DWARF Debugging Format

How the Compiler Tells Its Secrets to the Debugger

Michael J. Eager
eager@eagercon.com
How Program Development is Supposed to Work
Developer Has **Great Idea**
Translates Great Idea into C code
Compiler translates C code to machine language
Everything works!!
Real World Program Development

• Developer Has Great Idea

• Translates Great Idea into C code

• Compiler translates C to machine language
Something Unexpected Happens
Developer uses debugger to understand the translation from Great Idea to machine language
Many hours and many cups of coffee later translation error is fixed
Debugging

Great Idea $\implies$ C code $\implies$ machine language

- Two translation steps
- We look at the second translation to find problems in the first translation
What We Think the Compiler Does

- Reads clear and complete program source
- Linear translation from C code into machine language
- Follows programmer's directions to the letter
What Really Happens

• Compiler believes it knows better than the developer. Reorders and reorganizes the program to improve performance
  • If it isn't prohibited, it's permitted
  • If it isn't defined, compiler free to do anything
• Multi-step process of incremental optimization
• Each time a change is made, a little bit of information is lost
Goals of the Compiler

- Correctly interpret C (or other) language
  - Compare with language standard
  - Verify with test suite and regression tests
- Generate correct machine language
  - Defined by architecture manual
  - Verify with test suite
- Optimize code
  - Optimized result is the same as unoptimized code
  - Verify with test suite
- Generate debugging info
What the Debugger Knows

- Info from object file (executable, obj, library)
  - Symbol names and addresses
    - Global
    - Local (some)
- Info from processor
  - Memory contents
  - Register contents
- Info from system
  - Library locations
  - How to control programs
What DWARF Tells the Debugger

- Source files – name and path
- Names of functions, arguments, globals, locals
- Type descriptions
- Types of functions, variables, and parameters
- Block structure of program
- Mapping between source and object (line<=> address)
- Variable location (registers/memory)
- How to unwind stack
What DWARF Doesn't Tell

- Machine characteristics
  - Registers, address size, instructions
- OS characteristics
- ABI
  - Calling conventions
- Program flow
- Semantics
DWARF History

- Developed at AT&T as part of Unix SVR4
- PL SIG (Programming Languages SIG) of Unix International, Inc. formed in 1988
- DWARF version 1 (standard published 1992)
  - Compatible with AT&T SVR4 DWARF format
- DWARF version 2 (draft standard released 1993)
  - Not compatible with DWARF version 1
  - Broader functionality
  - More compact representation
- DWARF Committee reconstituted October, 1999
- DWARF version 3 (standard published 2005)
  - Compatible with DWARF version 2
- DWARF version 4 (standard published 2010)
DWARF Philosophy

- Permissive standard
  - Describes what various DWARF constructs mean
  - Does not mandate generation of specific constructs
- Extensible
  - Supports user extensions
  - Allows novel uses of existing attributes
- Upward compatible
  - Consumers (i.e. debuggers) can read later versions
  - Skip over unknown DIEs
DWARF Goals

• Permit accurate and complete description of source to object translation
  • Whether a particular compiler generates good or poor DWARF is a Quality of Implementation issue
• Compact data representation
• Efficient generation
• Open standard, transparent process
Languages and Processors

• Block structured procedural languages
  - C
  - C++
  - Cobol
  - Java
  - Ada
  - Fortran
  - Modula
  - Pascal

• Von Neuman or Harvard architecture
  - x86
  - IA32
  - ARM
  - IA64
  - PowerPC
  - MIPS
Basic Concepts

• DWARF can be used in any object file
  • Most commonly associated with ELF
• Multiple data sections
  • DWARF sections start with .debug_
    - .debug_info – Program organization
      • Functions & Variables
    - .debug_line – Line <=> address mapping
    - Several other sections
      • Compression – strings, abbreviations, types
      • Other info – call frames, indexes – address and name
Basic Data Structure

- Debugging Information Entry (DIE)
  - Each DIE has a TAG which identifies purpose
    - DW_TAG_compile_unit – Describe a compilation unit
    - DW_TAG_subprogram – Describe a subroutine
    - DW_TAG_variable – Describe a variable
    - DW_TAG_pointer_type – Describe various types
    - DW_TAG_formal_parameter – Describe arguments
Basic Data Structure

- Each DIE has one or more attribute/value pairs
- Each attribute has a name
  - Describes meaning of attribute
  - Value specified for each attribute
  - Data format specified in attribute encoding
- Examples
  - `DW_AT_name` – Name of object DIE describes
  - `DW_AT_location` – Source location of object
  - `DW_AT_low_pc` – Start address of object
  - `DW_AT_high_pc` – End address of object
  - `DW_AT_type` – Pointer to DIE describing type
DWARF Info Tree Structure

• Match block structure of source program
• Each DIE has zero or more sibling DIEs
• Each DIE has zero or more children
• Each Compilation is represented by a Compilation Unit DIE
  • Everything is a child of the Comp Unit DIE
DWARF Info Tree Structure

Compilation Unit
main.c
/home/example/src

Child

Subprogram
main
return int
extern

Child

Sibling

Sibling

Variable
argc
formal
signed int
frame+0x4

Variable
argv
formal
const char *
frame+0x8

Sibling

Subprogram
foo
return char *

Compilation Unit
foo.c
/home/example/src
Compile Unit DIE

- Describe compilation
- Source location
- Compilation directory
- Producer info
- Programming language
- Low and high PC range
- Pointers to other data
  - Line number info
  - Macro info
- Children DIEs describe the program
**Compile Unit DIE**

b3: DW_TAG_compile_unit

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DW_AT_producer</td>
<td>GNU C 4.6.1 20110627</td>
</tr>
<tr>
<td>DW_AT_language</td>
<td>1 (ANSI C)</td>
</tr>
<tr>
<td>DW_AT_name</td>
<td>bzip2.c</td>
</tr>
<tr>
<td>DW_AT_comp_dir</td>
<td>/ext/yocto/.../bzip2-1.0.6</td>
</tr>
<tr>
<td>DW_AT_low_pc</td>
<td>0x0</td>
</tr>
<tr>
<td>DW_AT_entry_pc</td>
<td>0x0</td>
</tr>
<tr>
<td>DW_AT_ranges</td>
<td>0x260</td>
</tr>
<tr>
<td>DW_AT_stmt_list</td>
<td>0x82</td>
</tr>
</tbody>
</table>
Subroutine DIE

- **DW_TAG_subprogram**
  - Describe subroutine, function, inlined subroutine, entry point, declaration vs. definition
  - Subroutine name and source location
  - Visibility – whether it is external
  - Reference to return type DIE
  - Low and high PC
  - Prototyped flag
- “Owns” children DIEs: arguments, variables, types, and blocks within subroutine
1ba2: DW_TAG_subprogram

  DW_AT_external : 1
  DW_AT_name : main
  DW_AT_decl_file : 1
  DW_AT_decl_line : 1776
  DW_AT_prototyped : 1
  DW_AT_type : <0x683>
  DW_AT_low_pc : 0x80491e0
  DW_AT_high_pc : 0x8049d90
  DW_AT_frame_base : 0x22e3 (location list)
  DW_AT_sibling : <0x212a>
Variable DIE

• Describe data object
  • Variable name
  • Reference to type DIE
  • Source location
  • Declaration vs. definition
  • Run time location
  • Default value
  • Constant value
Variable DIE

1c28: DW_TAG_variable
    DW_AT_name : decode
    DW_AT_decl_file : 1
    DW_AT_decl_line : 1782
    DW_AT_type : <0x657>
    DW_AT_location : 0x24ed (location list)
...

213b: 71 (DW_TAG_variable)
    DW_AT_name : stdin
    DW_AT_decl_file : 5
    DW_AT_decl_line : 165
    DW_AT_type : <0x465>
    DW_AT_external : 1
    DW_AT_declaration : 1
Base Type DIE

- Describe data type that is directly implemented by machine hardware
- Name of type
  - Examples: int, long, unsigned char, etc.
- Encoding
  - Example: address, boolean, signed, float, decimal
- Size
  - Size in bytes or bits needed to hold value
  - Offset within storage unit
## Base Type DIE

<table>
<thead>
<tr>
<th>DIE</th>
<th>Tag</th>
<th>Byte Size</th>
<th>Encoding</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>d4</td>
<td>DW_TAG_base_type</td>
<td>2</td>
<td>7 (unsigned)</td>
<td>short unsigned int</td>
</tr>
<tr>
<td>db</td>
<td>DW_TAG_base_type</td>
<td>4</td>
<td>7 (unsigned)</td>
<td>unsigned int</td>
</tr>
<tr>
<td>6d</td>
<td>DW_TAG_base_type</td>
<td>1</td>
<td>8 (unsigned char)</td>
<td>unsigned char</td>
</tr>
<tr>
<td>f7</td>
<td>DW_TAG_base_type</td>
<td>4</td>
<td>5 (signed)</td>
<td>int</td>
</tr>
</tbody>
</table>
Composite Type DIEs

- Type DIE constructed from references to other type DIEs, either Base Type or Composite Type
- Const_type, volatile_type
  - Represent “const” or “volatile” qualifier
- Pointer_type
  - Represent pointer to qualifier (“*”)
- Typedef
- Eventually reach Base Type
Composite Type DIEs

260: DW_TAG_structure_type
   DW_AT_name : _IO_FILE
   DW_AT_byte_size : 148
   DW_AT_decl_file : 6
   DW_AT_decl_line : 271
   DW_AT_sibling : <0x421>

26d: DW_TAG_member
   DW_AT_name : _flags
   DW_AT_decl_file : 6
   DW_AT_decl_line : 272
   DW_AT_type : <0x190>
   DW_AT_data_member_location: (DW_OP_plus_uconst: 0)

...
const unsigned char * volatile p;

A volatile pointer to a constant character.

This is encoded in DWARF as:

\[
\text{DW\_TAG\_variable} (p) \rightarrow \\
\text{DW\_TAG\_volatile\_type} \rightarrow \\
\text{DW\_TAG\_pointer\_type} \rightarrow \\
\text{DW\_TAG\_const\_type} \rightarrow \\
\text{DW\_TAG\_base\_type} \text{ (unsigned char)}
\]
Data Structures

- **DW_TAG_struct_type, DW_TAG_class_type, DW_TAG_union_type, DW_TAG_interface_type**
  - Define structure, class, union, Java interface
  - DIE “owns” members of the struct/class/union/interface
  - **DW_TAG_member**
    - Similar to a variable definition
    - Instead of memory location, has offset from start of object

- **DW_TAG_array_type**
  - Define array of same type object
  - Index is a subrange. In C, [0..n).

---

**Linux Collaboration Summit**
April 5, 2012

**DWARF Debugging Format**
Michael J. Eager
Locating Data

- **DW_AT_location** – location description
  - Single location description – fixed lifetime
    - Simple location – contiguous location (reg or memory)
    - Composite location – data split into pieces
    - Omitted – “variable optimized away”
  - Multiple location description – Location lists
    - Reference to `.debug_loc`
    - Define where data is located for specific PC ranges
    - Object can change location over its lifetime

- **DWARF expressions**
  - Complete stack-oriented expression evaluation
Locating Code

- **DW_AT_low_pc** – starting or only address
- **DW_AT_high_pc** – ending address
- **DW_AT_ranges** – non-contiguous range
  - Reference to `.debug_ranges`
  - Pairs of (beginning, ending) offset from base address
- Base Address
  - Default to start of compilation unit
  - May be explicitly specified
Mapping Address to Source

- Needed to set breakpoints, identify fault location, step through source
- `.debug_line` section
- Conceptual contents
  - One row for each code memory address
  - Source file name, line number, column
  - Flag beginning of statement
  - Flag beginning of basic block
  - Flag end of prologue, start of epilogue
  - Instruction set (e.g., ARM vs Thumb)
- Problem – unencoded table would be huge
Compressing Line Information

- Finite State Machine generates line info table
- Line Number Program
  - Operations drive FSM to generate next row
  - Duplicate rows are eliminated
  - Each value described as register, copied to next row unless changed
- Example ops
  - Add integer to source line number
  - Set statement, block, prologue, epilogue flag
  - Advance PC
Speeding Up Debugging

- `.debug_pubnames`
  - Names of global objects and functions
  - Reference to DIE defining object or function

- `.debug_pubtypes`
  - Names of types
  - Reference to DIE describing type

- `.debug_aranges`
  - Address start and length
  - Reference to compilation unit
Call Frame Information

- Describe details of function call
  - Locate previous frame
  - Locate saved register values
- Permit unwinding/walking stack
- CIE – Common Information Entry
- FDE – Frame Description Entry
  - Finite State Machine indexed by PC address
- Variant (.eh_frame) used to implement C++ exception handling
Compressed DWARF

- Uncompressed TAG/Attribute/Value huge
  - Major impetus for DWARF 1 to DWARF 2 migration
- Multiple approaches to compression
  - Data encoding – uleb, sleb
  - Indirection – references to other tables
  - Abbreviation table
  - Implicit sibling pointers
- Separate data for duplicate elimination
GCC Debug Options

- **-g**
  - Generate default debug info (DWARF)

- **-g3**
  - Generate debug info including macro descriptions

- **-ggdb**
  - Generate debug info for gdb (most expressive)

- **-gdwarf[234]**
  - Generate DWARF 2, 3, 4 debug info
  - May use some extensions from later versions
  - DWARF 4 requires gdb-7.0 for best results

- **-gstrict-dwarf**
  - Disallow extensions from later standard versions
Printing DWARF with Readelf

- `readelf -w` 
  - Dump all DWARF data

- `readelf -w[lLiaprmfFsorT]`
  - Print selected DWARF data
    - raw line table, decoded line table, info, abbrev, pubnames, aranges, macro, raw frames, frames-interp, str, location, ranges, public types
DWARF version 4

- Released June 10, 2010
- Extensive review and update of documentation
- Support for VLIW architectures (IA64)
- Separate type units – improved compression
- Improved language support
  - Fortran – identify main subprogram
  - C++ -- rvalue references, constant exprs, template aliases, template parameters, strong enum types
  - Generalize packed array descriptions
  - Support profile-based optimizations
DWARF version 5

- Anticipated release date late 2013
- Support C++11 features: atomic
- Separate debug data from object files
- Improved macro description
- Improve debug of optimized code
  - Optimized variables
- Improved debugger accelerator data
- Restructure documentation
DWARF Committee

- Committee website: dwarfstd.org
- Independent, no membership fees
- Open standard available without charge
- Broad based
  - Companies represented:
    - Apple
    - ARM
    - Concurrent Computer
    - Eager Consulting
    - Google
    - HP
    - IBM
    - Intel
    - RedHat
    - Rogue Wave
Questions/Answers

- Michael Eager – eager@eagercon.com
- DWARF Website – dwarfstd.org
  - Submit question/suggestion about standard
    - dwarfstd.org/Comment.php
- DWARF wiki – wiki.dwarfstd.org
- DWARF Discussion List
  - dwarf-discuss@lists.dwarfstd.org