all.so 0xa858fc
. . Ljava_2Fio_2FFileDescriptor_3B_7C_3Cclinit_3E_7C_28_29V { // libcore-all.so 0xa857a8
. . . Ljava_2Fio_2FFileDescriptor_3B_7CdupFd_7C_28I_29Ljava_2Fio_2FFileDescriptor_3B { // libcore-all.so 0xa858fc
. . . . Ljava_2Fio_2FFileDescriptor_3B_7C_3Cinit_3E_7C_28I_29V { // libcore-all.so 0xa30378
. . . . }
. . . . Landroid_2Fsystem_2FOsConstants_3B_7C_3Cclinit_3E_7C_28_29V { // libcore-all.so 0x8244ec
. . . . | Landroid_2Fsystem_2FOsConstants_3B_7Cplaceholder_7C_28_29I { // libcore-all.so 0x81be2c
. . . . | }
. . . . | Landroid_2Fsystem_2FOsConstants_3B_7Cplaceholder_7C_28_29I { // libcore-all.so 0x81be2ccation>:
```

7.1.2 add-loc-info.sh

Usage: ./add-loc-info.sh <trace-file> <location> [--vim]

<location info extracted from instruction trace in format <method-addr>:<offset>, e.g. 0xf46e90:0074

This script finds the file and line of high level source code (Java in this example) that corresponds to a generated Maple Engine instruction. Using the method:offset address in a execution trace as in 7.1.1, running add-loc-info.sh locates the source file name, path and line number that corresponds to the

generated instruction:

```
eching@figo:  ./add-loc-info.sh <output.txt>  <0x819f14:0054>
filename=Thread.java
linenumber=1596
/home/eching/maple//out/../../android-8.1.0_r69/libcore/ojluni/src/main/java/java/lang/Thread.java
0x819f14:0054 --> /home/eching/aosp/android-
8.1.0_r69/libcore/ojluni/src/main/java/java/lang/Thread.java:1596
done.
```

7.2 Sample generated code

```
168       .section       .rodata
169       .p2align       2
170 __method_desc__LE123__E_3B_7C_3Cinit_3E_7C_28_29V:
171       .long __methods_compact__LE123__E_3B+1-.
172       .word 16
...       ...
...       ...
178       .long __method_desc__LE123__E_3B_7C_3Cinit_3E_7C_28_29V-.
179 LE123__E_3B_7C_3Cinit_3E_7C_28_29V:
180       // mir2bin func begin: ========================
181       .cfi_startproc
182       .cfi_personality 155, DW.ref.__mpl_personality_v0
183       .ascii "MPLI"
184 LE123__E_3B_7C_3Cinit_3E_7C_28_29V_mirbin_info:
185       .long LE123__E_3B_7C_3Cinit_3E_7C_28_29V_mirbin_code - .
186       .word 1, 3, 2, 0 // func storage info
187       // PrimType of formal arguments
188       .byte 0xe      // %1
189       // PrimType of automatic variables
190       .byte 0x1      // %%retval
191       .byte 0x29     // %%thrownval
192       .byte 0xe      // %2
193       .p2align 1
194 LE123__E_3B_7C_3Cinit_3E_7C_28_29V_mirbin_code:
195       // LINE E123_E.java : 4, DEX_INSTIDX : 0 ||0000: invoke-direct: Ljava/lang/Exception;.<init>:()V
// method@0014
196       // muid_func_undef:Ljava_2Flang_2FException_3B_7C_3Cinit_3E_7C_28_29V
197       .byte OP_addroffpc, 0xe, 0x0, 0x0              // 0000
198       .long __muid_func_undef_tab$$Neg_dex-.
199       .byte OP_constval, 0xe, 0x40, 0x0              // 0008
...       ...
...       ...
212       // LINE E123_E.java : 5, DEX_INSTIDX : 7 ||0007: return-void: <no-index>
213       .byte OP_intrinsiccall, 0x0, 0x27, 0x0         // 0030: MPL_CLEANUP_LOCALREFVARS
214       .byte OP_return, 0x0, 0x0, 0x0                 // 0034
215       .cfi_endproc
216 .label.end.LE123__E_3B_7C_3Cinit_3E_7C_28_29V:
217       // mir2bin func end: ========================
```

Example of generated code:
lines 168-178: For Java, meta data for Maple Engine runtime to locate class method
lines 186-214: Maple Engine executable code for MapleIR function
LE123__E_3B_7C_3Cinit_3E_7C_28_29V
lines 186-192: Stack frame description for function LE123__E_3B_7C_3Cinit_3E_7C_28_29V
lines 186: Function has 1 formal arugment and 3 local variables
lines 187-192: Primy data type and name of the function's formal args and local vars.

7.3 Heap Usage Stats

Set environment variable "export MAPLE_SHOW_HEAP_USAGE=1" before invoking Java apps with Maple shell, mplsh, to enable display info on heap usage before and after running app. Unset to disable.

The following is a sample output from running Java HelloWorld:

```
[STATS] [HEAPSTATS] Heap Usage: Thu Mar  5 19:07:29 2020
[STATS] [HEAPSTATS]   High watermark (bytes):   11426
[STATS] [HEAPSTATS]   Net # object bytes in use:  10940
[STATS] [HEAPSTATS]   Net # objects in use:       214
[STATS] [HEAPSTATS]   Running total # objects allocated:       275
[STATS] [HEAPSTATS]   Running total # objects freed:           61
[STATS] [HEAPSTATS]   Running total # object bytes allocated: 15503
[STATS] [HEAPSTATS]   Running total # object bytes freed:     4563
Hello World!
[STATS] [HEAPSTATS] Heap Usage: Thu Mar  5 19:07:29 2020
[STATS] [HEAPSTATS]   High watermark (bytes):   41612
[STATS] [HEAPSTATS]   Net # object bytes in use:  40815
[STATS] [HEAPSTATS]   Net # objects in use:       353
[STATS] [HEAPSTATS]   Running total # objects allocated:       463
[STATS] [HEAPSTATS]   Running total # objects freed:           110
[STATS] [HEAPSTATS]   Running total # object bytes allocated: 47474
[STATS] [HEAPSTATS]   Running total # object bytes freed:     6659
```

8 JavaScript Support

8.1 Module Declarations

- globalmememsize - Size of global memory block in bytes for storing global static variables
- globalmemmap - Static init val of global memory block - vector of 32 bit int constants
- globalwordstypetagged - Bit vector init to list of 32 bit int constants (see usage in MapleIR Spec)
- globalwordsrefcounted - Bit vector init to list of 32 bit int constants (see usage in MapleIR Spec)

8.2 Function Declarations

- upformalsize – Byte size of upformal segment of formal parameters passed above frame ptr %FP
- framesize – Size of function stack frame in bytes
- formalwordstypetagged - Bit vector init to list of 32 bit int constants (see usage in MapleIR Spec)
- formalwordsrefcounted - Bit vector init to list of 32 bit int constants (see usage in MapleIR Spec)
- localwordstypetagged - Bit vector init to list of 32 bit int constants (see usage in MapleIR Spec)
- localwordsrefcounted- Bit vector init to list of 32 bit int constants (see usage in MapleIR Spec)

8.3 OP_regread and OP_regassign

mpl_be loweres .mpl output from js2mpl to .mmpl, which is used as input by Maple Engine code generator to generate maple engine image. Such lowered .mmpl uses a limited set of special pseudo

registers (%%GP, %%FP, %%retval, %%thrownval, etc), and do not use general pseudo registers (%1, %2, etc). So the frameIdx field in regread and regassign instructions in JavaScript image generated for Maple Engine is set to the pseudo register number (-3 = %%GP, -6 = %%retval, etc. see mir_preg.h) instead of the mapped value used in Maple Engine images generated from Java code.

8.4 Sample Maple Engine image generated from .mmpl input

.mmpl input

```
 1 flavor 4
 2 srclang 2
 3 id 65535
 4 globalmemsize 72
 5 globalmemmap = [ 0x0 0x0 0x0 0x0 0xb0000 0x64646120 0x6170203a 0xa7373 0x74001f00
0x20747365 0x6c696166 0x76206465 0x78652031 0x74636570 0x62203320 0x67207475 0x74    65
0xa0001 ]
 6 globalwordstypetagged = [ 0x5 ]
 7 globalwordsrefcounted = [ 0x0 ]
 8 numfuncs 2
 9 entryfunc &main
10 func &Add {
11   funcid 0
12   upformalsize 24
13   formalwordstypetagged = [ 0x15 ]
14   formalwordsrefcounted = [ 0x0 ]
15   framesize 24
16   localwordstypetagged = [ 0x28 ]
17   localwordsrefcounted = [ 0x0 ]
18
19   intrinsiccall JSOP_ADD (ireadfpoff dynany 8, ireadfpoff dynany 16)
20   iassignfpoff dynany -24 (regread dynany %%retval0)
21   iassignfpoff dynany -16 (ireadfpoff dynany -24)
22   regassign dynany %%retval0 (ireadfpoff dynany -16)
23   return ()
24   regassign i32 %%retval0 (constval i32 0)
25   return ()
26 }
```

Sample .s generated from .mmpl file:

```
172       .section      .rodata
173       .p2align 3
174       .global __mpljs_module_decl__
175 __mpljs_module_decl__:
176       .word 72      // globalMemSize bytes
177       .byte 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0xb,
0x0, 0x20, 0x61, 0x64, 0x64, 0x3a, 0x20, 0x70, 0x61, 0    x73, 0x73, 0xa, 0x0, 0x0, 0x1f, 0x0, 0x74, 0x65,
0x73, 0x74, 0x20, 0x66, 0x61, 0x69, 0x6c, 0x65, 0x64, 0x20, 0x76, 0x31, 0x20, 0x65, 0x78, 0x70, 0x65,
0x63, 0x74,    0x20, 0x33, 0x20, 0x62, 0x75, 0x74, 0x20, 0x67, 0x65, 0x74, 0x0, 0x0, 0x1, 0x0, 0xa, 0x0
// globalMemMap
178       .word 4                         // globalwordstypetagged/refcounted byte count
179       .byte 0x5, 0x0, 0x0, 0x0        // globalwordstypetagged
180       .byte 0x0, 0x0, 0x0, 0x0        // globalwordsrefcounted
181
182       .text
183       .p2align 2
184       .globl  Add
185       .type   Add, %function
186 Add:
187       // mir2bin func begin: ========================
188       .cfi_startproc
189       .ascii "MPJS"
190 Add_mirbin_info:
191       .long Add_mirbin_code - .
192       .word 24, 24, 2, 0     // upFormalSize, frameSize, evalStackDepth
193       .word 4, 4             // formalWords bit vector byte count, localWords bit vector byte count
194       .byte 0x15, 0x0, 0x0, 0x0     // formalWordsTypeTagged
195       .byte 0x0, 0x0, 0x0, 0x0      // formalWordsRefCounted
196       .byte 0x28, 0x0, 0x0, 0x0     // localWordsTypeTagged
197       .byte 0x0, 0x0, 0x0, 0x0      // localWordsRefCounted
198       .p2align 1
199 Add_mirbin_code:
200       .byte OP_ireadfpoff, 0x16, 0x8, 0x0          // 0010
201       .byte OP_ireadfpoff, 0x16, 0x10, 0x0         // 0014
202       .byte OP_intrinsiccall, 0x0, 0x32, 0x2       // 0018: JSOP_ADD
203       .byte OP_regread, 0x16, 0xfa, 0xff           // 001c: %-6
204       .byte OP_iassignfpoff, 0x16, 0xe8, 0xff      // 0020
205       .byte OP_ireadfpoff, 0x16, 0xe8, 0xff        // 0024
206       .byte OP_iassignfpoff, 0x16, 0xf0, 0xff      // 0028
…
…
212       .byte OP_return, 0x0, 0x0, 0x0               // 0040
213       .cfi_endproc
214 .label.end.Add:
215       // mir2bin func end: ========================
```