V8中的trampline和Embedded builtins

陆亚涵

中科院软件所 PLCT实验室

2021-05-28

1.Trampoline机制

2.Embedded builtins

3.Short builtin calls

1.Trampoline机制

1.Trampoline机制

Backgroud

riscv64 expample:

```
1          addi   a0, a1, 1
2
3          bne    a0, zero_reg, 4
4
5          j       L1
6
7   L2:    addi   a0, a1, 1
8
9          addi  a0, a2,  2
10
11         ...
12
13         j       L2
14
15         ...
16  L1 :   ret
```

1.Trampoline机制

Backgroud

riscv64 expample:

```
1         addi   a0, a1, 1
2
3         bne    a0, zero_reg, 4
4
5         j      L1
6
7   L2:   addi   a0, a1, 1
8
9         addi  a0, a2,  2
10
11        ...
12
13        j      L2
14
15        ...
16  L1 :  ret
```

对于 Gcc/LLVM等编译静态语言如c++/c, 可以扫描整个汇编程序后确定j的offset后再选择将伪指令j汇编成near jump 还是far jump

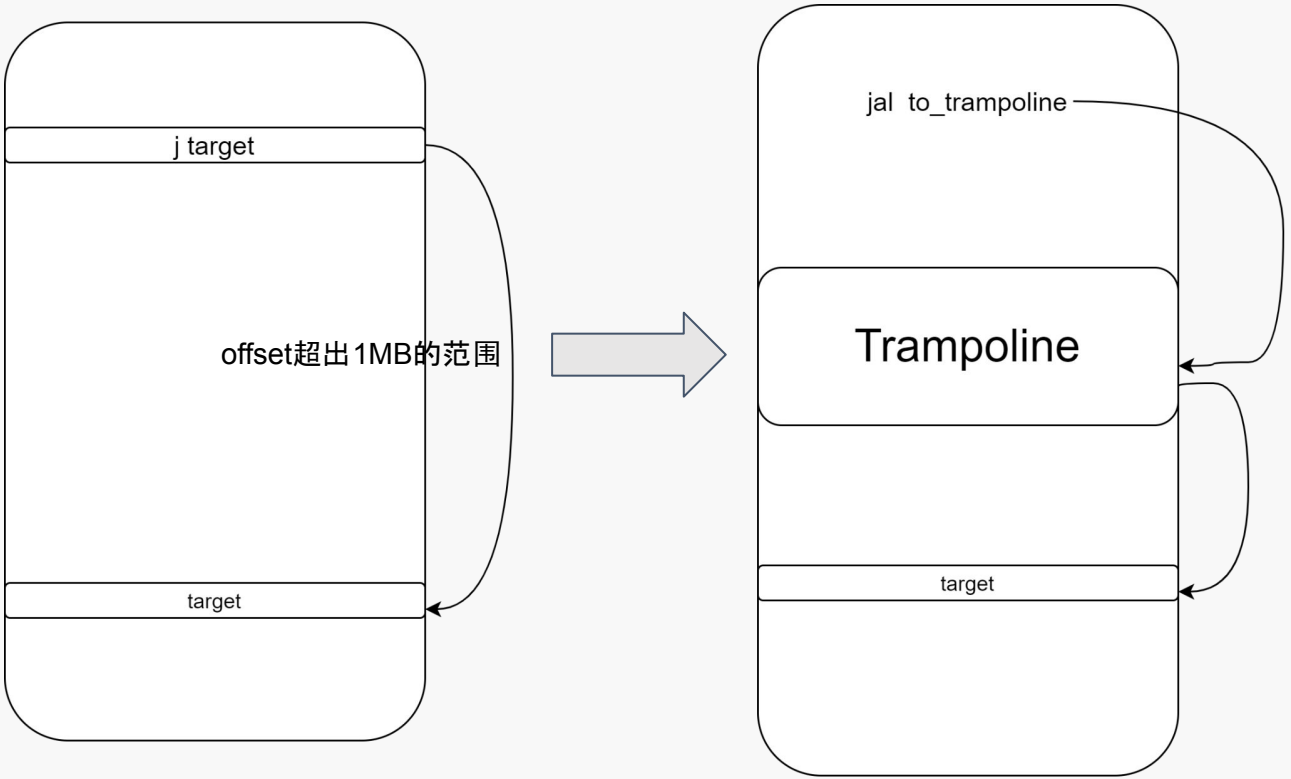
但对于JS语言来说,V8等引擎都是对JS代码进行JIT编译的,只会对V8生成的IR扫描一次就生成汇编代码.

表1 不同架构直接跳转的范围

x86	arm64	mips64r6	riscv64	mips64r2
Jmp: ±2GB	B/BL: ±128MB	Bc: ±128MB	jal:(±1 MiB)	B: ±128KB
	B.EQ: ± 1MB.	BEQ:(±32KB)	beq:(±4096 B)	BEQ:(±32KB)

1.Trampoline机制

Backgroud



1.Trampoline机制

V8实现

```
void TurboAssembler::Branch(Label* L, Condition cond, Register rs,
                            const Operand& rt) {
  if (L->is_bound()) {
    if (!BranchShortCheck(0, L, cond, rs, rt)) {
      if (cond != cc_always) {
        Label skip;
        Condition neg_cond = NegateCondition(cond);
        BranchShort(&skip, neg_cond, rs, rt);
        BranchLong(L);
        bind(&skip);
      } else {
        BranchLong(L);
        EmitConstPoolWithJumpIfNeeded();
      }
    }
  } else {
    if (is_trampoline_emitted()) {
      if (cond != cc_always) {
        Label skip;
        Condition neg_cond = NegateCondition(cond);
        BranchShort(&skip, neg_cond, rs, rt);
        BranchLong(L);
        bind(&skip);
      } else {
        BranchLong(L);
        EmitConstPoolWithJumpIfNeeded();
      }
    } else {
      BranchShort(L, cond, rs, rt);
    }
  }
}
```

1.Trampoline机制

实现

```
void TurboAssembler::BranchShortHelper(int32_t offset, Label* L) {
  DCHECK(L == nullptr || offset == 0);
  offset = GetOffset(offset, L, OffsetSize::kOffset21);
  j(offset);
}

void TurboAssembler::BranchShort(int32_t offset) {
  DCHECK(is_int21(offset));
  BranchShortHelper(offset, nullptr);
}

void TurboAssembler::BranchShort(Label* L) { BranchShortHelper(0, L); }

int32_t TurboAssembler::GetOffset(int32_t offset, Label* L, OffsetSize bits) {
  if (L) {
    offset = branch_offset_helper(L, bits);
  } else {
    DCHECK(is_intn(offset, bits));
  }
  return offset;
}
```

```
int32_t Assembler::branch_offset_helper(Label* L, OffsetSize bits) {
  int32_t target_pos;

  DEBUG_PRINTF("branch_offset_helper: %p to %p (%d)\n", L,
               reinterpret_cast<Instr*>(buffer_start_ + pc_offset()),
               pc_offset());
  if (L->is_bound()) {
    target_pos = L->pos();
    DEBUG_PRINTF("\tbound: %d", target_pos);
  } else {
    if (L->is_linked()) {
      target_pos = L->pos();
      L->link_to(pc_offset());
      DEBUG_PRINTF("\tadded to link: %d\n", target_pos);
    } else {
      L->link_to(pc_offset());
      if (!trampoline_emitted_) {
        unbound_labels_count_++;
        next_buffer_check_ -= kTrampolineSlotsSize;
      }
      DEBUG_PRINTF("\tstarted link\n");
      return kEndOfJumpChain;
    }
  }

  int32_t offset = target_pos - pc_offset();
  DCHECK(is_intn(offset, bits));
  DCHECK_EQ(offset & 1, 0);
  DEBUG_PRINTF("\toffset = %d\n", offset);
  return offset;
}
```

1.Trampoline机制

V8实现

```
void Assembler::emit(Instr x) {
  if (!is_buffer_growth_blocked()) {
    CheckBuffer();
  }
  DEBUG_PRINTF("%p: ", pc_);
  disassembleInstr(x);
  EmitHelper(x);
  CheckTrampolinePoolQuick();
}
```

```
void CheckTrampolinePoolQuick(int extra_instructions = 0) {
  DEBUG_PRINTF("\tpc_offset:%d %d\n", pc_offset(),
               next_buffer_check_ - extra_instructions * kInstrSize);
  if (pc_offset() >= next_buffer_check_ - extra_instructions * kInstrSize) {
    CheckTrampolinePool();
  }
}
```

```
// for BlockTrampolinePoolScope buffer.
next_buffer_check_ = FLAG_force_long_branches
                     ? kMaxInt
                     : kMaxBranchOffset - kTrampolineSlotsSize * 16;
```

1.Trampoline机制

实现

```
void Assembler::CheckTrampolinePool() {
  // Some small sequences of instructions must not be broken up by the
  // insertion of a trampoline pool; such sequences are protected by setting
  // either trampoline_pool_blocked_nesting_ or no_trampoline_pool_before_,
  // which are both checked here. Also, recursive calls to CheckTrampolinePool
  // are blocked by trampoline_pool_blocked_nesting_.
  DEBUG_PRINTF("\tpc_offset %d no_trampoline_pool_before:%d\n", pc_offset(),
               no_trampoline_pool_before_);
  DEBUG_PRINTF("\ttrampoline_pool_blocked_nesting:%d\n",
               trampoline_pool_blocked_nesting_);
  if ((trampoline_pool_blocked_nesting_ > 0) ||
      (pc_offset() < no_trampoline_pool_before_)) {
    // Emission is currently blocked; make sure we try again as soon as
    // possible.
    if (trampoline_pool_blocked_nesting_ > 0) {
      next_buffer_check_ = pc_offset() + kInstrSize;
    } else {
      next_buffer_check_ = no_trampoline_pool_before_;
    }
    return;
  }

  DCHECK(!trampoline_emitted_);
  DCHECK_GE(unbound_labels_count_, 0);
```

```
if (unbound_labels_count_ > 0) {
  // First we emit jump, then we emit trampoline pool.
  {
    DEBUG_PRINTF("inserting trampoline pool at %p (%d)\n",
                 reinterpret_cast<Instr*>(buffer_start_ + pc_offset()),
                 pc_offset());
    BlockTrampolinePoolScope block_trampoline_pool(this);
    Label after_pool;
    j(&after_pool);

    int pool_start = pc_offset();
    for (int i = 0; i < unbound_labels_count_; i++) {
      int64_t imm64;
      imm64 = branch_long_offset(&after_pool);
      DCHECK(is_int32(imm64));
      int32_t Hi20 = (((int32_t)imm64 + 0x800) >> 12);
      int32_t Lo12 = (int32_t)imm64 << 20 >> 20;
      auipc(t6, Hi20);  // Read PC + Hi20 into t6
      jr(t6, Lo12);     // jump PC + Hi20 + Lo12
    }
    // If unbound_labels_count_ is big enough, label after_pool will
    // need a trampoline too, so we must create the trampoline before
    // the bind operation to make sure function 'bind' can get this
    // information.
    trampoline_ = Trampoline(pool_start, unbound_labels_count_);
    bind(&after_pool);

    trampoline_emitted_ = true;
    // As we are only going to emit trampoline once, we need to prevent any
    // further emission.
    next_buffer_check_ = kMaxInt;
  }
} else {
  // Number of branches to unbound label at this point is zero, so we can
  // move next buffer check to maximum.
  next_buffer_check_ =
      pc_offset() + kMaxBranchOffset - kTrampolineSlotsSize * 16;
}
return;
```

1.Trampoline机制

实现

```
                -- B17 start --
0x4443328246c   4ec  02071293       slli      t0, a4, 32
0x44432824 70   4f0  7f8b0d13       addi      s10, s6, 2040
0x44432824 74   4f4  768d3383       ld        t2, 1896(s10)
0x44432824 78   4f8  02039393       slli      t2, t2, 32
0x4443328247c   4fc  227284e3       beq       t0, t2, 2600 -> 0x4443282ea4  (Call_ReceiverIsNullOrUndefined_Baseline_Compact)
                -- B18 start --
0x4443282ea0    f20  0740006f       j         116 -> 0x4443282f14  (Call_ReceiverIsNullOrUndefined_Baseline_Compact)
0x4443282ea4    f24  00000f97       auipc     t6, 0x0
0x4443282ea8    f28  698f8067       jalr      zero_reg, 1688(t6) -> 0x444328353c  (Call_ReceiverIsNullOrUndefined_Baseline_Compact)
0x4443282eac    f2c  00001f97       auipc     t6, 0x1
0x4443282eb0    f30  2c8f8067       jalr      zero_reg, 712(t6) -> 0x4443284174  (Call_ReceiverIsNullOrUndefined_Baseline_Compact)
0x4443282eb4    f34  00001f97       auipc     t6, 0x1
0x4443282eb8    f38  334f8067       jalr      zero_reg, 820(t6) -> 0x44432841e8  (Call_ReceiverIsNullOrUndefined_Baseline_Compact)
0x4443282ebc    f3c  00001f97       auipc     t6, 0x1
0x4443282ec0    f40  330f8067       jalr      zero_reg, 816(t6) -> 0x44432841ec  (Call_ReceiverIsNullOrUndefined_Baseline_Compact)
0x4443282ec4    f44  00001f97       auipc     t6, 0x1
0x4443282ec8    f48  354f8067       jalr      zero_reg, 852(t6) -> 0x4443284218  (Call_ReceiverIsNullOrUndefined_Baseline_Compact)
0x4443282ecc    f4c  00001f97       auipc     t6, 0x1
0x4443282ed0    f50  378f8067       jalr      zero_reg, 888(t6) -> 0x44432842 44  (Call_ReceiverIsNullOrUndefined_Baseline_Compact)
0x4443282ed4    f54  00001f97       auipc     t6, 0x1
0x4443282ed8    f58  428f8067       jalr      zero_reg, 1064(t6) -> 0x44432842fc  (Call_ReceiverIsNullOrUndefined_Baseline_Compact)
0x4443282edc    f5c  00001f97       auipc     t6, 0x1
0x4443282ee0    f60  44cf8067       jalr      zero_reg, 1100(t6) -> 0x4443284328  (Call_ReceiverIsNullOrUndefined_Baseline_Compact)
0x4443282ee4    f64  00001f97       auipc     t6, 0x1
0x4443282ee8    f68  470f8067       jalr      zero_reg, 1136(t6) -> 0x44432843 54  (Call_ReceiverIsNullOrUndefined_Baseline_Compact)
0x4443282eec    f6c  00001f97       auipc     t6, 0x1
0x4443282ef0    f70  54cf8067       jalr      zero_reg, 1356(t6) -> 0x4443284438  (Call_ReceiverIsNullOrUndefined_Baseline_Compact)
0x4443282ef4    f74  00000f97       auipc     t6, 0x0
0x4443282ef8    f78  020f8067       jalr      zero_reg, 32(t6) -> 0x4443282f14  (Call_ReceiverIsNullOrUndefined_Baseline_Compact)
                -- B94 start --
0x444328353c    15bc  fe843b83       ld        s7, -24(fp)
                [ DecompressTaggedPointer
0x4443283540    15c0  fffbe703       lwu       a4, -1(s7)
0x4443283544    15c4  00ed8733       add       a4, s11, a4
                ]
0x4443283548    15c8  00010993       mv        s3, sp
0x444328354c    15cc  ff810113       addi      sp, sp, -8
0x4443283550    15d0  ff017113       andi      sp, sp, 0xfffffff0
0x4443283554    15d4  01313023       sd        s3, 0(sp)
0x4443283558    15d8  fce43023       sd        a4, -64(fp)
```

2. Embedded builtins

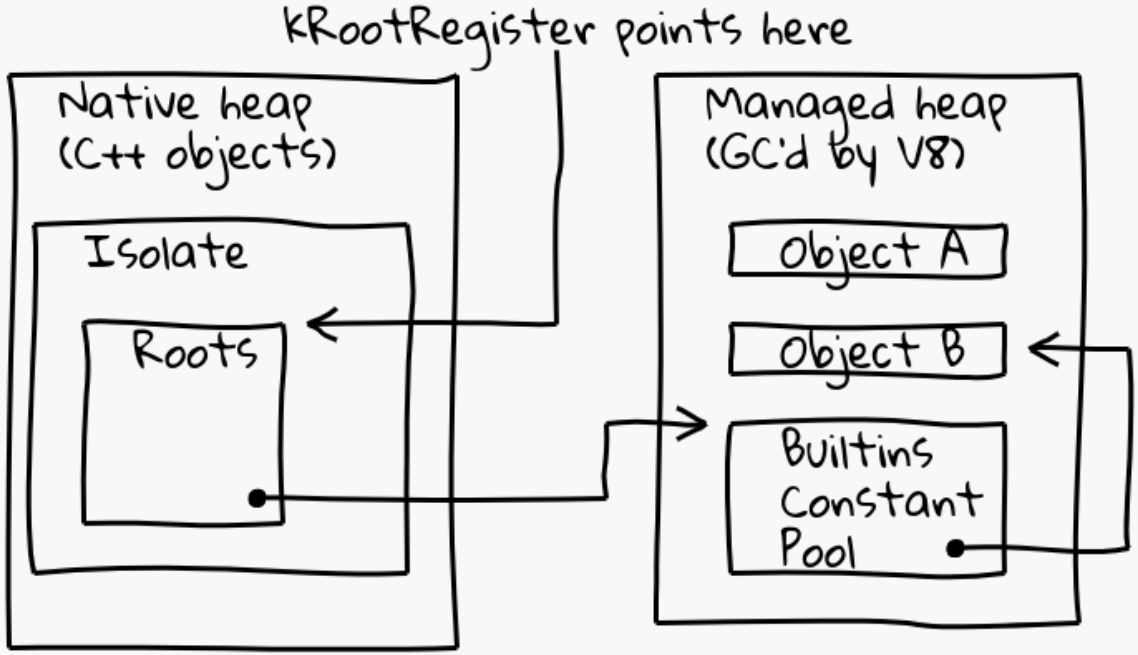
2.Embedded builtins

Backgroud

V8 snapshot size (including builtins) from 2015 until 2017
https://v8.dev/blog/embedded-builtins

2018年,spectre漏洞的发现, Chrome 浏览器开启了 site isolation(站点隔离)特性. 因此造成浏览器中的每一个网页都会创建大量的渲染线程和更多的V8 isolates实例,每个实例都包含一份builtin代码.

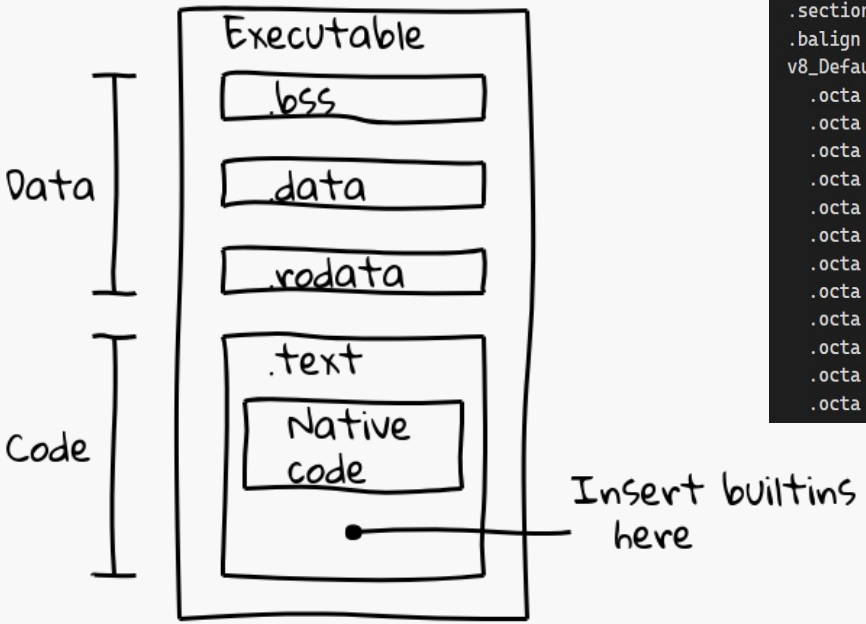
2.Embedded builtins



Isolate- and process-independent code

https://v8.dev/blog/embedded-builtins

2.Embedded builtins



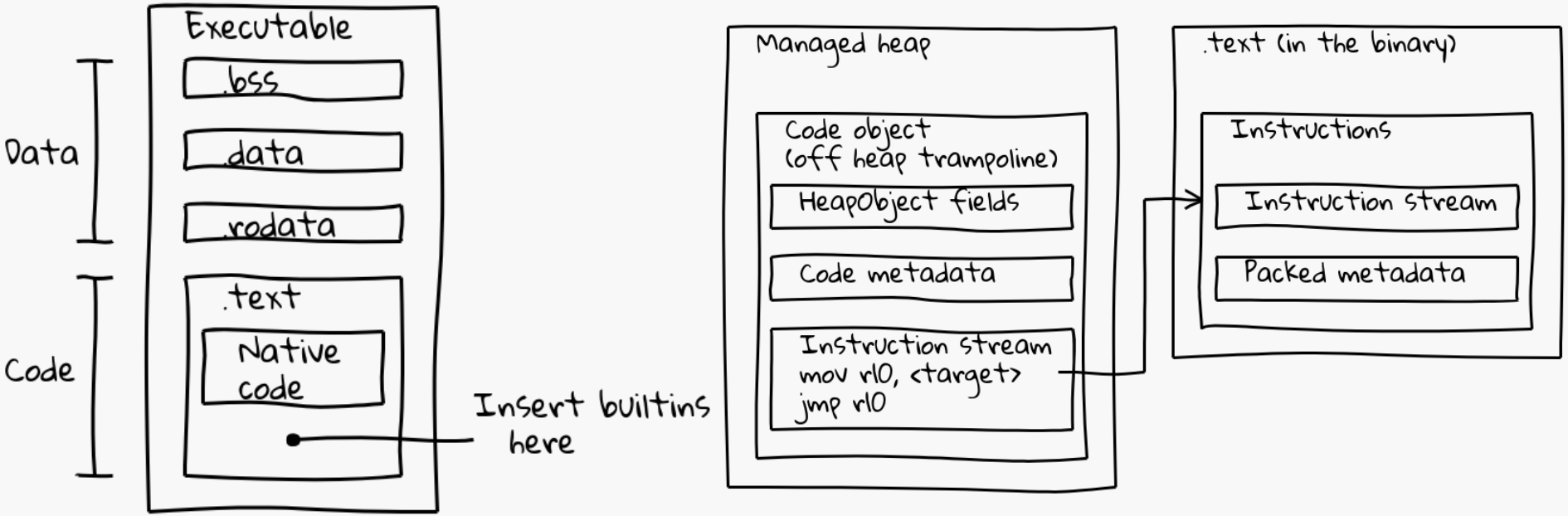
Sections of an executable binary file

https://v8.dev/blog/embedded-builtins

```
// The embedded blob data section starts here.
.section .rodata
.balign 8
v8_Default_embedded_blob_data_data_:
  .octa 0xc790027f000000003bcaf050,0x5a0000000003ae53c50d042f227,0x218000005c00000037800000000
  .octa 0x3a8000007e00000016c00000378,0x17000000ba0000003780000004e4
  .octa 0x19000000d200000016c0000085c,0xe800000ec000000088000009c8
  .octa 0x26000000fc00000008800000a50,0x2ac000124000000315000000ad8
  .octa 0x324000015000000033c00000df0,0x350000018400000034300000112c
  .octa 0x12000001ba0000003c700001470,0xc000001ce00000224000001838
  .octa 0xc000001dc00000116000001a5c,0xc000001ea00000116000001b74,0x2b7c00001f800000011600001c8c
  .octa 0x2b6c00004b0000003e4a00001da4,0x2b8c0000768000003e4a00005bf0
  .octa 0x2b7c0000a22000003e4a00009a3c,0x2b8c0000cda000003e4a0000d888
  .octa 0x2b7c0000f94000003e4a000116d4,0x2bb0000124c000003e4a00015520
  .octa 0x2bb000015080000003e760001936c,0x2bb000017c4000003e760001d1e4
  .octa 0xfd00001a80000003e760002105c,0xd80001b7e000001a8a00024ed4
```

V8 mksnapshot生成的emdedd.S文件

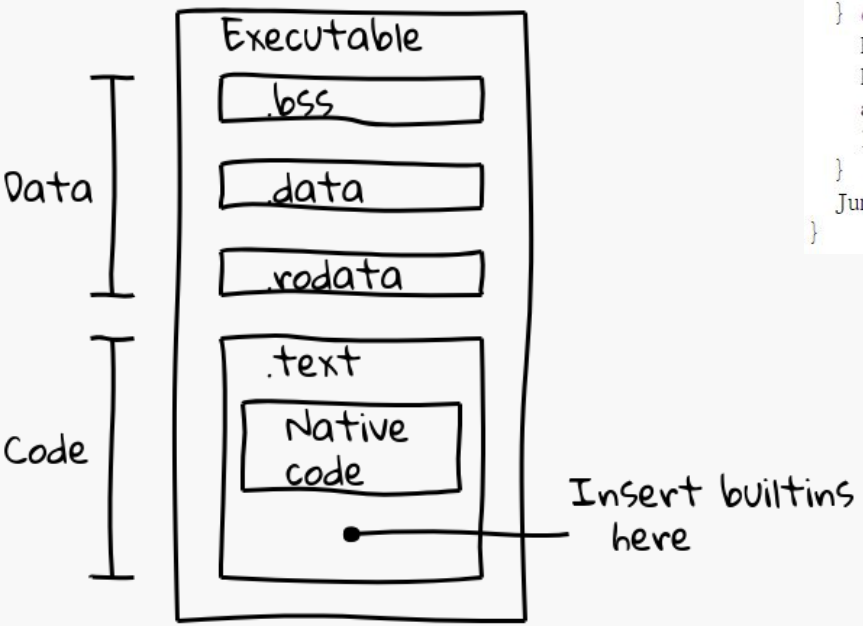
2.Embedded builtins



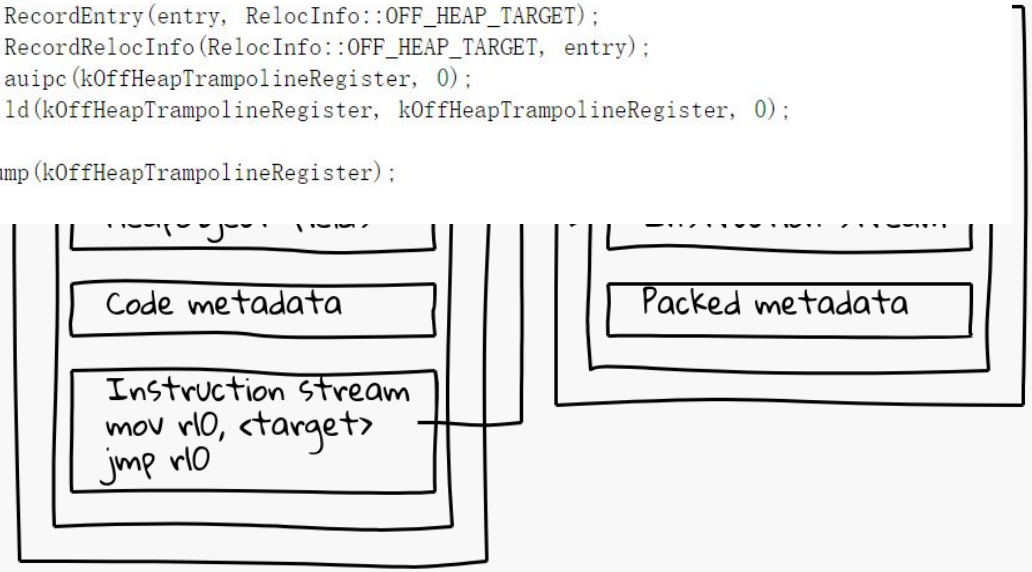
Sections of an executable binary file

https://v8.dev/blog/embedded-builtins

2.Embedded builtins




```
void MacroAssembler::JumpToInstructionStream(Address entry) {
  // Ld a Address from a constant pool.
  // Record a value into constant pool.
  if (!FLAG_riscv_constant_pool) {
    li(kOffHeapTrampolineRegister, Operand(entry, RelocInfo::OFF_HEAP_TARGET));
  } else {
    RecordEntry(entry, RelocInfo::OFF_HEAP_TARGET);
    RecordRelocInfo(RelocInfo::OFF_HEAP_TARGET, entry);
    auipc(kOffHeapTrampolineRegister, 0);
    ld(kOffHeapTrampolineRegister, kOffHeapTrampolineRegister, 0);
  }
  Jump(kOffHeapTrampolineRegister);
}
```



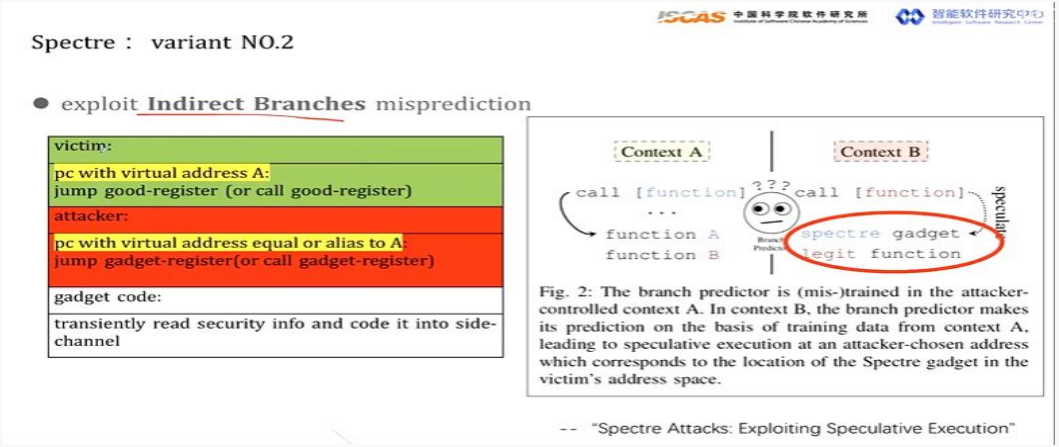
Sections of an executable binary file

https://v8.dev/blog/embedded-builtins

3.Short builtin calls

3.Short builtin calls

Due to Spectre v2 various device/OS combinations have turned off indirect branch prediction. This means that on such configurations we'll get very costly stalls on function calls from JIT code that rely on the CallFunction builtin



Spectre v2 攻击说明
bilibili:BV1hp4y1t7Mx

For 64-bit applications, branch prediction performance can be negatively impacted when the target of a branch is more than 4 GB away from the branch.