**Apache Harmony JIT Summary:**

Directory drlvm\vm\port\src\encoder\ia32\_em64t:

(1)

文件enc\_defs.h

宏定义：

ENCODER\_NAMESPACE\_START 命名空间开始结束，在ENCODER\_ISOLATE宏被定义后有效

ENCODER\_NAMESPACE\_END

COUNTOF(a) : sizeof(a)/sizeof(a[0]) 计算数组元素个数

REG\_STACK 堆栈指针寄存器 EM64T指向RegName\_RSP，IA32指向RegName\_ESP

REG\_MAX 最大的GP(general purpose?)寄存器 EM64T指向RegName\_R15，IA32指向RegName\_EDI

MAX\_REGS 寄存器的数量 EM64T为15，IA32为8

STACK\_SLOT\_SIZE (sizeof(void\*)) A number of bytes 'eaten' by an ordinary PUSH/POP

JMP\_TARGET\_ALIGMENT (16) A recommended by Intel Arch Manual aligment for instructions that

MAX\_NATIVE\_INST\_SIZE (15) A maximum possible size of native instruction.

Enum OpndKind { 操作数类型，和位置相关: memory, immediate or register, it can be used as a bit mask

OpndKind\_Null=0,

OpndKind\_GPReg = 0x01, OpndKind\_MinRegKind = OpndKind\_GPReg,

OpndKind\_SReg = 0x02,

#ifdef \_HAVE\_MMX\_

OpndKind\_MMXReg = 0x03,

#endif

OpndKind\_FPReg = 0x04,

OpndKind\_XMMReg = 0x08,

OpndKind\_OtherReg = 0x10,

OpndKind\_StatusReg = OpndKind\_OtherReg,

OpndKind\_MaxRegKind = OpndKind\_StatusReg, // a max existing kind of register

OpndKind\_MaxReg, // -'- + 1 to be used in array defs

//

OpndKind\_Immediate = 0x20, OpndKind\_Imm=OpndKind\_Immediate,

OpndKind\_Memory = 0x40, OpndKind\_Mem=OpndKind\_Memory,

//

OpndKind\_Reg = 0x1F,

OpndKind\_Any = 0x7F,

// syntetic constants. Normally not used anywhere, but are used for

// human-readable showing under the debugger

OpndKind\_GPReg\_Mem = OpndKind\_GPReg|OpndKind\_Mem,

#ifdef \_HAVE\_MMX\_

OpndKind\_MMXReg\_Mem = OpndKind\_MMXReg|OpndKind\_Mem,

#endif

OpndKind\_XMMReg\_Mem = OpndKind\_XMMReg|OpndKind\_Mem,

}

enum OpndSize { 操作数大小

OpndSize\_Null = 0,

OpndSize\_8 = 0x01,

OpndSize\_16 = 0x02,

OpndSize\_32 = 0x04,

OpndSize\_64 = 0x08,

#if !defined(TESTING\_ENCODER)

OpndSize\_80 = 0x10,

OpndSize\_128 = 0x20,

#endif

OpndSize\_Max,

OpndSize\_Any = 0xFF,

OpndSize\_Default = OpndSize\_Any

}

enum OpndRole { 操作数角色

OpndRole\_Null=0,

OpndRole\_Use=0x1,

OpndRole\_Def=0x2,

OpndRole\_UseDef=OpndRole\_Use|OpndRole\_Def,

OpndRole\_All=0xffff,

}

////

寄存器定义：

31 24 23 16 15 0

|  |  |  |
| --- | --- | --- |
| 操作数类型  OpndKind | 操作数大小  OpndSize | 寄存器索引 |

定义了非常多的寄存器，有EM64T的，有MMX的

////

enum ConditionMnemonic { ：条件指令助记符

ConditionMnemonic\_O=0,

ConditionMnemonic\_NO=1,

ConditionMnemonic\_B=2, ConditionMnemonic\_NAE=ConditionMnemonic\_B, ConditionMnemonic\_C=ConditionMnemonic\_B,

ConditionMnemonic\_NB=3, ConditionMnemonic\_AE=ConditionMnemonic\_NB, ConditionMnemonic\_NC=ConditionMnemonic\_NB,

ConditionMnemonic\_Z=4, ConditionMnemonic\_E=ConditionMnemonic\_Z,

ConditionMnemonic\_NZ=5, ConditionMnemonic\_NE=ConditionMnemonic\_NZ,

ConditionMnemonic\_BE=6, ConditionMnemonic\_NA=ConditionMnemonic\_BE,

ConditionMnemonic\_NBE=7, ConditionMnemonic\_A=ConditionMnemonic\_NBE,

ConditionMnemonic\_S=8,

ConditionMnemonic\_NS=9,

ConditionMnemonic\_P=10, ConditionMnemonic\_PE=ConditionMnemonic\_P,

ConditionMnemonic\_NP=11, ConditionMnemonic\_PO=ConditionMnemonic\_NP,

ConditionMnemonic\_L=12, ConditionMnemonic\_NGE=ConditionMnemonic\_L,

ConditionMnemonic\_NL=13, ConditionMnemonic\_GE=ConditionMnemonic\_NL,

ConditionMnemonic\_LE=14, ConditionMnemonic\_NG=ConditionMnemonic\_LE,

ConditionMnemonic\_NLE=15, ConditionMnemonic\_G=ConditionMnemonic\_NLE,

ConditionMnemonic\_Count=16

};

#define CCM(prefix,cond) Mnemonic\_##prefix##cond=Mnemonic\_##prefix##cc+ConditionMnemonic\_##cond

enum Mnemonic { ： 指令助记符，定义了很多的指令

Mnemonic\_NULL=0, Mnemonic\_Null=Mnemonic\_NULL,

Mnemonic\_ADC, // Add with Carry

Mnemonic\_ADD, // Add

Mnemonic\_ADDSD, // Add Scalar Double-Precision Floating-Point Values

Mnemonic\_ADDSS, // Add Scalar Single-Precision Floating-Point Values

Mnemonic\_AND, // Logical AND

...

}

///指令前缀，分四组

enum InstPrefix {

InstPrefix\_Null = 0,

// Group 1

InstPrefix\_LOCK = 0xF0,

InstPrefix\_REPNE = 0xF2,

InstPrefix\_REPNZ = InstPrefix\_REPNE,

InstPrefix\_REP = 0xF3, InstPrefix\_REPZ = InstPrefix\_REP,

// Group 2

InstPrefix\_CS = 0x2E,

InstPrefix\_SS = 0x36,

InstPrefix\_DS = 0x3E,

InstPrefix\_ES = 0x26,

InstPrefix\_FS = 0x64,

InstPrefix\_GS = 0x65,

//

InstPrefix\_HintTaken = 0x3E,

InstPrefix\_HintNotTaken = 0x2E,

// Group 3

InstPrefix\_OpndSize = 0x66,

// Group 4

InstPrefix\_AddrSize = 0x67

};

///////////////

inline unsigned getSizeBytes(OpndSize sz) ：返回操作数字节数，OpndSize\_64,32,16,8分别返回8，4，2，1

inline bool isRegKind(OpndKind kind)：检查操作数是否是寄存器操作数，在OpndKind\_GPReg和OpndKind\_MaxRegKind之间的认为是RegKind

RegName getRegName(const char \* regname)：将字符串描述的寄存器名称转成实际寄存器的值(数字，地址?)

inline RegName getRegName(OpndKind k, OpndSize s, int idx)：根据Opnd的kind, size和index，获取实际的寄存器的值(数字，地址?)

inline unsigned getRegMask(RegName reg)： 获取mask，1<<(reg&0xff)

inline OpndKind getRegKind(RegName reg)：获取类型(OpndKind)(reg>>24)

inline OpndSize getRegSize(RegName reg)：获取大小(OpndSize)((reg>>16)&0xFF)

inline unsigned char getRegIndex(RegName reg)：获取索引(unsigned char)(reg&0xFF)

const char \* getRegNameString(RegName reg)：获取字符串表示的寄存器名称

const char \* getOpndSizeString(OpndSize size)：获取字符串表示的寄存器大小

OpndSize getOpndSize(const char \* sizeString)：将字符串表示的寄存器大小转成实际大小

const char \* getOpndKindString(OpndKind kind)：获取字符串表示的寄存器类型

OpndKind getOpndKind(const char \* kindString)：将字符串表示的寄存器类型转成实际类型

const char \* getConditionString(ConditionMnemonic cm)：获取字符串表示的条件助记符

inline RegName getAliasReg(RegName reg, OpndSize sz)

inline bool equals(RegName r0, RegName r1)：Tests two RegName-s of the same kind for equality，Kind和Index都相等，则认为相等

///////////////////

(2)

文件enc\_prvt.h

Contains some definitions/constants and other stuff used by the Encoder internally.

enum OpcodeByteKind {

OpcodeByteKind\_ZeroOpcodeByte = 0x0100,

//

// The names \_SlashR, \_SlahsNum, \_ib, \_iw, etc

// represent the appropriate abbreviations used

// in the mnemonic descriptions in the Intel's arch manual.

//

OpcodeByteKind\_SlashR = 0x0200,

OpcodeByteKind\_SlashNum = 0x0300,

OpcodeByteKind\_ib = 0x0400,

OpcodeByteKind\_iw = 0x0500,

OpcodeByteKind\_id = 0x0600,

...

};

struct Rex { Represents the REX part of instruction.

unsigned char b : 1;

unsigned char x : 1;

unsigned char r : 1;

unsigned char w : 1;

unsigned char dummy : 4; // must be '0100'b

unsigned int :24;

};

struct SIB { Describes SIB (scale,index,base) byte.

unsigned char base:3;

unsigned char index:3;

unsigned char scale:2;

unsigned int padding:24;

};

struct ModRM { Describes ModRM byte.

unsigned char rm:3;

unsigned char reg:3;

unsigned char mod:2;

unsigned int padding:24;

};

//OpcodeInfo结构，**exactly the same as EncoderBase::OpcodeDesc, but also holds info about platform on which the opcode is applicable. (?)**

struct OpcodeInfo {

enum platform {

/// an opcode is valid on all platforms

all,

// opcode is valid on IA-32 only

em64t,

// opcode is valid on Intel64 only

ia32,

// opcode is added for the sake of disassembling, should not be used in encoding

decoder,

// only appears in master table, replaced with 'decoder' in hashed version

decoder32,

// only appears in master table, replaced with 'decoder' in hashed version

decoder64,

};

platform platf;

unsigned opcode[4+1+1];

EncoderBase::OpndDesc opnds[3];

{

OpndKind kind;

OpndSize size;

RegName reg;

}

EncoderBase::OpndRolesDesc roles;

{

unsigned count;

unsigned defCount;

unsigned useCount;

unsigned roles;

}

};

宏助记符标志：

#define MF\_NONE (0x00000000) Operation has no special properties.

#define MF\_AFFECTS\_FLAGS (0x00000001) Operation affects flags

#define MF\_USES\_FLAGS (0x00000002) Operation uses flags - conditional operations, ADC/SBB/ETC

#define MF\_CONDITIONAL (0x00000004) Operation is conditional - MOVcc/SETcc/Jcc/ETC

#define MF\_SYMMETRIC (0x00000008) Operation is symmetric - its args can be swapped (ADD/MUL/etc).

#define MF\_SAME\_ARG\_NO\_USE (0x00000010) Operation is XOR-like - XOR, SUB - operations of 'arg,arg' is pure def, without use

// MnemonicInfo结构，指令结构

struct MnemonicInfo {

Mnemonic mn; 助记符

unsigned flags; 助记符标志MF\_\*

EncoderBase::OpndRolesDesc roles; 助记符角色描述

const char \* name; 助记符字符串表示

/\*\*

\* Array of opcodes.

\* The terminating opcode description always have OpcodeByteKind\_LAST

\* at the opcodes[i].opcode[0].

\* The size of '25' has nothing behind it, just counted the max

\* number of opcodes currently used (MOV instruction).

\*/

OpcodeInfo opcodes[25];

{

platform platf;

unsigned opcode[4+1+1];

EncoderBase::OpndDesc opnds[3];

{

OpndKind kind;

OpndSize size;

RegName reg;

}

EncoderBase::OpndRolesDesc roles;

{

unsigned count;

unsigned defCount;

unsigned useCount;

unsigned roles;

}

}

};

////////////////////

(3)文件enc\_tabl.cpp

定义指令表：

MnemonicInfo masterEncodingTable[] = {

...

}

每个指令的结构：

{

mn 指令助记符，Mnemonic\_\*\*

flags 标志

roles 角色描述

name 字符串表示

opcodes[] opcodes数组

}

int EncoderBase::buildTable(void)

建立指令表，根据定义的数组，按mn进行排序，建立HashMap，建立opcodes，建立Mnemonic Desc

void EncoderBase::buildMnemonicDesc(const MnemonicInfo \* minfo)

建立某个指令信息

/////////////////////

(4)文件enc\_base.h

class EncoderBase{

//结构OpndDesc，主要供结构OpcodeDesc使用

struct OpndDesc {

OpndKind kind;

OpndSize size;

RegName reg;

};

//结构OpndRolesDesc，主要供结构OpcodeDesc使用

struct OpndRolesDesc {

unsigned count;

unsigned defCount;

unsigned useCount;

unsigned roles;

};

//结构OpcodeDesc

union OpcodeDesc {

char dummy[128]; // To make total size a power of 2

struct {

char opcode[5];

unsigned opcode\_len;

unsigned aux0;

unsigned aux1;

OpndDesc opnds[3];

unsigned first\_opnd;

OpndRolesDesc roles;

char last;

char platf;

};

}

static const unsigned int MAX\_OPCODES = 32;

static OpcodeDesc opcodes[Mnemonic\_Count][MAX\_OPCODES]; Array of available opcodes. Mnemonic\_Count在enum Mnemonic中定义

struct MnemonicDesc {

Mnemonic mn;

unsigned flags;

OpndRolesDesc roles;

const char \* name;

};

static MnemonicDesc mnemonics[Mnemonic\_Count]; Array of mnemonics.

static const unsigned char size\_hash[OpndSize\_64+1]; A table used for the fast computation of hash value.

static const unsigned char kind\_hash[OpndKind\_Mem+1]; A table used for the fast computation of hash value.

static unsigned char opcodesHashMap[Mnemonic\_Count][HASH\_MAX]; Mapping between operands hash code and operands.

static char \* encode(char \* stream, Mnemonic mn, const Operands& opnds); 将指令和操作数编码成真正的二进制指令

static char \* nops(char \* stream, unsigned howMany); Generates the smallest possible number of NOP-s.

static char \* prefix(char\* stream, InstPrefix pref); Inserts a prefix into the code buffer.

static const MnemonicDesc \* getMnemonicDesc(Mnemonic mn) Returns #MnemonicDesc by the given Mnemonic.

static Mnemonic str2mnemonic(const char \* mn\_name); Returns a Mnemonic for the given name.

static const char \* getMnemonicString(Mnemonic mn) Returns a string representation of the given Mnemonic.

static const char \* toStr(Mnemonic mn)

static OpndRole getOpndRoles(OpndRolesDesc ord, unsigned idx) Extracts appropriate OpndRole for a given operand.

OpcodeDesc \* lookup(Mnemonic mn, const Operands& opnds) 根据助记符mn从opcodes数组查找OpcodeDesc，先利用哈希表查找；找不到查完全匹配的；再

找不到，则查近似的

static char\* encodeModRM(char\* stream, const Operands& opnds, Encodes mod/rm byte.

unsigned idx, const OpcodeDesc \* odesc, Rex \* prex);

static char\* encode\_aux(char\* stream, unsigned aux, Encodes special things of opcode description - '/r', 'ib', etc.

const Operands& opnds, const OpcodeDesc \* odesc,

unsigned \* pargsCount, Rex\* prex);

static unsigned char getHWRegIndex(const RegName reg)

static bool is\_em64t\_extra\_reg(const RegName reg)

static int buildTable(void); implemented in enc\_tabl.cpp

static void buildMnemonicDesc(const MnemonicInfo \* minfo); implemented in enc\_tabl.cpp

static unsigned short getHash(const OpcodeInfo\* odesc); Computes hash value for the given operands.

class Operand{...}

class Operands{...}

}

///////////////////

(5)Encoder.h, Encoder.cpp, Encoder.inc

定义了非常多的编码函数，调用EncoderBase::encode进行指令编码，目前发现在vmcore\src\lil\ia32\lil\_code\_generator\_ia32.cpp和vmcore\src\lil\em64t\lil\_code\_generator\_em64t.cpp中有调用。

/////////////////////

drlvm\vm\jitrino\src\vm\JITInterface.cpp:

JIT\_init

JIT\_deinit

JIT\_next\_command\_line\_argument

JIT\_set\_profile\_access\_interface

JIT\_enable\_profiling

JIT\_profile\_notification\_callback

JIT\_recompiled\_method\_callback

JIT\_compile\_method\_with\_params：

JIT\_Result JIT\_compile\_method\_with\_params(JIT\_Handle jit, Compile\_Handle compilation, Method\_Handle method\_handle, OpenMethodExecutionParams compilation\_params)

Vm里没有： JIT\_Handle, Compile\_Handle, Method\_Handle

用到的类：

MemoryManager

JITInstanceContext

CompilationInterface

CompilationContext

最终调用：

result = Jitrino::CompileMethod(&cs) ? JIT\_SUCCESS : JIT\_FAILURE;

进行编译

drlvm\vm\jitrino\src\main\Jitrino.h

class Jitrino:

static bool Init(JIT\_Handle jit, const char\* name);

static void DeInit(JIT\_Handle jit);

static bool CompileMethod(CompilationContext\* compilationContext);

ComplicationConext有Pipeline, Pipeline是一条一条的，逐条编译？转成SessionAction sa ,

CompilationContext c

sa->setCompilationContext(c);

sa->start(), sa->run(), sa->stop()

用到CompilationInterface, MethodDesc

CompilationContext::getVMCompilationInterface获取CompilationInterface

CompilationInterface::getMethodToCompile获取MethodDesc，方法描述

然后使用Pipeline, SessionAction进行编译

static RuntimeInterface\* getRuntimeInterface() {return runtimeInterface;}

static MemoryManager& getGlobalMM() { return \*global\_mm; }

SessionAction:

从IAction继承下来，被

HIR2LIRSelectorSessionAction

CodeGenerator

IRBuilder

OptInitSession, OptPass

TranslatorSession等继承

ComplicationConext里面的SessionAction的run最终应该调用CodeGenerator的genCode，最后调用

CodeGenerator的run代码。

CodeGenerator有两个地方：src\codegenerator\CodeGenIntfc.h和src\codegenerator\ia32\Ia32CodeGenerator.cpp

///////////////////////////////

CodeSelector

drlvm/vm/jitrino/src/optimizer/CodeSelectors.h, .cpp

涉及到四个selector：\_VarCodeSelector, \_BlockCodeSelector, \_CFGCodeSelector, \_MethodCodeSelector

先是在drlvm/vm/jitrino/src/codegenrator/CodeGenIntfc.h里面定义了基础类

VarCodeSelector, BlockCodeSelector, CFGCodeSelector, MethodCodeSelector

VarCodeSelector:

genCode(Callback&), 内部的class Callback包含函数：defVar和setManagedPointerBase

需要了解的概念：

VarOpnd, varMap(U32指针), GCBasePointerMap

OpndBase

//////////////////////////////////////////////////////////////

**Android JIT Summary**

Global Structures:

**EncoderBase::OpndDesc**{ //操作数描述

OpndKind kind;

OpndSize size;

OpndExt ext;//Extention of the operand

RegName reg;//Appropriate RegName if operand must reside on a particular register (i.e. CWD/CDQ instructions), RegName\_Null otherwise.

}

**EncoderBase::OpndRolesDesc**{ //操作数角色描述

unsigned count;//operands number in the operation

unsigned defCount;//defs number

unsigned useCount;//uses number

roles;//Operands roles, bit-packed

}

**EncoderBase::OpcodeDesc**{ //操作码描述，包含了操作数描述和操作数角色描述

char dummy[128];

char opcode[5];//Raw opcode bytes

unsigned opcode\_len;

unsigned aux0, aux1;

OpndDesc opnds[3];//Info about opcode's operands

unsigned first\_opnd;

OpndRolesDesc roles;//Info about operands

char last;

char platf;

}

Operand Class and Operands Class

**class EncoderBase::Operand**{ //操作数类，包含了操作数描述及更加详细的信息，已经相关的操作函数

//Contains info about a single instructions's operand - its location, size and a value for immediate or RegName for register operands.

Member variables:

OpndKind m\_kind;

OpndSize m\_size;

OpndExt m\_ext;

RegName m\_base;

RegName m\_index;

unsigned m\_scale;

union{

int m\_disp;

RegName m\_reg;

unsigned m\_scale;

};

unsigne m\_hash;

bool m\_need\_rex;

And define many related operation for them

}

**class EncoderBase::Operands{}**

**Inst**{//指令结构

Mnemonic mn; //Mnemonic of the instruction.s，助记符，枚举类型

unsigned int prefc; //Number of prefixes (1 byte each).

InstPrefix pref[4]; //Instruction prefixes，枚举类型

unsigned size; //Size, in bytes, of the instruction.

Unsigned flags; //Flags of the instruction.

unsigned argc; //Number of arguments of the instruction.

EncoderBase::Operand operands[3];//最多三个操作数

EncoderBase::OpcodeDesc\* odesc;

}

//////////////

Based on encoder/ia32\_em64t of apache harmony, libenc add:

enc\_defs\_ext.h

enc\_wrapper.h, enc\_wrapper.cpp

**enc\_wrapper.h:**

Enumerations，定义了很多枚举类型

PhysicalReg

Reg\_No

Opnd\_Size

ALU\_Opcode

ConditionCode

InstrPrefix

**LowOpndRegType ？**

Export many encode/decode interface for upper layer:

char\* encoder\_imm(Mnemonic m, OpndSize size, int imm, char\* stream);

unsigned encoder\_get\_inst\_size(char \* stream);

char\* encoder\_update\_imm(int imm, char \* stream);

char\* encoder\_mem(Mnemonic m, OpndSize size,

int disp, int base\_reg, bool isBasePhysical, char\* stream);

char\* encoder\_reg(Mnemonic m, OpndSize size, int reg, bool isPhysical, LowOpndRegType type, char\* stream);

char\* encoder\_reg\_reg(Mnemonic m, OpndSize size, int reg, bool isPhysical, int reg2, bool isPhysical2, LowOpndRegType type, char\* stream);

char\* encoder\_mem\_reg(Mnemonic m, OpndSize size, int disp, int base\_reg,

bool isBasePhysical, int reg, bool isPhysical, LowOpndRegType type, char\* stream);

char\* encoder\_mem\_scale\_reg(Mnemonic m, OpndSize size, int base\_reg, bool isBasePhysical, int index\_reg, bool isIndexPhysical, int scale,

int reg, bool isPhysical, LowOpndRegType type, char\* stream);

char\* encoder\_reg\_mem\_scale(Mnemonic m, OpndSize size, int reg, bool isPhysical,

int base\_reg, bool isBasePhysical, int index\_reg, bool isIndexPhysical, int scale,

LowOpndRegType type, char\* stream);

char \* encoder\_mem\_disp\_scale\_reg(Mnemonic m, OpndSize size,

int base\_reg, bool isBasePhysical, int disp, int index\_reg, bool isIndexPhysical, int scale,

int reg, bool isPhysical, LowOpndRegType type, char \* stream);

char \* encoder\_movzs\_mem\_disp\_scale\_reg(Mnemonic m, OpndSize size,

int base\_reg, bool isBasePhysical, int disp, int index\_reg, bool isIndexPhysical, int scale,

int reg, bool isPhysical, LowOpndRegType type, char \* stream);

char\* encoder\_reg\_mem\_disp\_scale(Mnemonic m, OpndSize size,

int reg, bool isPhysical,

int base\_reg, bool isBasePhysical, int disp, int index\_reg, bool isIndexPhysical, int scale,

LowOpndRegType type, char\* stream);

char\* encoder\_reg\_mem(Mnemonic m, OpndSize size,

int reg, bool isPhysical,

int disp, int base\_reg, bool isBasePhysical, LowOpndRegType type, char\* stream);

char\* encoder\_imm\_reg(Mnemonic m, OpndSize size,

int imm, int reg, bool isPhysical, LowOpndRegType type, char\* stream);

char \* encoder\_update\_imm\_rm(int imm, char \* stream);

char\* encoder\_imm\_mem(Mnemonic m, OpndSize size,

int imm,

int disp, int base\_reg, bool isBasePhysical, char\* stream);

char\* encoder\_fp\_mem(Mnemonic m, OpndSize size, int reg,

int disp, int base\_reg, bool isBasePhysical, char\* stream);

char\* encoder\_mem\_fp(Mnemonic m, OpndSize size,

int disp, int base\_reg, bool isBasePhysical,

int reg, char\* stream);

char\* encoder\_return(char\* stream);

char\* encoder\_compare\_fp\_stack(bool pop, int reg, bool isDouble, char\* stream);

char\* encoder\_movez\_mem\_to\_reg(OpndSize size,

int disp, int base\_reg, bool isBasePhysical,

int reg, bool isPhysical, char\* stream);

char\* encoder\_moves\_mem\_to\_reg(OpndSize size,

int disp, int base\_reg, bool isBasePhysical,

int reg, bool isPhysical, char\* stream);

char \* encoder\_movez\_reg\_to\_reg(OpndSize size,

int reg, bool isPhysical, int reg2,

bool isPhysical2, LowOpndRegType type, char \* stream);

char \* encoder\_moves\_reg\_to\_reg(OpndSize size,

int reg, bool isPhysical, int reg2,

bool isPhysical2, LowOpndRegType type, char \* stream);

int decodeThenPrint(char\* stream\_start);

char\* decoder\_disassemble\_instr(char\* stream, char\* strbuf, unsigned int len);

///////////////

compiler/codegen/x86 files references:

|  |  |
| --- | --- |
| AnalysisO1.h | 定义巨多的基础结构，数据类型，函数接口；供本层目录其它文件调用。是重要的接口函数。 |
| AnalysisO1.cpp | This file implements register allocator, constant folding. |
| BytecodeVisitor.cpp | This file implements visitors of the bytecode |
| CodegenInterface.cpp |  |
| Lower.cpp | This file implements the high-level wrapper for lowering |
| Lower.h | A header file to define interface between lowering and register allocator. |
| LowerAlu.cpp | This file lowers ALU bytecodes. |
| LowerConst.cpp | This file lowers the following bytecodes: CONST\_XXX. |
| LowerGetPut.cpp | This file lowers the following bytecodes: XGET|PUT\_XXX |
| LowerInvoke.cpp | This file lowers the following bytecodes: INVOKE\_XXX |
| LowerJump.cpp | This file lowers the following bytecodes: IF\_XXX, GOTO |
| LowerMove.cpp | This file lowers the following bytecodes: MOVE\_XXX |
| LowerObject.cpp | This file lowers the following bytecodes: CHECK\_CAST, . |
| LowerReturn.cpp | This file lowers the following bytecodes: RETURN |
| NcgAot.cpp |  |
| NcgAot.h |  |
| NcgHelper.cpp |  |
| NcgHelper.h |  |
| Translator.h |  |

//////////////////

AnalysisO1.h

concept: basic block, register, method, byte code, edge, GG GPR VR,

def, use, virtual reg, const VR, live range, bound, bound check, memory VR, TempReg,

compile table entry, regAllocStateEntry, edge,

Basic Block has edges

Method has basic blocks

byte code has registers

Enumerations，定义了很多枚举类型

|  |  |
| --- | --- |
| OverlapCase | various cases of overlapping between 2 variables |
| RegAccessType | access type of a variable |
| GlobalType | a variable can be local (L), globally local (GL) or global (GG) |
| VRState |  |
| VRDelayFreeFlags | helper state to determine if freeing VRs needs to be delayed |
| TRState | state of temporary registers |
| XferType |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

一些结构

//!information about a physical register，寄存器信息

RegisterInfo {

PhysicalReg physicalReg;

bool isUsed;

bool isCalleeSaved;

int freeTimeStamp;

}

UniqueRegister {

LowOpndRegType physicalType;

int regNum;

int numExposedUsage;

PhysicalReg physicalReg;

}

RegAllocConstraint {

PhysicalReg physicalReg; //枚举类型

int count;

}

XferPoint

DefOrUse

for def: accessType means which part of the VR defined at offestPC is live now

for use: accessType means which part of the usage comes from the reachingDef

DefOrUseLink a link list of DefOrUse

DefUsePair pair of def and uses

VirtualRegInfo

information associated with a virtual register

the pair <regNum, physicalType> uniquely determines a variable

ConstVRInfo information of whether a VR is constant and its value

LiveRange specifies one live range

//////////

AnalysisO1.cpp, .h 寄存器分配算法和常量折叠算法

三种变量类型

virtual register虚拟寄存器, type & LowOpndRegType\_virtual

temporary不是虚拟寄存器，且regNum < PhysicalReg\_GLUE\_DVMDEX

glue variables regNum>=PhysicalReg\_GLUE\_DVMDEX

两种变量之间的覆盖关系

比较复杂，有待进一步研究。

////////////////////////////

Dalvik JIT Lower:

The framework graph:

**lowerByteCode**

Lower.h, Lower.cpp

**Lower Helper**

lower\_imm/reg/mem/reg\_mem, ...

lower IA32 native instructions:

move, store, compare, push, ...

**Upper Lower Interfaces**

LowerAlu, LowerConst, LowerInvoke, LowerJump, Move, Return, Object, etc.

**IA32\_EM64 lib**

EncoderBase::encode

DecoderBase::decode

enc\_base.h, dec\_base.h, .cpp

**Register Allocation**

encoder\_imm/mem/reg, ...

enc\_wrapper.h, .cpp, enc\_defs\_ext.h

**Encoder Wrapper**

encoder\_imm/mem/reg, ...

enc\_wrapper.h, .cpp, enc\_defs\_ext.h

//////////

frame info:

|  |  |
| --- | --- |
| JeffInterpFrame \*frame; |  |
| register uint8 \*frame\_ip; | cache of frame->ip, Instruction pointer of the bytecode array  **frame\_ip = code\_block->bytecode** |
| register uint32 \*frame\_lp; | cache of frame->lp, Points to local variable array of the current frame  当前帧大小：JeffInterpFrame结构 + (max\_locals + max\_stack) \* 5  JeffInterpFrame结构后面是local variable array:  **frame\_lp = (uint32 \*)((uint8 \*)frame + sizeof (\*frame))**  **大小：code\_block->max\_locals \* 4** |
| register uint32 \*frame\_sp; | cache of frame->sp, Operand stack top pointer of the current frame.  **frame\_sp = frame\_lp + code\_block->max\_locals;**  **大小：code\_block->max\_stack \* 4** |
| register uint8 \*frame\_ref; | cache of frame->ref, An array recording whether the corresponding local variable or stack slot is a reference of an object, i.e. ref[i] is non-zero indicates lp[i] is a reference.  ref[i]标记lp[i]是不是一个对象引用  **frame\_ref = (uint8 \*)(frame\_sp + code\_block->max\_stack);**  **大小：code\_block->max\_locals + code\_block->max\_stack** |

|  |  |  |  |
| --- | --- | --- | --- |
| OPCODE | Opnds | Operation Detail |  |
|  |  |  |  |
| jeff\_nop | none |  | do nothing |
| jeff\_aconst\_null | none | PUSH\_R(NULL):  \*FRAME\_REF (frame\_sp) = 1;  \*(JeffObjectRef\*)frame\_sp++= (JeffObjectRef)(NULL); | push a NULL object to stack and set ref flag |
| jeff\_iconst\_m1 |  | PUSH\_I(-1):  \*(int32\*)frame\_sp++=(int32)(-1); | push -1 to stack |
| jeff\_iconst\_0 |  | PUSH\_I(0):  \*(int32\*)frame\_sp++=(int32)(0); | push 0 to stack |
| jeff\_iconst\_1 |  | PUSH\_I(1):  \*(int32\*)frame\_sp++=(int32)(1); | push 1 to stack |
| jeff\_iconst\_2 |  | PUSH\_I(2):  \*(int32\*)frame\_sp++=(int32)(2); | push 2 to stack |
| jeff\_iconst\_3 |  | PUSH\_I(3):  \*(int32\*)frame\_sp++=(int32)(3); | push 3 to stack |
| jeff\_iconst\_4 |  | PUSH\_I(4):  \*(int32\*)frame\_sp++=(int32)(4); | push 4 to stack |
| jeff\_iconst\_5 |  | PUSH\_I(5):  \*(int32\*)frame\_sp++=(int32)(5); | push 5 to stack |
| jeff\_lconst\_0 |  | PUSH\_L(0):  PUT\_INT64\_TO\_ADDR(frame\_sp, 0);  frame\_sp += 2; | push 0 to stack, two bytes |
| jeff\_lconst\_1 |  | PUSH\_L(1):  PUT\_INT64\_TO\_ADDR(frame\_sp, 1);  frame\_sp += 2; | push 1 to stack, two bytes |
| jeff\_fconst\_0 |  | PUSH\_F(0):  \*(float\*)frame\_sp++ = (float)(0); | push float 0 to stack |
| jeff\_fconst\_1 |  | PUSH\_F(1):  \*(float\*)frame\_sp++ = (float)(1); | push float 1 to stack |
| jeff\_fconst\_2 |  | PUSH\_F(2):  \*(float\*)frame\_sp++ = (float)(2); | push float 2 to stack |
| jeff\_dconst\_0 |  | PUSH\_D(0):  PUT\_DOUBLE\_TO\_ADDR (frame\_sp, 0);  frame\_sp += 2; | push double 0 to stack, two bytes |
| jeff\_dconst\_1 |  | PUSH\_D(1):  PUT\_DOUBLE\_TO\_ADDR (frame\_sp, 1);  frame\_sp += 2; | push double 1 to stack, two bytes |
| jeff\_bipush |  | PUSH\_I (read\_int8 (frame\_ip)):  [read\_xxx(p): return p and p++] | read a byte value from ip, and push it as an integer to stack |
| jeff\_sipush |  | PUSH\_I(read\_int16\_align(frame\_ip)); | read a short value from ip, and push it as an integer to stack |
| jeff\_iload |  | PUSH\_I (LOCAL\_I (read\_uint8 (frame\_ip)));  [LOCAL\_I(n): (\*(int32 \*)(frame\_lp + n))] | read a byte value from ip, and use it as the position of local variable to read local variable int value, and push the int value to stack |
| jeff\_lload |  | PUSH\_L (LOCAL\_L (read\_uint8 (frame\_ip)));  [LOCAL\_L(n): (GET\_INT64\_FROM\_ADDR (frame\_lp + n))] | read a byte value from ip, and use it as the position of local variable to read local variable long value, and push the long value to stack |
| jeff\_fload |  | PUSH\_F (LOCAL\_F (read\_uint8 (frame\_ip)));  [LOCAL\_F(n): (\*(float \*)(frame\_lp + n))] | read a byte value from ip, and use it as the position of local variable to read local variable float value, and push the float value to stack |
| jeff\_dload |  | PUSH\_D (LOCAL\_D (read\_uint8 (frame\_ip)));  [LOCAL\_D(n): (GET\_DOUBLE\_FROM\_ADDR (frame\_lp + n))] | read a byte value from ip, and use it as the position of local variable to read local variable double value, and push the double value to stack |
| jeff\_aload |  | PUSH\_R(LOCAL\_R(read\_uint8(frame\_ip)));  [LOCAL\_R(n): (\*(JeffObjectRef \*)(frame\_lp + n)) | read a byte value from ip, and use it as the position of local variable to read local variable ref value, and push the ref value to stack |
| jeff\_iload\_0 |  | PUSH\_I (LOCAL\_I (0)); | read int value from position 0 of local variable table, and push the int value to stack |
| jeff\_iload\_1 |  | PUSH\_I (LOCAL\_I (1)); | read int value from position 1 of local variable table, and push the int value to stack |
| jeff\_iload\_2 |  | PUSH\_I (LOCAL\_I (2)); | read int value from position 2 of local variable table, and push the int value to stack |
| jeff\_iload\_3 |  | PUSH\_I (LOCAL\_I (3)); | read int value from position 3 of local variable table, and push the int value to stack |
| jeff\_lload\_0 |  | PUSH\_L (LOCAL\_L (0)); | read long value from position 0 of local variable table, and push the long value to stack |
| jeff\_lload\_1 |  | PUSH\_L (LOCAL\_L (1)); | read long value from position 1 of local variable table, and push the long value to stack |
| jeff\_lload\_2 |  | PUSH\_L (LOCAL\_L (2)); | read long value from position 2 of local variable table, and push the long value to stack |
| jeff\_lload\_3 |  | PUSH\_L (LOCAL\_L (3)); | read long value from position 3 of local variable table, and push the long value to stack |
| jeff\_fload\_0 |  | PUSH\_F (LOCAL\_F (0)); | read float value from position 0 of local variable table, and push the float value to stack |
| jeff\_fload\_1 |  | PUSH\_F (LOCAL\_F (1)); | read float value from position 1 of local variable table, and push the float value to stack |
| jeff\_fload\_2 |  | PUSH\_F (LOCAL\_F (2)); | read float value from position 2 of local variable table, and push the float value to stack |
| jeff\_fload\_3 |  | PUSH\_F (LOCAL\_F (3)); | read float value from position 3 of local variable table, and push the float value to stack |
| jeff\_dload\_0 |  | PUSH\_D (LOCAL\_D(0)); | read double value from position 0 of local variable table, and push the double value to stack |
| jeff\_dload\_1 |  | PUSH\_D (LOCAL\_D(1)); | read double value from position 1 of local variable table, and push the double value to stack |
| jeff\_dload\_2 |  | PUSH\_D (LOCAL\_D(2)); | read double value from position 2 of local variable table, and push the double value to stack |
| jeff\_dload\_3 |  | PUSH\_D (LOCAL\_D(3)); | read double value from position 3 of local variable table, and push the double value to stack |
| jeff\_aload\_0 |  | PUSH\_R (LOCAL\_R(0)); | read ref value from position 0 of local variable table, and push the ref value to stack |
| jeff\_aload\_1 |  | PUSH\_R (LOCAL\_R(1)); | read ref value from position 1 of local variable table, and push the ref value to stack |
| jeff\_aload\_2 |  | PUSH\_R (LOCAL\_R(2)); | read ref value from position 2 of local variable table, and push the ref value to stack |
| jeff\_aload\_3 |  | PUSH\_R (LOCAL\_R(3)); | read ref value from position 3 of local variable table, and push the ref value to stack |
| jeff\_iaload | 数组 | DEF\_OPCODE\_XALOAD(int32, I) | pop index from stack  pop array from stack  read int32 array[index]  push array[index] as int to stack |
| jeff\_laload |  | DEF\_OPCODE\_XALOAD(int64, L) | pop index from stack  pop array from stack  read long array[index]  push array[index] as long to stack |
| jeff\_faload |  | DEF\_OPCODE\_XALOAD(float, F) | pop index from stack  pop array from stack  read float array[index]  push array[index] as float to stack |
| jeff\_daload |  | DEF\_OPCODE\_XALOAD(double, D) | pop index from stack  pop array from stack  read double array[index]  push array[index] as double to stack |
| jeff\_aaload |  | DEF\_OPCODE\_XALOAD(JeffObjectRef, R) | pop index from stack  pop array from stack  read ref array[index]  push array[index] as ref to stack |
| jeff\_baload |  | DEF\_OPCODE\_XALOAD(int8, I) | pop index from stack  pop array from stack  read int8 array[index]  push array[index] as int to stack |
| jeff\_caload |  | DEF\_OPCODE\_XALOAD(int16, I) | pop index from stack  pop array from stack  read int16 array[index]  push array[index] as int to stack |
| jeff\_saload |  | DEF\_OPCODE\_XALOAD(int16, I) | pop index from stack  pop array from stack  read int 16array[index]  push array[index] as int to stack |
| jeff\_istore |  | SET\_LOCAL\_I (read\_uint8 (frame\_ip), POP\_I ());  SET\_LOCAL\_I:  int n = (N);  frame\_ref[n] = 0;  \*(int32\*)(frame\_lp+n)=(int32)(val); | read uint8 as local variable index from ip  pop an int value from stack  store the value to that index of lp and set ref[index] to 0 |
| jeff\_lstore |  | SET\_LOCAL\_L (read\_uint8 (frame\_ip), POP\_L ()); | read uint8 as local variable index from ip  pop a long value from stack  store the value to that index of lp and set ref[index] to 0 |
| jeff\_fstore |  | SET\_LOCAL\_F (read\_uint8 (frame\_ip), POP\_F ()); | read uint8 as local variable index from ip  pop a float value from stack  store the value to that index of lp and set ref[index] to 0 |
| jeff\_dstore |  | SET\_LOCAL\_D (read\_uint8 (frame\_ip), POP\_D ()); | read uint8 as local variable index from ip  pop a double value from stack  store the value to that index of lp and set ref[index] to 0 |
| jeff\_astore |  | STORE\_LOCAL\_R (read\_uint8 (frame\_ip)); | read uint8 as local variable index from ip  pop a ref value from stack  store the value to that index of lp and set ref[index] to 1 |
| jeff\_istore\_0 |  | SET\_LOCAL\_I (0, POP\_I ()); | pop an int value from stack  store the value to position 0 of lp and set ref[0] to 0 |
| jeff\_istore\_1 |  | SET\_LOCAL\_I (1, POP\_I ()); | pop an int value from stack  store the value to position 1 of lp and set ref[1] to 0 |
| jeff\_istore\_2 |  | SET\_LOCAL\_I (2, POP\_I ()); | pop an int value from stack  store the value to position 2 of lp and set ref[2] to 0 |
| jeff\_istore\_3 |  | SET\_LOCAL\_I (3, POP\_I ()); | pop an int value from stack  store the value to position 3 of lp and set ref[3] to 0 |
| jeff\_lstore\_0 |  | SET\_LOCAL\_L (0, POP\_L ()); | pop an long value from stack  store the value to position 0 of lp and set ref[0] to 0 |
| jeff\_lstore\_1 |  | SET\_LOCAL\_L (1, POP\_L ()); | pop an long value from stack  store the value to position 1 of lp and set ref[1] to 0 |
| jeff\_lstore\_2 |  | SET\_LOCAL\_L (2, POP\_L ()); | pop an long value from stack  store the value to position 2 of lp and set ref[2] to 0 |
| jeff\_lstore\_3 |  | SET\_LOCAL\_L (3, POP\_L ()); | pop an long value from stack  store the value to position 3 of lp and set ref[3] to 0 |
| jeff\_fstore\_0 |  |  |  |
| jeff\_fstore\_1 |  |  |  |
| jeff\_fstore\_2 |  |  |  |
| jeff\_fstore\_3 |  |  |  |
| jeff\_dstore\_0 |  |  |  |
| jeff\_dstore\_1 |  |  |  |
| jeff\_dstore\_2 |  |  |  |
| jeff\_dstore\_3 |  |  |  |
| jeff\_astore\_0 |  |  |  |
| jeff\_astore\_1 |  |  |  |
| jeff\_astore\_2 |  |  |  |
| jeff\_astore\_3 |  |  |  |
| jeff\_iastore | 数组 |  |  |
| jeff\_lastore |  |  |  |
| jeff\_fastore |  |  |  |
| jeff\_dastore |  |  |  |
| jeff\_aastore |  |  |  |
| jeff\_bastore |  |  |  |
| jeff\_castore |  |  |  |
| jeff\_sastore |  |  |  |
|  |  |  |  |
| jeff\_pop |  | POP(1) | pop an element from stack and set the popped original memory to zero |
| jeff\_pop2 |  | POP(2) | pop two elements from stack and set the popped original memory to zero |
| jeff\_dup |  | frame\_sp[0] = frame\_sp[-1]  frame\_sp++ | duplicate value form the top of stack and push it to stack |
| jeff\_dup\_x1 |  | frame\_sp[0] = frame\_sp[-1];  frame\_sp[-1] = frame\_sp[-2];  frame\_sp[-2] = frame\_sp[0];  frame\_sp++; | duplicate value from to top of stack and push it to stack  the original top = original top -1  the orginal top-1 = current value |
| jeff\_op\_dup\_x2 |  | frame\_sp[0] = frame\_sp[-1];  frame\_sp[-1] = frame\_sp[-2];  frame\_sp[-2] = frame\_sp[-3];  frame\_sp[-3] = frame\_sp[0];  frame\_sp++; | duplicate value from to top of stack and push it to stack  the original top = original top -1  the orginal top-1 = original top -2  the original top - 2 = current value |
| jeff\_dup2 |  | frame\_sp[1] = frame\_sp[-1];  frame\_sp[0] = frame\_sp[-2];  frame\_sp += 2; | duplicate two vaue form the top of the stack and push them to stack |
| jeff\_dup2\_x1 |  |  |  |
| jeff\_dup2\_x2 |  |  |  |
| jeff\_swap |  | uint32 tmp = frame\_sp[-1];  frame\_sp[-1] = frame\_sp[-2];  frame\_sp[-2] = tmp; | swap the top elements of the stack |
| jeff\_iadd |  | DEF\_OPCODE\_MATH\_BINARY (I, I, I, +); | pop two int elements from stack  push the add result of them to stack |
| jeff\_ladd |  | DEF\_OPCODE\_MATH\_BINARY (L, L, L, +); | pop two long elements from stack  push the add result of them to stack |
| jeff\_fadd |  |  |  |
| jeff\_dadd |  |  |  |
| jeff\_isub |  |  |  |
| jeff\_lsub |  |  |  |
| jeff\_fsub |  |  |  |
| jeff\_dsub |  |  |  |
| jeff\_imul |  |  |  |
| jeff\_lmul |  |  |  |
| jeff\_fmul |  |  |  |
| jeff\_dmul |  |  |  |
| jeff\_idiv |  |  |  |
| jeff\_ldiv |  |  |  |
| jeff\_fdiv |  |  |  |
| jeff\_ddiv |  |  |  |
| jeff\_irem | 取模 |  |  |
| jeff\_lrem |  |  |  |
| jeff\_frem |  |  |  |
| jeff\_drem |  |  |  |
| jeff\_ineg | 取反 |  |  |
| jeff\_lneg |  |  |  |
| jeff\_fneg |  |  |  |
| jeff\_dneg |  |  |  |
| jeff\_ishl | I<< |  |  |
| jeff\_lshl | L<< |  |  |
| jeff\_ishr | I>> |  |  |
| jeff\_lshr | L>> |  |  |
| jeff\_iushr |  | 无符号整数右移 |  |
| jeff\_lushr |  | 无符号长整数右移 |  |
| jeff\_iand | I& |  |  |
| jeff\_land | L& |  |  |
| jeff\_ior | I| |  |  |
| jeff\_lor | L| |  |  |
| jeff\_ixor | I^ |  |  |
| jeff\_lxor | L^ |  |  |
| jeff\_iinc | I+=n |  |  |
| jeff\_i2l |  |  |  |
| jeff\_i2f |  |  |  |
| jeff\_i2d |  |  |  |
| jeff\_l2i |  |  |  |
| jeff\_l2f |  |  |  |
| jeff\_l2d |  |  |  |
| jeff\_f2i |  |  |  |
| jeff\_f2l |  |  |  |
| jeff\_f2d |  |  |  |
| jeff\_d2i |  |  |  |
| jeff\_d2l |  |  |  |
| jeff\_d2f |  |  |  |
| jeff\_i2b |  |  |  |
| jeff\_i2c |  |  |  |
| jeff\_i2s |  |  |  |
| 条件语句 |  |  |  |
| jeff\_lcmp |  |  |  |
| jeff\_fcmpl |  |  |  |
| jeff\_fcmpg |  |  |  |
| jeff\_dcmpl |  |  |  |
| jeff\_dcmpg |  |  |  |
| jeff\_ifeq |  |  |  |
| jeff\_ifne |  |  |  |
| jeff\_iflt |  |  |  |
| jeff\_ifge |  |  |  |
| jeff\_ifgt |  |  |  |
| jeff\_ifle |  |  |  |
| jeff\_if\_icmpeq |  |  |  |
| jeff\_if\_icmpne |  |  |  |
| jeff\_if\_icmplt |  |  |  |
| jeff\_if\_icmpge |  |  |  |
| jeff\_if\_icmpgt |  |  |  |
| jeff\_if\_icmple |  |  |  |
| jeff\_if\_acmpeq |  |  |  |
| jeff\_if\_acmpne |  |  |  |
| jeff\_goto |  |  |  |
| jeff\_tablewitch |  |  |  |
| jeff\_op\_lookupswitch |  |  |  |
| jeff\_ireturn |  |  |  |
| jeff\_freturn |  |  |  |
| jeff\_areturn |  |  |  |
| jeff\_lreturn |  |  |  |
| jeff\_dreturn |  |  |  |
| jeff\_return |  |  |  |
| jeff\_getstatic |  |  |  |
| jeff\_putstatic |  |  |  |
| jeff\_getfield |  |  |  |
| jeff\_putfield |  |  |  |
| jeff\_ invokevirtual |  |  |  |
| jeff\_ invokespecial |  |  |  |
| jeff\_invokestatic |  |  |  |
| jeff\_ invokeinterface |  |  |  |
| jeff\_new |  |  |  |
| jeff\_newarray |  |  |  |
| jeff\_arraylength |  |  |  |
| jeff\_athrow |  |  |  |
| jeff\_checkcast |  |  |  |
| jeff\_instanceof |  |  |  |
| jeff\_monitorenter |  |  |  |
| jeff\_monitorexit |  |  |  |
| jeff\_multianewarray |  |  |  |
| jeff\_ifnull |  |  |  |
| jeffjeff\_ifnonnull |  |  |  |
| jeff\_breakpoint |  |  |  |
| jeff\_newconstarray |  |  |  |
| jeff\_slookupswitch |  |  |  |
| jeff\_stableswith |  |  |  |
| jeff\_iinc\_w |  |  |  |
| jeff\_cldc |  |  |  |
| jeff\_sldc\_index |  |  |  |
| jeff\_ildc |  |  |  |
| jeff\_lldc |  |  |  |
| jeff\_fldc |  |  |  |
| jeff\_dldc |  |  |  |
| jeff\_dload\_w |  |  |  |
| jeff\_dstore\_w |  |  |  |
| jeff\_fload\_w |  |  |  |
| jeff\_fstore\_w |  |  |  |
| jeff\_iload\_w |  |  |  |
| jeff\_istore\_w |  |  |  |
| jeff\_lload\_w |  |  |  |
| jeff\_lstore\_w |  |  |  |
| jeff\_aload\_w |  |  |  |
| jeff\_astore\_w |  |  |  |
| jeff\_impdep2 |  |  |  |