

# Dagger

## Decompiling to IR

Ahmed Bougacha

with Geoffroy Aubey, Pierre Collet, Thomas Coudray, Jonathan Salwan, Amaury de la Vieuville

Semantics ?

The decompilation process

Use cases & tools

# Semantics

Binary > IR

x86

```
add rax, 15
```

```
sub [rbx + 8], rax
```

x86

IR

```
add rax, 15
```

```
%rax2 = add i64 %rax1, 15
```

x86

IR

add rax, 15

%rax2 = add i64 %rax1, 15

x86

IR

```
add rax, 15
```

```
%rax2 = add i64 %rax1, 15
```

x86

```
add rax, 15
```

IR

```
%rax2 = add i64 %rax1, 15
```



x86

```
add rax, 15
```

IR

```
%rax2 = add i64 %rax1, 15
```

x86

```
sub [rbx + 8], rax
```

IR

```
%1 = add i64 %rbx1, 8  
%2 = inttoptr i64 %1 to i64*  
%3 = load i64* %2  
%4 = sub i64 %3, %rax2  
store i64 %4, i64* %2
```

x86

```
sub [rbx + 8], rax
```

IR

```
%1 = add i64 %rbx1, 8  
%2 = inttoptr i64 %1 to i64*  
%3 = load i64* %2  
%4 = sub i64 %3, %rax2  
store i64 %4, i64* %2
```

x86

```
sub [rbx + 8], rax
```

IR

```
%1 = add i64 %rbx1, 8  
%2 = inttoptr i64 %1 to i64*  
%3 = load i64* %2  
%4 = sub i64 %3, %rax2  
store i64 %4, i64* %2
```

x86

```
sub [rbx + 8], rax
```

IR

```
%1 = add i64 %rbx1, 8  
%2 = inttoptr i64 %1 to i64*  
%3 = load i64* %2  
%4 = sub i64 %3, %rax2  
store i64 %4, i64* %2
```

x86

```
add rax, 15
```

```
sub [rbx + 8], rax
```

IR

```
%rax2 = add i64 %rax1, 15
```

```
%1 = add i64 %rbx1, 8
```

```
%2 = inttoptr i64 %1 to i64*
```

```
%3 = load i64* %2
```

```
%4 = sub i64 %3, %rax2
```

```
store i64 %4, i64* %2
```

# Dozens of SUBs:

x86

```
...  
sub reg32, reg32 // SUB32rr  
sub mem32, reg32 // SUB32mr  
sub reg32, imm32 // SUB32ri  
sub reg64, reg64 // SUB64rr  
...
```

# Dozens of SUBs:

x86

```
...  
sub reg32, reg32  
sub mem32, reg32  
sub reg32, imm32  
sub reg64, reg64  
...
```

IR

```
%dst = sub iXX %src1, %src2
```



# Defining Semantics

Binary > Mir > IR

```
def SUB : InstructionSemantics<[  
  (set vop0, (sub vop1, vop2))  
  ]>;
```

```
def : OpcodesSemantics<  
    SUB,  
    [SUB32ri, SUB32mr, SUB32rr, ...]  
>;
```

# TableGen Operands

```
def GR32 // RegisterClass
...
def i32mem // Operand
...

def SUB32mr { // Instruction
...
    dag OutOperandList = (outs);
    dag InOperandList = (ins i32mem:$dst, GR32:$src);
...
}
```

# MC Operands

```
sub [ebx + 8], eax
```

```
## <MCInst #2562 SUB32mr  
## <MCOperand Reg:45>  
## <MCOperand Imm:1>  
## <MCOperand Reg:0>  
## <MCOperand Imm:8>  
## <MCOperand Reg:0>  
## <MCOperand Reg:43>>
```

# Virtual Operands

# Virtual Operands

*Input*

Register class:                   get the register value

Operand:                           look for OperandMapping

# Virtual Operands

*Output*

Register class:

put the value in the register

Operand:

look for OperandMapping



# Operand Mapping: Register Classes

```
def : OperandMapping<  
    GR32,  
    /* In */ (get mc_op0),  
    /* Out */ (put mc_op0, result)  
>;
```

# Operand Mapping: Immediates

```
def : OperandMapping<  
    imm32,  
    /* In */ (mov mc_op0),  
    /* Out */ ()  
>;
```

# Operand Mapping: Custom Operands

```
// base + index * scale + offset
//  op0 +   op1 *   op2 +   op3

def BIS0 : SemaFrag<
    (add mc_op0,
        (add mc_op3,
            (mul mc_op1,
                mc_op2))))
>;
```

# Operand Mapping: Custom Operands

```
def : OperandMapping<  
    i32mem,  
    /* In */ (load (BIS0)),  
    /* Out */ (store (BIS0), result)  
>;
```

# Virtual Operand Expansion

```
(sub vop1, vop2)
```

*SUB32mr*

```
(sub  
  (load (add ..)),  
  (get mc_op5))
```

# Virtual Operand Expansion

(sub vop1, vop2)

*SUB32mr*

*SUB32ri*

```
(sub  
  (load (add ..)),  
  (get mc_op5))
```

```
(sub  
  (get mc_op0),  
  (mov mc_op1))
```

# Virtual Operand Expansion

(sub vop1, vop2)

*SUB32ri*

*untyped expression tree*

```
(sub  
  (get mc_op0),  
  (mov mc_op1))
```

*typed instruction list*

```
%0 = get32 mcop0  
%1 = mov32 mcop1  
%r = sub32 %0, %1
```

# Mir

Binary > Mir > IR



# Mir: Target registers

```
get %td0, 4  
...  
put 4, %td3
```

## Mir: Advance

9: 81 c3 d2 04 00 00            add ebx, 1234

advance @9

```
get %td0, EBX
mov %td1, 1234
add %td2, %td0, %td1
put EBX, %td2
advance +6
```

IR

Binary > Mir > IR

# Generating IR

x86

Mir

IR

```
sub ebx, ecx
```

```
...  
sub %td2, ...  
put EBX, %td2
```

```
add ebx, 12
```

```
get %td0, EBX  
mov %td1, 12  
add %td2, %td0, %td1  
put EBX, %td2
```

# Generating IR

x86

Mir

IR

sub ebx, ecx

...  
sub %td2, ...  
put EBX, %td2

...  
**%ebx2 = sub i32 ...**

add ebx, 12

get %td0, EBX  
mov %td1, 12  
**add %td2, %td0, %td1**  
put EBX, %td2

**%ebx3 = add i32**

# Generating IR

x86

Mir

IR

```
sub ebx, ecx
```

```
...  
sub %td2, ...  
put EBX, %td2
```

```
...  
%ebx2 = sub i32 ...
```

```
add ebx, 12
```

```
get %td0, EBX  
mov %td1, 12  
add %td2, %td0, %td1  
put EBX, %td2
```

```
%ebx3 = add i32 %ebx2
```

# Generating IR

x86

Mir

IR

```
sub ebx, ecx
```

```
...  
sub %td2, ...  
put EBX, %td2
```

```
...  
%ebx2 = sub i32 ...
```

```
add ebx, 12
```

```
get %td0, EBX  
mov %td1, 12  
add %td2, %td0, %td1  
put EBX, %td2
```

```
%ebx3 = add i32 %ebx2
```

# Generating IR

x86

Mir

IR

sub ebx, ecx

...  
sub %td2, ...  
put EBX, %td2

...  
**%ebx2** = sub i32 ...

add ebx, 12

get %td0, EBX  
mov %td1, 12  
add %td2, %td0, %td1  
put EBX, %td2

...  
**%ebx3** = add i32 %ebx2



# Generating IR

x86

Mir

IR

sub ebx, ecx

...  
sub %td2, ...  
put EBX, %td2

...  
**%ebx2** = sub i32 ...

add ebx, 12

get %td0, EBX  
mov %td1, 12  
add %td2, **%td0**, %td1  
put EBX, %td2

...  
**%ebx3** = add i32 **%ebx2**

# Generating IR

x86

Mir

IR

sub ebx, ecx

...  
sub %td2, ...  
put EBX, %td2

...  
%ebx2 = sub i32 ...

add ebx, 12

get %td0, EBX  
**mov %td1, 12**  
add %td2, %td0, **%td1**  
put EBX, %td2

...  
%ebx3 = add i32 %ebx2, **12**

# Generating IR

x86

Mir

IR

sub ebx, ecx

...  
sub %td2, ...  
put EBX, %td2

...  
**%ebx2** = sub i32 ...

add ebx, 12

get %td0, EBX  
mov %td1, 12  
add %td2, %td0, %td1  
put EBX, %td2

**%ebx3** = add i32 %ebx2, 12

# Generating Branches

```
22: 48 83 c1 08    add rcx, 8
...
xx: xx xx xx xx  jmp 22
```

Mir

```
advance @22
get %tq0, RCX
mov %tq1, 8
add %tq2, %tq0, %tq1
put RCX, %tq2
advance +4
...
jmp 22
```

IR

```
I22:
%rcx2 = add i64 %rcx1, 8
br label %I22
```

# Generating Indirect Branches

```
22: 48 83 c1 08    add rcx, 8
26: 83 eb 03      sub ebx, 3
```

```
JumpTable:
%p = phi ...
```

```
switch i64 %p, label %fail
  [i64 22, label %I22
   i64 26, label %I26]
```

```
I22:
%rcx2 = add i64 %rcx1, 8
I26:
%ebx2 = sub i64 %ebx1, 3
```

# Generating Predicated Instructions

```
addge r7, r5, #1
```

Mir

```
get %td0, R5  
mov %td1, 1  
add %td2, %td0, %td1  
select %td3, xx, %td2, %td0  
put R7, %td3
```

IR

```
%1 = add i32 %r5_1, 1  
%r7_2 = select xx, i64 %1, %r5_1
```

# Generating Condition Codes

```
22: 48 83 c1 08    add rcx, 8
26: xx xx xx xx    jne 22
```

## Mir

```
advance @22
get %tq0, RCX
mov %tq1, 8
add %tq2, %tq0, %tq1
...
cmpne %f3, %tq2
...
put RCX, %tq2
advance +4

jmpne 22
```

## IR

```
I22:

%rcx2 = add i64 %rcx1, 8
%ne2 = icmp ne i64 %rcx2, 0

br i1 %ne2, label %I22
```

# Using the IR

IR > ?



# Binary Rewriting

# Binary Rewriting

Missing semantics → Inline assembly

# Binary Rewriting

Missing semantics → Inline assembly

Data sections → Map it all

# Static Binary Translation

# Dynamic Binary Translation

# Dynamic Binary Translation

Self-altering code → Mark read/execute

# Dynamic Binary Translation

Self-altering code → Mark read/execute

Code discovery → Per-BB translation

# Dynamic Binary Instrumentation



# Binary Analysis

Simulation

# Simulation

Missing semantics → Runtime library

# Simulation

Missing semantics → Runtime library

Cycle accuracy → Machine Model?

# To-Source Decompile

# To-source Decompile

C source output → C Backend!

# To-source Decompilation

C source output → C Backend!

IR “highering” → Optimizations

# To-source Decompilation

C source output → C Backend!

IR “highering” → Optimizations

Lack of accuracy → Metadata



Going forward

Going forward

Merging semantics with SD patterns?

# Going forward

Merging semantics with SD patterns?

Removing the Mir backend

# Going forward

Merging semantics with SD patterns?

Removing the Mir backend

Analyzes & Highering

# Going forward

Merging semantics with SD patterns?

Removing the Mir backend

Analyzes & Highering

Tools!

# Questions?

<http://dagger.repzret.org>