Intermediate Representation

Instructions

This document briefly describes which field of struct instruction is used by which operation.

Some of those fields are used by almost all instructions, some others are specific to only one or a few instructions. The common ones are:

- .src1, .src2, .src3: (pseudo_t) operands of binops or ternary ops.
- .src: (pseudo_t) operand of unary ops (alias for .src1).
- .target: (pseudo_t) result of unary, binary & ternary ops, is sometimes used otherwise by some others instructions.
- .cond: (pseudo_t) input operands for condition (alias .src/.src1)
- .type: (symbol*) usually the type of .result, sometimes of the operands

Terminators

OP_RET

Return from subroutine.

- .src : returned value (NULL if void)
- .type: type of .src

OP_BR

Unconditional branch

- .bb_true: destination basic block

OP_CBR

Conditional branch

- .cond: condition
- .type: type of .cond, must be an integral type
- .bb_true, .bb_false: destination basic blocks

OP_SWITCH

Switch / multi-branch

- .cond: condition
- .type: type of .cond, must be an integral type
- .multijmp_list: pairs of case-value - destination basic block

OP_UNREACH

Mark code as unreachable
OP_COMPUTEDGOTO

Computed goto / branch to register

- .src: address to branch to (void*)
- .multijmp_list: list of possible destination basic blocks

Arithmetic binops

They all follow the same signature:

- .src1, .src2: operands (types must be compatible with .target)
- .target: result of the operation (must be an integral type)
- .type: type of .target

OP_ADD

Integer addition.
OP_SUB

Integer subtraction.
OP_MUL

Integer multiplication.
OP_DIVU

Integer unsigned division.
OP_DIVS

Integer signed division.
OP_MODU

Integer unsigned remainder.
OP_MODS

Integer signed remainder.
OP_SHL

Shift left (integer only)
OP_LSR

Logical Shift right (integer only)
OP_ASR

Arithmetic Shift right (integer only)

Floating-point binops

They all follow the same signature:

- .src1, .src2: operands (types must be compatible with .target)
- .target: result of the operation (must be a floating-point type)
- .type: type of .target

OP_FADD

Floating-point addition.
OP_FSUB

Floating-point subtraction.
OP_FMUL

Floating-point multiplication.
OP_FDIV

Floating-point division.

Logical ops

They all follow the same signature:

- .src1, .src2: operands (types must be compatible with .target)
- .target: result of the operation
- .type: type of .target, must be an integral type

OP_AND

Logical AND
OP_OR

Logical OR
OP_XOR

Logical XOR

Integer compares

They all have the following signature:

- .src1, .src2: operands (types must be compatible)
- .target: result of the operation (0/1 valued integer)
- .type: type of .target, must be an integral type
- .itype: type of the input operands

OP_SET_EQ

Compare equal.
OP_SET_NE

Compare not-equal.
OP_SET_LE

Compare less-than-or-equal (signed).
OP_SET_GE

Compare greater-than-or-equal (signed).
OP_SET_LT

Compare less-than (signed).
OP_SET_GT

Compare greater-than (signed).
OP_SET_B

Compare less-than (unsigned).
OP_SET_A

Compare greater-than (unsigned).
OP_SET_BE

Compare less-than-or-equal (unsigned).
OP_SET_AE

Compare greater-than-or-equal (unsigned).

Floating-point compares

They all have the same signature as the integer compares.

The usual 6 operations exist in two versions: ‘ordered’ and ‘unordered’. These operations first check if any operand is a NaN and if it is the case the ordered compares return false and then unordered return true, otherwise the result of the comparison, now guaranteed to be done on non-NaNs, is returned.

OP_FCMP_OEQ

Floating-point compare ordered equal
OP_FCMP_ONE

Floating-point compare ordered not-equal
OP_FCMP_OLE

Floating-point compare ordered less-than-or-equal
OP_FCMP_OGE

Floating-point compare ordered greater-or-equal
OP_FCMP_OLT

Floating-point compare ordered less-than
OP_FCMP_OGT

Floating-point compare ordered greater-than
OP_FCMP_UEQ

Floating-point compare unordered equal
OP_FCMP_UNE

Floating-point compare unordered not-equal
OP_FCMP_ULE

Floating-point compare unordered less-than-or-equal
OP_FCMP_UGE

Floating-point compare unordered greater-or-equal
OP_FCMP_ULT

Floating-point compare unordered less-than
OP_FCMP_UGT

Floating-point compare unordered greater-than
OP_FCMP_ORD

Floating-point compare ordered: return true if both operands are ordered (none of the operands are a NaN) and false otherwise.
OP_FCMP_UNO

Floating-point compare unordered: return false if no operands is ordered and true otherwise.

Unary ops

OP_NOT

Logical not.

- .src: operand (type must be compatible with .target)
- .target: result of the operation
- .type: type of .target, must be an integral type

OP_NEG

Integer negation.

- .src: operand (type must be compatible with .target)

- .target: result of the operation (must be an integral type)
- .type: type of .target

OP_FNEG

Floating-point negation.

- .src: operand (type must be compatible with .target)
- .target: result of the operation (must be a floating-point type)
- .type: type of .target

OP_SYMADDR

Create a pseudo corresponding to the address of a symbol.

- .src: input symbol (must be a PSEUDO_SYM)
- .target: symbol's address

OP_COPY

Copy (only needed after out-of-SSA).

- .src: operand (type must be compatible with .target)
- .target: result of the operation
- .type: type of .target

Type conversions

They all have the following signature:

- .src: source value
- .orig_type: type of .src
- .target: result value
- .type: type of .target

Currently, a cast to a void pointer is treated like a cast to an unsigned integer of the same size.

OP_TRUNC

Cast from integer to an integer of a smaller size.
OP_SEXT

Cast from integer to an integer of a bigger size with sign extension.
OP_ZEXT

Cast from integer to an integer of a bigger size with zero extension.
OP_UTPTR

Cast from pointer-sized unsigned integer to pointer type.
OP_PTRTU

Cast from pointer type to pointer-sized unsigned integer.
OP_PTRCAST

Cast between pointers.
OP_FCVTU

Conversion from float type to unsigned integer.
OP_FCVTS

Conversion from float type to signed integer.
OP_UCVTF

Conversion from unsigned integer to float type.
OP_SCVTF

Conversion from signed integer to float type.
OP_FCVTF

Conversion between float types.

Ternary ops

OP_SEL

- .src1: condition, must be of integral type
- .src2, .src3: operands (types must be compatible with .target)
- .target: result of the operation
- .type: type of .target

OP_FMADD

Fused multiply-add.

- .src1, .src2, .src3: operands (types must be compatible with .target)
- .target: result of the operation (must be a floating-point type)
- .type: type of .target

OP_RANGE

Range/bounds checking (only used for an unused sparse extension).

- .src1: value to be checked
- .src2, src3: bound of the value (must be constants?)
- .type: type of .src[123]?

Memory ops

OP_LOAD

Load.

- .src: base address to load from
- .offset: address offset
- .target: loaded value
- .type: type of .target

OP_STORE

Store.

- .src: base address to store to
- .offset: address offset
- .target: value to be stored
- .type: type of .target

Others

OP_SETFVAL

Create a pseudo corresponding to a floating-point literal.

- .fvalue: the literal's value (long double)
- .target: the corresponding pseudo
- .type: type of the literal & .target

OP_SETVAL

Create a pseudo corresponding to a string literal. The value is given as an expression EXPR_STRING.

- .val: (expression) input expression
- .target: the resulting value
- .type: type of .target, the value

OP_LABEL

Create a pseudo corresponding to a label-as-value.

- .bb_true: the BB corresponding to the label
- .target: the resulting value
- .type: type of .target (void *)

OP_PHI

Phi-node (for SSA form).

- .phi_list: phi-operands (type must be compatible with .target)
- .target: "result"
- .type: type of .target

OP_PHISOURCE

Phi-node source. Like OP_COPY but exclusively used to give a defining instructions (and thus also a type) to *all* OP_PHI operands.

- .phi_src: operand (type must be compatible with .target, alias .src)
- .target: the "result" PSEUDO_PHI
- .type: type of .target
- .phi_node: the unique phi instruction using the target pseudo

OP_CALL

Function call.

- .func: (pseudo_t) the function (can be a symbol or a "register", alias .src))
- .arguments: (pseudo_list) list of the associated arguments
- .target: function return value (if any)
- .type: type of .target
- .fntypes: (symbol_list) list of the function's types: the first entry is the full function type, the next ones are the type of each arguments

OP_INLINED_CALL

Only used as an annotation to show that the instructions just above correspond to a function that

have been inlined.

- .func: (pseudo_t) the function (must be a symbol, alias .src))
- .arguments: list of pseudos that where the function's arguments
- .target: function return value (if any)
- .type: type of .target

OP_SLICE

Extract a "slice" from an aggregate.

- .base: (pseudo_t) aggregate (alias .src)
- .from: offset of the "slice" within the aggregate
- .target: result
- .type: type of .target

OP_ASM

Inlined assembly code.

- .string: asm template
- .asm_rules: asm constraints, rules

Sparse tagging (line numbers, context, whatever)

OP_CONTEXT

Currently only used for lock/unlock tracking.

- .context_expr: unused
- .increment: (1 for locking, -1 for unlocking)
- .check: (ignore the instruction if 0)

Misc ops

OP_ENTRY

Function entry point (no associated semantic).
OP_BADOP

Invalid operation (should never be generated).
OP_NOP

No-op (should never be generated).
OP_DEATHNOTE

Annotation telling the pseudo will be death after the next instruction (other than some other annotation, that is).