The Journey of a Source Line (1)

CERN Academic Training – Feb 2018

Andrzej Nowak

http://tik.services
http://tik.services
/* General case */
MPFR_SAVE_EXPO_MARK (expo);
mpz_init (m);
e = mpfr_get_z_exp (m, x);
if ((negative = MPFR_IS_NEG(x)))
    mpz_neg (e, m);
e = e % (mpfr_exp_t) k;
if (r < b)
    r = k; /* now r = q (mod k) with 0 <= r < k */
/* x = (n^2)^r = 2^r where e-r is a multiple of k */
/* For rounding to nearest, we want the round bit to be in the root */
/* we now multiply e by 2^(r+k)*sh so that root(a,k) will give exactly n bits; we want k*\(n \pm 1)2^e = \text{size}_n + k*sh + r = k*n
1. e. sh = floor ((k*\text{size}_n + r)k)/k
if ((mpfr_exp_t) size_n + r > k) /* (mpfr_exp_t) n)
sh = 0; /* we already have too many bits */
Outline

Day 1: Mostly the upper layers & software

Day 2: Mostly the lower layers & hardware
The upper layers
What is a programming language?

A simple language system designed to facilitate the writing of computer programs

Collins English Dictionary
```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    printf(“Hello, world!\n”); return 0;
}
```

```java
public class Hello {
    public static void main(String argv[]) {
        System.out.println(“Hello, world!”);
    }
}
```

print “Hello, world!” print(“Hello, world!”)
What happens to code

**Compilation**
- Typically C, C++, Fortran, Pascal

**Interpretation**
- Typically Python, JavaScript, Smalltalk, Matlab, VBA

**Reccompilation**
- VMs, Rosetta, Dynamo(RIO)

**Transcompilation**
- E.g., C to HDL, ROSE
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf(“Hello, world!\n”);
    return 0;
}

Compilers
Ahead-of-time compilation
OS view

1. Write Source Codes
2. Preprocess
3. Compile
4. Link Edit
5. Load
6. Execute
Preprocessing

• The preprocessor modifies the source, in memory, as specified by preprocessor directives in the source

• Typical tasks:
  – Adding headers
  – Handling macros
  – Removing formatting, comments

• gcc -E
Compiler front- and back-ends

Front-end
- Language focus (Language-specific)
- Machine-independent

Back-end
- Language-independent
- Optimization
- Machine focus (Machine-specific)
  - Architectural analysis
  - Code generation
Ahead-of-time compilation
Compiler view

Source code
Lexical Analyzer
Tokenized code
Syntactic Analyzer
Parsed code
Semantic Analyzer
Qualified code
Code Generator
Object code
Optimizer
Final code

Front-end
Back-end
Intermediate representations
Source file

```c
int main () {
    int a,b,c,x,y,z;
    a = b+c;
    x = a+z+y+2;
}
```
Intermediate representations
AST (viz format)

digraph {
  title: "main"

  node [ label = "ENTRY", shape = circle ]
  node [ label = "EXIT", shape = circle ]

  edge [ source = "ENTRY", target = "0", style = solid, priority = 100 ]

  node [ label = "0", shape = ellipse ]
  node [ label = "0\nmodify_expr (11)\nreturn_expr (13)" ]

  edge [ source = "0", target = "EXIT", priority = 100, style = solid ]
}

Credit: cse.iitb.ac.in/~uday
07-Feb-18
Intermediate representations
GIMPLE

;; Function main (main)
main ()
{
    int z;
    int y;
    int x;
    int c;
    int b;
    int a;

    int D.1138;
    int D.1137;
    # BLOCK 0
    # PRED: ENTRY (fallthru)
    a = b + c;
    D.1137 = a + z;
    D.1138 = D.1137 + y;
    x = D.1138 + 2;
    return;
    # SUCC: EXIT
}
Simple RTL example

(set (reg:SI 140)
  (plus:SI (reg:SI 138)
    (reg:SI 139)))
Intermediate representations
RTL (>200 lines!)

(note 16 2 5 0 [bb 0] NOTE_INSN_BASIC_BLOCK)

(insn 5 16 6 0 (parallel [
(set (reg/f:SI 7 sp)
  (and:SI (reg/f:SI 7 sp)
    (const_int -16 [0xfffffffff0]))
  (clobber (reg:CC 17 flags))
]) -1 (nil)
(nil))

(insn 6 5 7 0 (set (reg:SI 61)
  (const_int 0 [0x0]) -1 (nil) (nil)) ......
Intermediate representations

ASM

main:

pushl %ebp
movl %esp, %ebp
subl $40, %esp
andl $-16, %esp
movl $0, %eax
addl $15, %eax
addl $15, %eax
shrl $4, %eax
sall $4, %eax
subl %eax, %esp

movl -16(%ebp), %eax
addl -20(%ebp), %eax
movl %eax, -24(%ebp)
movl -4(%ebp), %eax
movl -4(%ebp), %eax
addl -24(%ebp), %eax
addl -8(%ebp), %eax
addl $2, %eax
movl %eax, -12(%ebp)
leave
ret

Credit: cse.iitb.ac.in/~uday

07-Feb-18
Typical compiler optimizations

- Algebraic optimizations
- Dead code elimination
- Loop invariants
- Vectorization
- Register allocation optimization

...and many more
Advanced optimizations

LTO – Link-time optimization / IPO – Interprocedural optimization

- Analyzes calls program-wide
- E.g., variable sharing and modification, memory aliasing

FDO – Feedback-driven optimization / PGO – Profile-guided optimization

- Optimizes code based on runtime behavior
- Requires recompilation
Linking

- The linker joins object files to produce a final file
- Linkers can produce executable, library or object files
- Object files may be sourced from the same program or libraries
Compilers in practice

Compilation error

Parse error. Internet Explorer has a non-standard interpretation of trailing commas. Arrays will have the wrong length and objects will not parse at all.

function sum(first, second) {
  alert(sum(first, second));
}

deb(2,3), 4;
Be paranoid!
Java

```java
public class Hello {
    public static void main(String argv[]) {
        System.out.println("Hello, world!");
    }
}
```
Just-in-time compilation
a.k.a. “dynamic translation”

• A program can be compiled to an intermediate representation and left until runtime
• Compilation during execution
  – From source
  – Or from “bytecode”
• Sometimes seen as a combination of ahead-of-time compilation and interpretation
• JIT compilers can handle files, functions or code fragments
• JIT compilers implement heavy caching
Bytecode

• Source code is translated/compiled to an intermediate representation called “bytecode”
• Bytecode can be interpreted or run by a virtual machine
• Bytecode is not machine code
# A few Java bytecode ops
(operands not shown)

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Opcode (hex)</th>
<th>Stack change</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>dup</strong></td>
<td>59</td>
<td>value → value, value</td>
<td>duplicate the value on top of the stack</td>
</tr>
<tr>
<td><strong>idiv</strong></td>
<td>6c</td>
<td>value1, value2 → result</td>
<td>divide two integers</td>
</tr>
<tr>
<td><strong>ifeq</strong></td>
<td>99</td>
<td>value →</td>
<td>if value is 0, branch to instruction at branchoffset (branchbyte1 &lt;&lt; 8 + branchbyte2)</td>
</tr>
<tr>
<td><strong>isub</strong></td>
<td>64</td>
<td>value1, value2 → result</td>
<td>int subtract</td>
</tr>
<tr>
<td><strong>lstore_0</strong></td>
<td>3f</td>
<td>value →</td>
<td>store a long value in a local variable 0</td>
</tr>
</tbody>
</table>
outer:
for (int i = 2; i < 1000; i++) {
   for (int j = 2; j < i; j++) {
      if (i % j == 0)
         continue outer;
   }
   System.out.println (i);
}
Java bytecode dump example

0:  iconst_2
1:  istore_1
2:  iload_1
3:  sipush 1000
6:  if_icmpge 44
9:  iconst_2
10: istore_2
11: iload_2
12: iload_1
13: if_icmpge 31
16: iload_1
17: iload_2
18: irem
19: ifne 25
22:  goto  38
25:  iinc  2, 1
28:  goto  11
31:  getstatic   #84; // Field java/lang/System.out:Ljava/io/PrintStream;
34:  iload_1
35:  invokevirtual #85; // Method java/io/PrintStream.println:(I)V
38:  iinc  1, 1
41:  goto  2
44:  return
Java Virtual Machine (JVM)

- An abstract computing machine
  - Operates on primitives – int/fp values and references
  - Garbage collection
- There are various implementations
- JRE – Java Runtime Environment
  - Typically contains a JVM implementation and a class library
  - Most well-known is HotSpot (Oracle, ex-Sun)
- JDK – Java Development Kit
  - Includes a compiler in addition to the above
The TAQ
Trollingly Asked Questions
Is ICC faster than GCC?
Is C++ slower than C?
Why can’t C/C++ be interpreted or JIT-ed?
CLING

• https://root.cern.ch/cling
• “interactive compiler”
  – Command line prompt and JIT compiler
• Designed for rapid turnaround and prototyping
• Jupyter Notebook front-ends exist
CLING + Jupyter

Interpreting the C++ programming language

CLING has a broad support of the features of C++. You can define functions, classes, templates, etc...

Functions

```cpp
In [8]: double sqr(double a)
   { return a * a;
   }

In [ ]: double a = 2.5;
double asqr = sqr(a);
asqr
```

Classes

```cpp
In [ ]: class Foo
   {
   public:
   virtual ~Foo() {};
   ```
CLING + Jupyter

In [13]: fig.animation_duration = 1000;

In [16]: scatter.x = randn(size); \
scatter.y = randn(size);

In [ ]:
Recap: main types of code

Source code

```
template<class InputString, 
  bool unhexlify(const InputSt 
if (input.size() % 2 != 0) 
  return false; 
} 
output.resize(input.size()) 
int j = 0; 
auto unhex = []((char c) -> 
  return c >= '0' && c <= '9' ? 

(set (reg:SI 140) 
  (plus:SI (reg:SI 138) 
  (reg:SI 139)) 

```

Object code

```
<table>
<thead>
<tr>
<th>Push Literal</th>
<th>Push Literal</th>
<th>Push Literal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable</td>
<td>Variable</td>
<td>Variable</td>
</tr>
<tr>
<td>22</td>
<td>102</td>
<td>104</td>
</tr>
<tr>
<td>103</td>
<td>119</td>
<td>120</td>
</tr>
<tr>
<td>118</td>
<td>PopInt</td>
<td>PopInt</td>
</tr>
<tr>
<td></td>
<td>Variable</td>
<td>Variable</td>
</tr>
<tr>
<td></td>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>103</td>
<td>105</td>
</tr>
<tr>
<td></td>
<td>119</td>
<td>PopInt Temp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>118</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Push 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Push 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Return</td>
<td></td>
</tr>
</tbody>
</table>
```

Bytecode

```
<table>
<thead>
<tr>
<th>x</th>
<th>m</th>
<th>uu</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>.100</td>
<td>0</td>
<td>0111111 010 0</td>
<td></td>
</tr>
</tbody>
</table>
| t24q
|     | 0 | 111111 uu f |
|     | 0 | 101001 100 0 |
|     | 0 | regmd8 |
|     | 1 |    |
|     | 1 | 111001 1' |
```

Machine code

```
401c3b: 31 ff
401c3d: ba 05 0
401c42: e8 e1 f
401c47: 48 89 d
401c4a: 48 89 c
401c4d: bf 01 0
401c52: 31 c0
```

Microcode
Thank you

e-mail: an@tik.services

http://tik.services