LLV8

Динамическая компиляция программ на языке JavaScript в статически типизированное внутреннее представление LLVM

Ваагн Варданян
vaag@ispras.ru

December 2, 2015

JavaScript

- ✔ Один из самых популярных языков веб разработки

JavaScript

- ✔ Один из самых популярных языков веб разработки
- ✔ Динамическое связывание типов

JavaScript

- ✔ Один из самых популярных языков веб разработки
- ✔ Динамическое связывание типов
- ✔ Garbage collected

JITы

- ✔ Если для скриптового языка важна производительность, используется JIT
- ✔ Вопрос **баланса** между временем компиляции и качеством кода стоит особенно остро
- ✔ Для решения проблемы многие современные JIT-компиляторы поддерживают разные уровни компиляции (оптимизации) горячих участков кода

JITы

- ✔ Если для скриптового языка важна производительность, используется JIT
- ✔ Вопрос **баланса** между временем компиляции и качеством кода стоит особенно остро
- ✔ Для решения проблемы многие современные JIT-компиляторы поддерживают разные уровни компиляции (оптимизации) горячих участков кода

JITы

- ✔ Если для скриптового языка важна производительность, используется JIT
- ✔ Вопрос **баланса** между временем компиляции и качеством кода стоит особенно остро
- ✔ Для решения проблемы многие современные JIT-компиляторы поддерживают разные уровни компиляции (оптимизации) горячих участков кода

JIT компиляторы JavaScript

V8 (Google)

- ✔ Node.js, Chrome, Android, Opera, ChromeOS, TizenOS
- ✔ 2 уровня компиляции*

JavaScriptCore (Apple)

- ✔ Safari (Mac OS X и IOS)
- ✔ 4 уровня компиляции (4-й уровень использует LLVM MCJIT)

SpiderMonkey (Mozilla)

- ✔ Firefox, Firefox OS
- ✔ 2 уровня компиляции
- ✔ Модуль OdinMonkey специально для asm.js

JIT компиляторы JavaScript

V8 (Google)

- ✔ Node.js, Chrome, Android, Opera, ChromeOS, TizenOS
- ✔ 2 уровня компиляции*

JavaScriptCore (Apple)

- ✔ Safari (Mac OS X и IOS)
- ✔ 4 уровня компиляции (4-й уровень использует LLVM MCJIT)

SpiderMonkey (Mozilla)

- ✔ Firefox, Firefox OS
- ✔ 2 уровня компиляции
- ✔ Модуль OdinMonkey специально для asm.js

JIT компиляторы JavaScript

V8 (Google)

- ✔ Node.js, Chrome, Android, Opera, ChromeOS, TizenOS
- ✔ 2 уровня компиляции*

JavaScriptCore (Apple)

- ✔ Safari (Mac OS X и IOS)
- ✔ 4 уровня компиляции (4-й уровень использует LLVM MCJIT)

SpiderMonkey (Mozilla)

- ✔ Firefox, Firefox OS
- ✔ 2 уровня компиляции
- ✔ Модуль OdinMonkey специально для asm.js

LLVM MCJIT – популярный тренд

LLVM MCJIT – популярный тренд

- ✔ Pyston (Python, Dropbox)
- ✔ HHVM (Hack&PHP, Facebook)
- ✔ LLILC (MSIL, .NET Foundation)
- ✔ Julia (Julia, Community)
- ✔ JavaScriptCore (JavaScript, Apple) – Fourth tier LLVM [FTL]

Постановка задачи

Добавить в V8 новый уровень оптимизации используя LLVM MCJIT

Схема работы V8

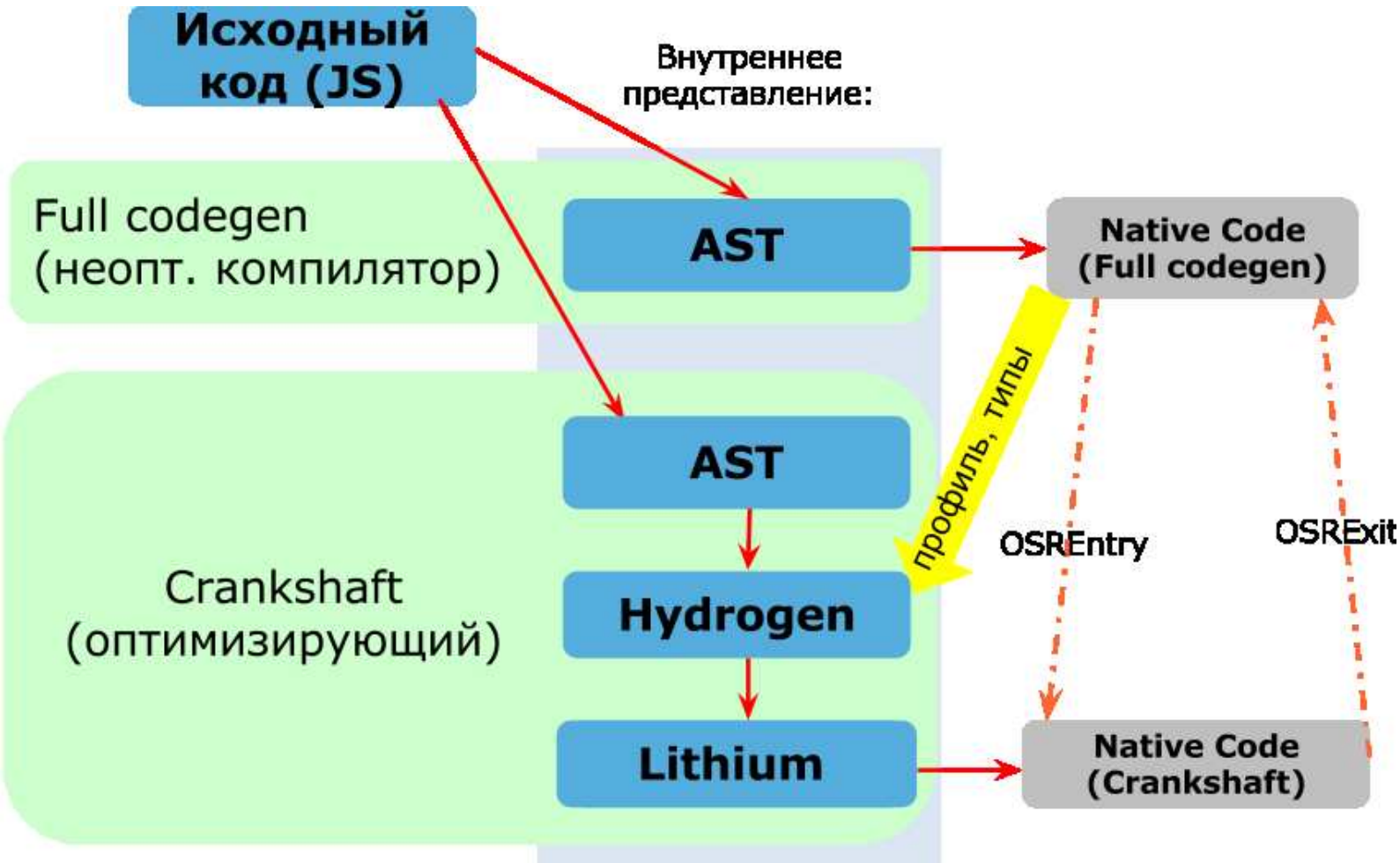
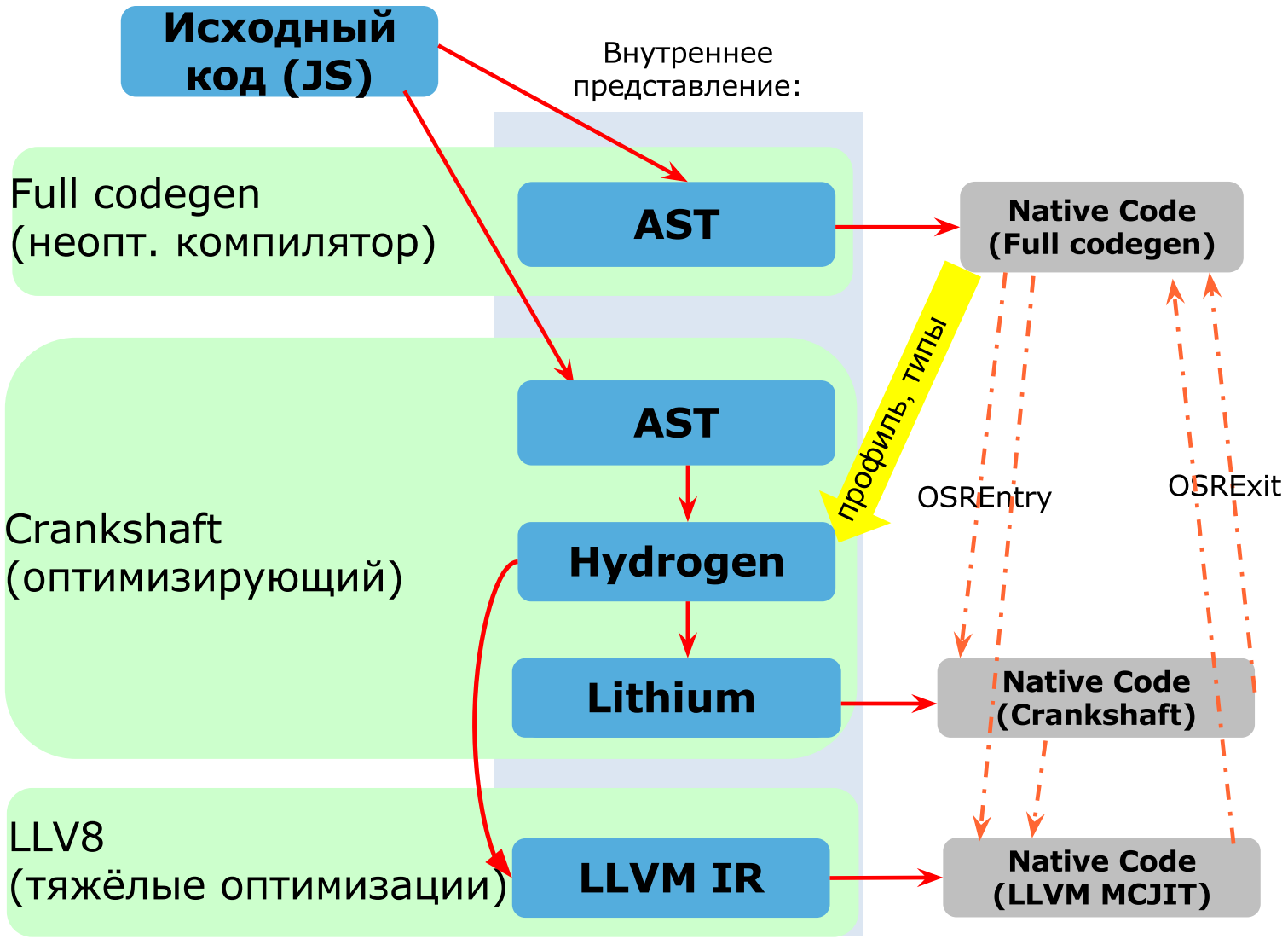


Схема работы LLV8



Спекулятивная компиляция

- ✔ Частичная компиляция кода ⇒ Уменьшение времени выполнения и экономия используемой памяти
- ✔ Предсказание типов переменных
- ✔ Полиморфный встроенный кэш вызовов (polymorphic inline cache)

При неправильной спекуляции - переход на нижний (неоптимизированный) уровень (деоптимизация)

Спекулятивная компиляция

- ✔ Частичная компиляция кода ⇒ Уменьшение времени выполнения и экономия используемой памяти
- ✔ Предсказание типов переменных
- ✔ Полиморфный встроенный кэш вызовов (polymorphic inline cache)

При неправильной спекуляции - переход на нижний (неоптимизированный) уровень (деоптимизация)

Спекулятивная компиляция

- ✔ Частичная компиляция кода ⇒ Уменьшение времени выполнения и экономия используемой памяти
- ✔ Предсказание типов переменных
- ✔ Полиморфный встроенный кэш вызовов (polymorphic inline cache)

При неправильной спекуляции - переход на нижний (неоптимизированный) уровень (деоптимизация)

Деоптимизация в деталях

- ✔ Надо передать значения локальных переменных из уровня Hydrogen к уровню Full-Codegen (стековая машина)
- ✔ Знаем, куда (слоты стека) попадут Hydrogen значения при переходе (через деоптимизацию) на Full code
- ✔ Crankshafted code имеет при себе Translation – отображение (регистры/стек-слоты → стек-слоты). Deoptimizer "перекладывает"
- ✔ Для реализации того же в LLV8 необходимо обладать информацией о том, как будут распределены регистры (отображение llvm::Value → регистры/стек-слоты)

Деоптимизация в деталях

- ✔ Надо передать значения локальных переменных из уровня Hydrogen к уровню Full-Codegen (стековая машина)
- ✔ Знаем, куда (слоты стека) попадут Hydrogen значения при переходе (через деоптимизацию) на Full code
- ✔ Crankshafted code имеет при себе Translation – отображение (регистры/стек-слоты → стек-слоты). Deoptimizer "перекладывает"
- ✔ Для реализации того же в LLV8 необходимо обладать информацией о том, как будут распределены регистры (отображение llvm::Value → регистры/стек-слоты)

Деоптимизация в деталях

- ✔ Надо передать значения локальных переменных из уровня Hydrogen к уровню Full-Codegen (стековая машина)
- ✔ Знаем, куда (слоты стека) попадут Hydrogen значения при переходе (через деоптимизацию) на Full code
- ✔ Crankshafted code имеет при себе Translation – отображение (регистры/стек-слоты → стек-слоты). Deoptimizer "перекладывает"
- ✔ Для реализации того же в LLV8 необходимо обладать информацией о том, как будут распределены регистры (отображение llvm::Value → регистры/стек-слоты)

Деоптимизация в деталях

- ✔ Надо передать значения локальных переменных из уровня Hydrogen к уровню Full-Codegen (стековая машина)
- ✔ Знаем, куда (слоты стека) попадут Hydrogen значения при переходе (через деоптимизацию) на Full code
- ✔ Crankshafted code имеет при себе Translation – отображение (регистры/стек-слоты → стек-слоты). Deoptimizer "перекладывает"
- ✔ Для реализации того же в LLV8 необходимо обладать информацией о том, как будут распределены регистры (отображение llvm::Value → регистры/стек-слоты)

Patchpoint


```
declare void
@llvm.experimental.stackmap(
    i64 <id>, i32 <numShadowBytes>, ...)
```

```
declare void
@llvm.experimental.patchpoint.void(
    i64 <id>, i32 <numBytes>,
    i8* <target>, i32 <numArgs>, ...)

declare i64
@llvm.experimental.patchpoint.i64(
    i64 <id>, i32 <numBytes>,
    i8* <target>, i32 <numArgs>, ...)

; <numBytes> -- reserved, padded with nops
```

StackMaps

Stack Map section[1]

```
...
StkSizeRecord[NumFunctions] {
  uint64 : Function Address
  uint64 : Stack Size
}
StkMapRecord[NumRecords] {
  uint64 : PatchPoint ID
  uint32 : Instruction Offset
  uint16 : NumLocations

  Location[NumLocations] {
    uint8  : Register | Direct | Indirect |
        Constant | ConstantIndex
    uint16 : Dwarf RegNum
  }
}
```

[1] http://llvm.org/docs/StackMaps.html

Relocations

Код – тоже объект в куче. Вызывающий код может переехать.
Вызов кода по известному адресу:

```
mov rax, addr_64
call rax
```

либо

```
call offset_32 ; encoding: 0xe8 offset_32
```

Relocations

Код – тоже объект в куче. Вызывающий код может переехать.

Вызов кода по известному адресу:

```
mov rax, addr_64
call rax
```

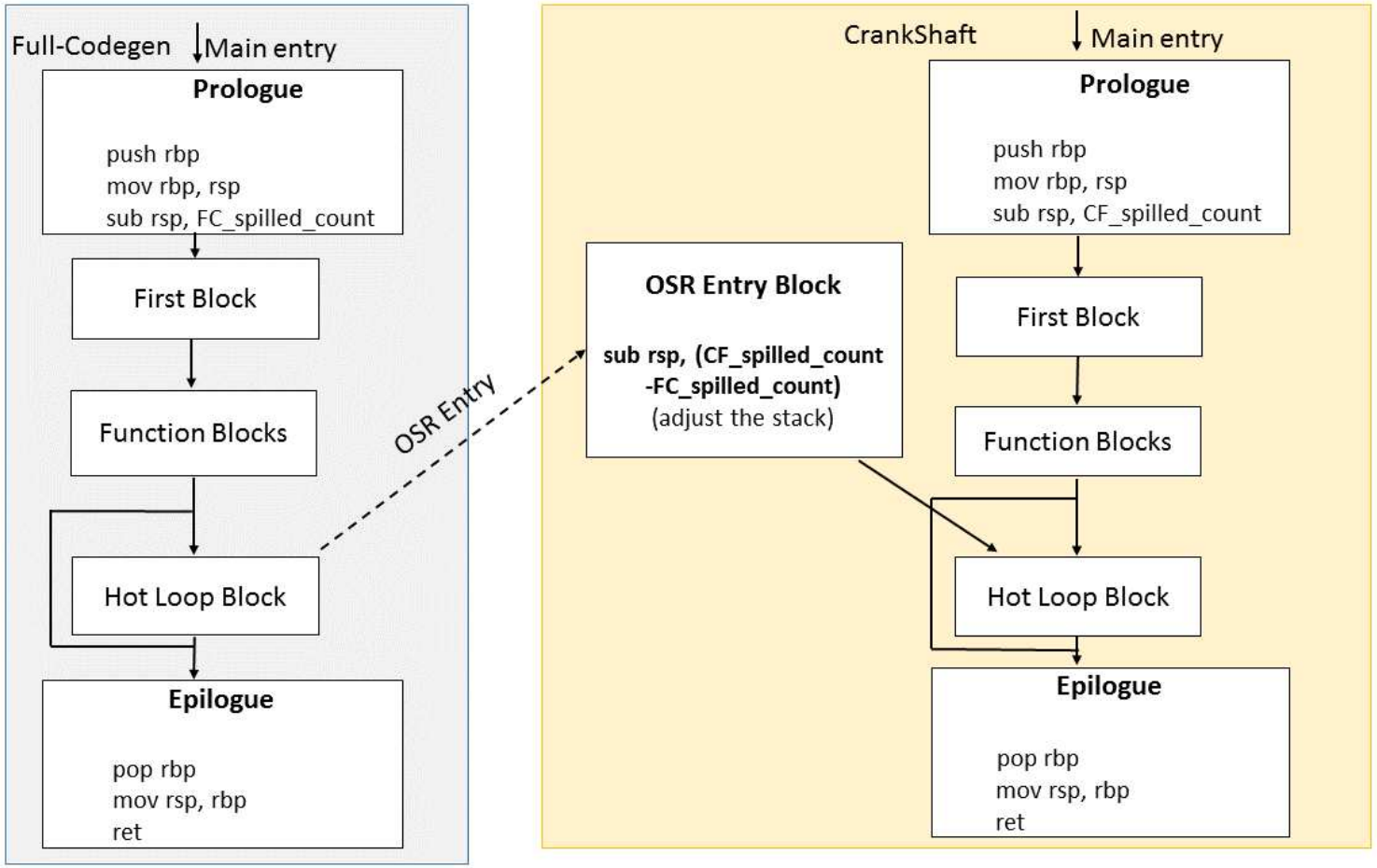
либо

```
call offset_32 ; encoding: 0xe8 offset_32
```

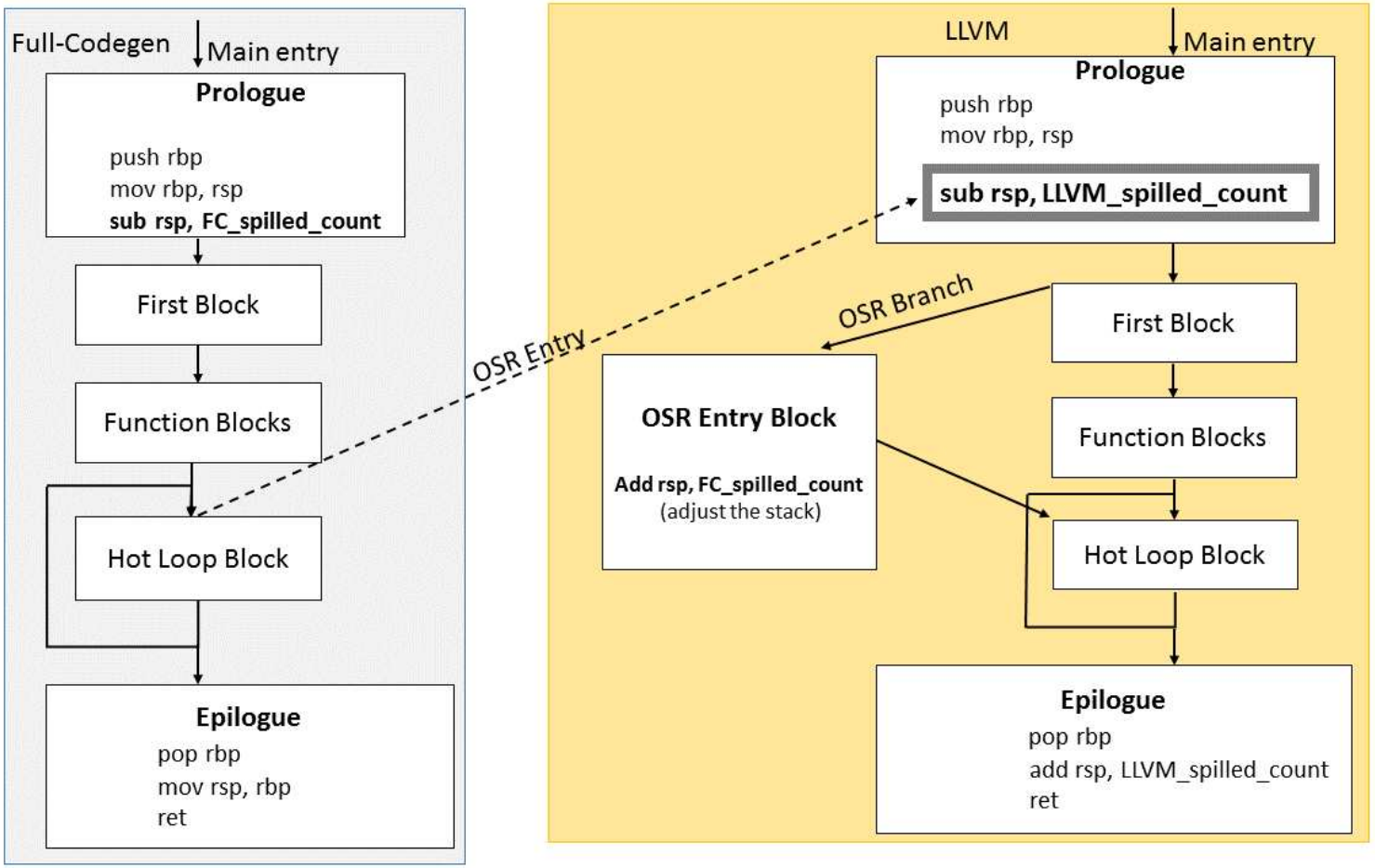
```
llvm.experimental.patchpoint.i64(
    i64 <id>, i32 <numBytes>,
    i8* <target>, i32 <numArgs>, ...)

<id> = id
<numBytes> = 5
<target> = null
<numArgs> = numArgs
; Main purpose -- to get Instruction Offset
```

OSR Entry Crankshaft



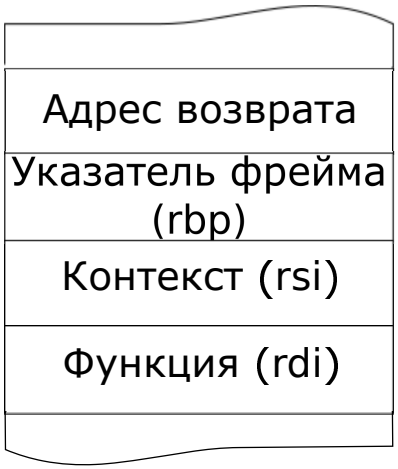
OSR Entry LLVM



Бинарная совместимость

Register pinning

Указатель на массив корневых объектов хранится в R13. Этот регистр был удалён из списков распределяемых регистров.



Обход стека вызовов (например, при GC)
Фрейм стека должен иметь специальный вид, который ожидает V8 и который позволяет переходить к фрейму вызвавшей функции.

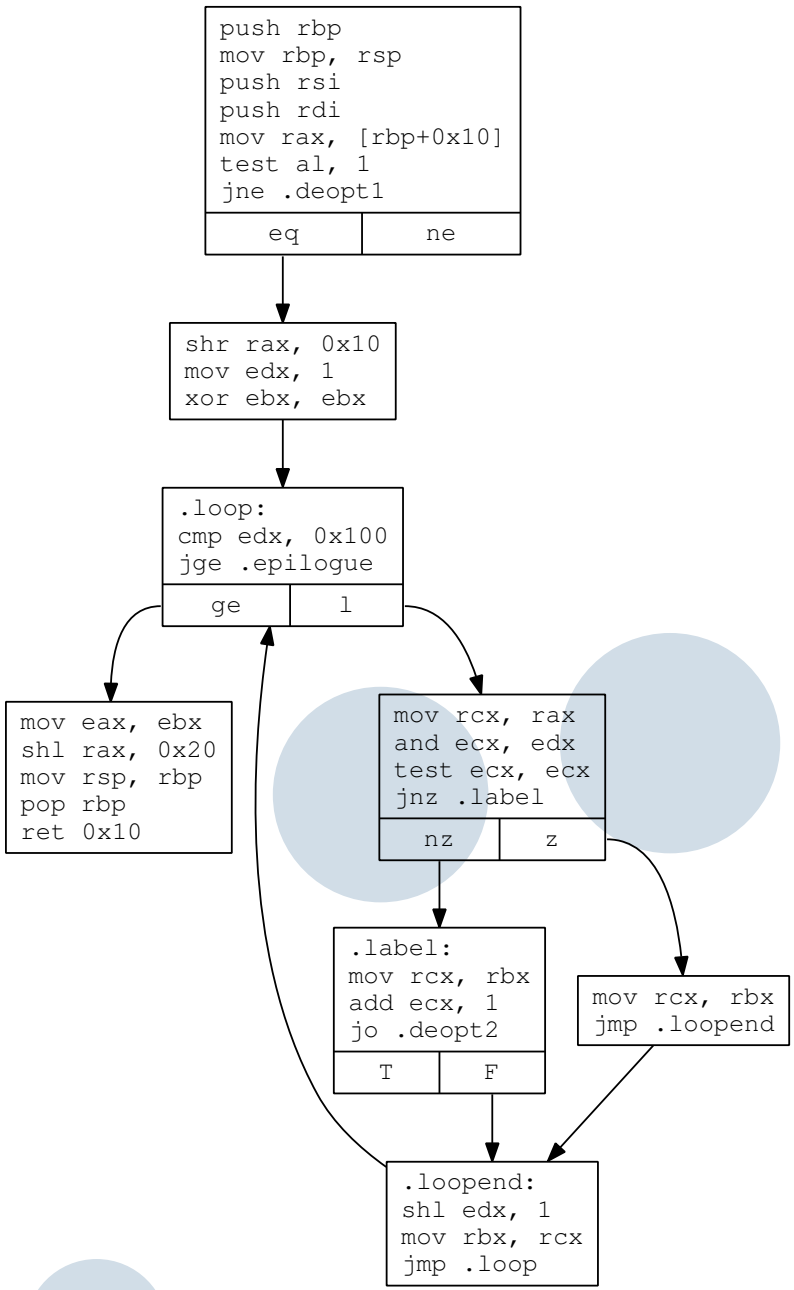
Соглашения о вызовах

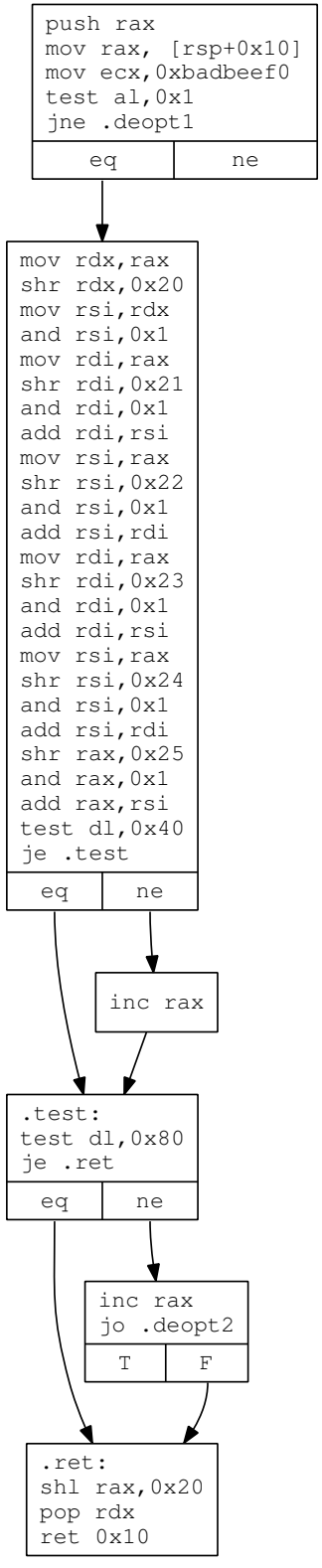
V8 использует собственные соглашения о вызовах, которые не поддерживались LLVM. Поддержка соответствующего соглашения для x86-64 была добавлена в LLVM, что позволило передавать управление на LLVM-сгенерированный код и вызывать из этого кода функции full codegen и Crankshaft.

Тестирование на SunSpider


```
function foo(b) {
    var m = 1, c = 0;
    while(m < 0x100) {
        if(b & m) c++;
        m <<= 1;
    }
    return c;
}
function TimeFunc(func) {
    var sum = 0;
    for(var x = 0; x < ITER; x++)
        for(var y = 0; y < 256; y++) sum += func(y);
    return sum;
}
result = TimeFunc(foo);
```

ITER (число итераций)	3500	35000	350000	3500000
Execution time, Crankshaft, мс	36	216	2050	23312
Execution time, LLV8, мс	27	74	538	5300
Ускорение, число раз	**1.3**	**2.9**	**3.8**	**4.4**


push rbp
mov rbp, rsp
push rsi
push rdi
mov rax, [rbp+0x10]
test al, 1
jne .deopt1
eq
ne
shr rax, 0x10
mov edx, 1
xor ebx, ebx
.loop:
cmp edx, 0x100
jge .epilogue
ge
l
mov eax, ebx
shl rax, 0x20
mov rsp, rbp
pop rbp
ret 0x10
mov rcx, rax
and ecx, edx
test ecx, ecx
jnz .label
nz
z
.label:
mov rcx, rbx
add ecx, 1
jo .deopt2
T
F
mov rcx, rbx
jmp .loopend
.loopend:
shl edx, 1
mov rbx, rcx
jmp .loop


push rax
mov rax, [rsp+0x10]
mov ecx,0xbadbeef0
test al,0x1
jne .deopt1
eq
ne
mov rdx,rax
shr rdx,0x20
mov rsi,rdx
and rsi,0x1
mov rdi,rax
shr rdi,0x21
and rdi,0x1
add rdi,rsi
mov rsi,rax
shr rsi,0x22
and rsi,0x1
add rsi,rdi
mov rdi,rax
shr rdi,0x23
and rdi,0x1
add rdi,rsi
mov rsi,rax
shr rsi,0x24
and rsi,0x1
add rsi,rdi
shr rax,0x25
and rax,0x1
add rax,rsi
test dl,0x40
je .test
eq
ne
inc rax
.test:
test dl,0x80
je .ret
eq
ne
inc rax
jo .deopt2
T
F
.ret:
shl rax,0x20
pop rdx
ret 0x10

Результаты тестирования на SunSpider

Тест	Ориг. кол-во итераций	*10	*100
3d-cube	2.6	2.9	3
3d-raytrace	0.8	0.86	0.9
bitops-bits-in-byte	1.1	1.1	1.3
bitops-nsieve-bit	1	1	1
controlflow-recursive	0.95	0.97	0.97
access-binary-trees	1	1	1
access-nbody	0.8	0.84	0.9
access-nsieve	1	1	1
math-cordic	1.07	1.08	1.1
math-spectral-norm	1.2	1.2	1.3

Текущее состояние

- ✔ Inlining
- ✔ Поддержка OSR Entry
- ✔ Поддержка спекулятивной компиляции (деоптимизация, OSR exit)
- ✔ Поддержка сборщика мусора

Текущее состояние

- ✔ Inlining
- ✔ Поддержка OSR Entry
- ✔ Поддержка спекулятивной компиляции (деоптимизация, OSR exit)
- ✔ Поддержка сборщика мусора

Текущее состояние

- ✔ Inlining
- ✔ Поддержка OSR Entry
- ✔ Поддержка спекулятивной компиляции (деоптимизация, OSR exit)
- ✔ Поддержка сборщика мусора

Текущее состояние

- ✔ Inlining
- ✔ Поддержка OSR Entry
- ✔ Поддержка спекулятивной компиляции (деоптимизация, OSR exit)
- ✔ Поддержка сборщика мусора

Будущая работа

- ✔ Поддержка оставшихся вершин Hydrogen (~ 80/120)
- ✔ Проанализировать применяемые проходы LLVM, выбрать что-то более подходящее, чем -O3
- ✔ Разработка проходов, учитывающих особенности структуры биткода, получаемого из JavaScript. С другой стороны: генерация биткода, лучше поддающегося оптимизациям
- ✔ Векторизация циклов
- ✔ Оптимизации специально для asm.js
- ✔ Open source, привлечь community