GCC Introductory Tutorial

Creating a compiler for your new chip
GNU Tool Chain Components

Source code
Ada
C/C++
Fortran
OpenMP
Rust
...

GCC

GAS

binutils

CGEN

GDB

Object code

compiler libraries
libgcc
libstdc++v3

libc/libm
Newlib
Glibc
Musl

RV32GC_Xcv*
RV64GC_Xcv*
GNU Tool Chain Components

- Source code
  - Ada
  - C/C++
  - Fortran
  - OpenMP
  - Rust
  - ...

- GCC

- GAS

- binutils

- CGEN

- Object code

- compiler libraries
  - libgcc
  - libstdc++v3

- libc/libm
  - Newlib
  - Glibc
  - Musl

- GDB

- RV32GC_Xcv*
- RV64GC_Xcv*
Scope

• How to get a new GCC back end up and running
  – sources of information
  – adding the key elements for a new architecture
  – debugging

• Outcomes
  – you know where to get started
Sources of Information
Textbooks: Introductory
Textbook: Comprehensive
The Real World: GCC Internals Manual

This file documents the internals of the GNU compiler.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation, with the Invariant Sections being "Funding Free Software", the Front-Cover Text being (a) (see below), and with the Back-Cover Text being (b) (see below). A copy of the license is included in the section entitled "GNU Free Documentation License".

(a) The FSF's Front-Cover Text is:

A GNU Manual

(b) The FSF's Back-Cover Text is:

You have freedom to copy and modify this GNU Manual, like GNU software. Copies published by the Free Software Foundation raise funds for GNU development.

Short Table of Contents

1 Contributing to GCC Development
2 GCC and Portability
3 Interfacing to GCC Output
4 The GCC low-level runtime library
5 Language Front Ends in GCC
6 Source Tree Structure and Build System
7 Tools
8 Option specification files
9 Passes and Files of the Compiler
10 Error messages and normal invariants
11 GENERIC
12 CIMPHE
13 Analysis and Optimization of CIMPHE in GCC
14 RTL Representation
15 Control-Flow Graph
16 Analysis and Representation of Loops
17 Machine Description
18 Target Description Macros and Functions
19 Host Configuration
20 Miscellaneous Programs
21 Collected
22 Standard Header File Directories
23 Memory Management and Type Information
24 Plugins
25 Link Time Optimization
26 Stubs and Symply
27 Static Analyzer
28 Debugging C Code
Funding Free Software
The GNU Project and GNU/Linux
GNU General Public License
GNU Free Documentation License
Contributors to GCC
Option Index
Concept Index
Our New Chip
The VAM Architecture

Main memory (byte addressed)

16 general-purpose registers
- Program counter
- Status flag

ALU

Output port (8 bit)

HALT
NOP
TRAP
<arith> Rx, Ry
<logic> Rx, Ry
<shift> Rx, offset
STI rx, offset(Ry)
LDI offset(Rx), Ry

LDA offset(Rx), Ry
LDR Rx, Ry
B<flag> offset
BN<flag> offset
BRA offset
BAL Rx, Ry
The VAM Architecture

- Byte stream architecture with 32-bit registers
  - R0 is tied to zero
- Byte addressed memory with 32-bit PC
- All arithmetic is 2 address and signed, ADD, SUB, MUL, DIV
- All logical ops are 2 address: AND, OR, XOR, LSL, LSR, ASR
- All shift ops use offset as shift constant: LSL, LSR, ASR
- Status flags
  - Z set if arithmetic result or load is zero
  - N set if arithmetic result or load is negative
  - C set if arithmetic result is unsigned overflow
  - V set if arithmetic result is signed overflow
Getting Started
Clone GCC

```bash
git clone git://gcc.gnu.org/git/gcc.git
```

There is also a mirror on GitHub
Map of the gcc directory

- gcc/
  - gcc/  
  - config/  
    - riscv/  
    - avr/  
    - i386/  
      - (lots of archs here)
  
- lib*/
  - libstdc++-v3/
  - libgcc/
Map of the target configuration directory

- gcc/gcc/config/riscv
  - riscv.h: all the RISC-V specific stuff
    - target-specific definitions, e.g. what registers are available, byte/bit endianness, etc.
  - riscv.c: target-specific algorithms to tailor code generation specifically for this architecture
  - riscv.md: the “Machine Description” for RISC-V
  - riscv.opt: target-specific flags and options for RISC-V
Configuring GCC
for Your New Chip
Configuring: config.sub

• Generic across all GNU projects in its own repo
  – git.savannah.gnu.org/cgit/config.git/plain/config.sub
  – update by sending patch to config-patches@gnu.org

• Automatically copied from there to the top level of GCC hierarchy

• For now, we’ll do a simplistic edit of the GCC file for VAM...

*)

# Recognize the canonical CPU types that are allowed with any
# company name.
case $cpu in
  1750a | 580 |
... vam \
| vax \
| visium \
...
Configuring: gcc/config.gcc

- GCC specific configuration within gcc folder
- Can specify any specific files to be included for your target
  - set specific parameters for each category of file.
  - will automatically assume `{target}.{c,h,md,org}`
- For VAM we will just specify we also want the standard bare metal header

```plaintext
...;

vam-*-elf)
  tm_file="elfos.h ${tm_file}"
;;
vax-*-linux*)
...;
```
Running configure

$ mkdir bd  # Peer directory of the main GCC repo

$ ../gcc/configure --target vam-unknown-elf \
   --prefix=/opt/vam --without-headers \ 
   --enable-languages=c -disable-bootstrap

- This is a stage 1 build, we just want a plain C compiler
- There are many more options that can be used with GCC
  - mostly to disable more features
Building your compiler

$ make all-gcc

(lots happens)

...-

make[1]: *** No rule to make target '..../gcc/gcc/config/vam/vam.md', needed by 's-mddeps'. Stop.

make[1]: Leaving directory '/home/jeremy/gittrees/gcc-vam/bd/gcc'

make: *** [Makefile:4674: all-gcc] Error 2

$
Adding Implementation Detail:
The Header File
Adding the Missing Files

```
$ pushd ../gcc/gcc/config
$ mkdir vam
$ touch vam/vam.{cc,h,md,opt}
$ popd
$ make all-gcc
...
(lots more happens)
...
was not declared in this scope; did you mean 'FIRST_VIRTUAL_REGISTER'?
  322 |            return (*regno < FIRST_PSEUDO_REGISTER);
      | ^~~~~~~~~~~~~~~~~~~~~FIRST_VIRTUAL_REGISTER
...
```
vam.h: Predefined macros

/* Names to predefine in the preprocessor for this target machine. */
#define TARGET_CPU_CPP_BUILTINS() do {
    builtin_define ("__VAM__");
    builtin_define ("__vam__");
    builtin_assert ("cpu=vam");
    builtin_assert ("machine=vam");
} while (0)
What goes in the Header File?

- See **Target Description Macros and Functions**
- Easy approach: copy an existing architecture
  - **or1k** is a good, simple one
- Associated implementation code goes in **vam.cc**
- Contents
  - data storage, data types
  - register model
  - ABI implementation
/* Storage layout. */
#define DEFAULT_SIGNED_CHAR 1
#define BITS_BIG_ENDIAN 0
#define BYTES_BIG_ENDIAN 1
#define WORDS_BIG_ENDIAN 1
#define BITS_PER_WORD 32
#define UNITS_PER_WORD 4
#define POINTER_SIZE 32
#define BIGGEST_ALIGNMENT 32
#define STRICT_ALIGNMENT 1
#define FUNCTION_BOUNDARY 32
#define PARM_BOUNDARY 32
#define STACK_BOUNDARY 32
#define PREFERRED_STACK_BOUNDARY 32
#define MAX_FIXED_MODE_SIZE 64
/* Layout of source language data types. */
#define INT_TYPE_SIZE 32
#define SHORT_TYPE_SIZE 16
#define LONG_TYPE_SIZE 32
#define LONG_LONG_TYPE_SIZE 64
#define FLOAT_TYPE_SIZE 32
#define DOUBLE_TYPE_SIZE 64
#define LONG_DOUBLE_TYPE_SIZE 64
#define WCHAR_TYPE_SIZE 32
#undef SIZE_TYPE
#define SIZE_TYPE "unsigned int"
#undef PTRDIFF_TYPE
#define PTRDIFF_TYPE "int"
#undef WCHAR_TYPE
#define WCHAR_TYPE "unsigned int"
/* In VAM there are 32 general purpose registers with the following designations:

r0    always 0
r1    stack pointer
r2    frame pointer (optional)
r3    function call return link address
r4    arg 0 / function return value (low 32 bits)
r5    arg 1 / function return value (upper 32 bits)
r6    arg 2
r7    arg 3
r8    arg 4
r9    arg 5
r10-r18 callee saved registers
r19-r31 scratch registers

In addition we have
r32   Status register */
#define FIRST_PSEUDO_REGISTER 33

#define REGISTER_NAMES { \\
    "r0", "r1", "r2", "r3", "r4", "r5", "r6", "r7", \\
    "r8", "r9", "r10", "r11", "r12", "r13", "r14", "r15", \\
    "r16", "r17", "r18", "r19", "r20", "r21", "r22", "r23", \\
    "r24", "r24", "r26", "r27", "r28", "r29", "r30", "r31", \\
    "sr" }

#define FIXED_REGISTERS \\
{ 1, 1, 0, 0, 0, 0, 0, 0, \\
  0, 0, 0, 0, 0, 0, 0, 0, \\
  0, 0, 0, 0, 0, 0, 0, 0, \\
  0, 0, 0, 0, 0, 0, 0, 0, \\
  1 }
```c
#define REG_ALLOC_ORDER {
  10, 11, 12, 13, 14, 15, 16, 17,
  18,
  4, 5,
  6, 7, 8, 9,
  19, 20, 21, 22, 23, 24, 25, 26,
  27, 28, 29, 30,
  2,
  3,
  0,
  1,
  32
}
/* callee saved */
/* non-saved return values */
/* non-saved argument regs */
/* caller saved */
/* saved hard frame pointer */
/* saved return address */
/* fixed zero reg */
/* fixed stack pointer */
/* fixed status reg */
```
enum reg_class
{
    NO_REGS,
    GENERAL_REGS,
    FLAG_REGS,
    ALL_REGS,
    LIM_REG_CLASSES
};

#define N_REG_CLASSES (int) LIM_REG_CLASSES

#define REG_CLASS_NAMES { /*
    "NO_REGS",
    "GENERAL_REGS",
    "FLAG_REGS",
    "ALL_REGS"
}
vam.h: Register classes (2)

#define REG_CLASS_CONTENTS
{
  { 0x00000000, 0x00000000 }, \n  { 0xffffffff, 0x00000000 }, \n  { 0x00000000, 0x00000001 }, \n  { 0xffffffff, 0x00000001 }
}

/* A C expression whose value is a register class containing hard
register REGNO. In general there is more that one such class;
choose a class which is "minimal", meaning that no smaller class
also contains the register. */
#define REGNO_REG_CLASS(REGNO) ((REGNO) == SR_REGNUM ? FLAG_REGS : GENERAL_REGS)
With vam.h

$ make all-gcc
...
(even more happens)
...
../../../../gcc/gcc/config/vam/vam.h:228:30: error: ‘SP_REGNUM’ was not declared in this scope
228 | #define STACK_POINTER_REGNUM SP_REGNUM
    | ^~~~~~~~~
../../../../gcc/gcc/c-family/c-cppbuiltin.cc:1569:38: note: in expansion of macro ‘STACK_POINTER_REGNUM’
1569 | STACK_POINTER_REGNUM);
    | ^~~~~~~~~~~~~~~~~~~~
make[2]: *** [Makefile:1197: c-family/c-cppbuiltin.o] Error 1
...
Adding Implementation Detail: The Machine Description
Code Generation in GCC

- Made generic
- GCC is a pattern matching compilers
  - match standard patterns in RTL
  - give template of how they are to be generated
  - this is the Machine Description
- A big part of optimization is replacing patterns
GCC Intermediate Representations

- **GENERIC**: high-level language specific TREE
  - starting point for middle-end transformations

- **GIMPLE**: high-level language independent TREE
  - subset of GENERIC
  - tree transformations (e.g. vectorization, loop unrolling)
  - really a flattened tree

- **RTL**: low-level three address code
  - three address code transformations
GCC Type System: RTL Modes

- **qi/QI**: Quarter-integer (8-bits)
- **hi/HI**: Half-integer (16-bits)
- **si/SI**: Single-integer (32-bits)
- **di/DI**: Double-integer (64-bits)
- **ti/TI**: Tetra-integer (128-bits)
- **sf/SF**: Single-float (32-bits)
- **df/DF**: Double-float (64-bits)
- **u-/-U--**: Unsigned variant of above (where applicable)
Lowering GIMPLE to RTL

• Provide a set of standard patterns, e.g.
  - addq13 – “Add 2 quarter-integer (8-bit) values”
  - subd13 – “Subtract 2 double-integer (64-bit) values”
  - mulsi3 – “Multiply 2 single-integer (32-bit) values”

• Many of the basic low-level RTL patterns must be defined
  - e.g. addition and multiplication

• Many of the predefined RTL pattern names can be omitted
  - e.g. various atomic and vector-op named patterns
Machine Descriptions

• written in a Scheme-like language
  - reusing RTL expressions, machine modes and syntax.
• parsed at compile time
  - generate C instruction selectors and pattern matchers
• compiled, and linked into the GCC executable
  - invoked by GCC when the compiler is run.
• See Machine Descriptions
  - or copy an existing machine description
Machine Description Example: \texttt{riscv.md} (1)

\begin{verbatim}
(define_insn "addsi3"
  [(set (match_operand:SI 0 "register_operand" ",=r,r")
      (plus:SI (match_operand:SI 1 "register_operand" ",r,r")
               (match_operand:SI 2 "arith_operand" ",r,I")))]
 ""
  { return TARGET_64BIT ? "addw\t%0,%1,%2" : "add\t%0,%1,%2" ; }
 [(set_attr "type" "arith")
   (set_attr "mode" "SI")])
\end{verbatim}

- \texttt{define_insn}: semantics of an insn this architecture supports
- name can be anything, but it matters
- \texttt{addsi3}: one of the predefined names
  - GCC can lower RTL using this predefined name
Machine Description Example: \texttt{riscv.md} (2)

\[
\begin{align*}
&\text{(set (match_operand:SI 0 "register_operand" ",=r,r")}
&\quad\text{(plus:SI (match_operand:SI 1 "register_operand" ",r,r")}
&\quad\text{(match_operand:SI 2 "arith_operand" ",r,I")))}
\end{align*}
\]

- \textbf{match\_operand:SI 0}
  - RTL expression is a placeholder for any operand
  - but it must be of machine mode \texttt{SI}
- \textbf{the 0 is simply an identifier for this operand slot}
  - used later in template for assembly output
Machine Description Example: riscv.md (3)

```
[(set (match_operand:SI 0 "register_operand" ":=r,r")
   (plus:SI (match_operand:SI 1 "register_operand" ":r,r")
     (match_operand:SI 2 "arith_operand" ":r,I")))]
```

- **register_operand**
  - the predicate – is a given candidate fit for this operand slot
  - an ALLOW/DENY gating function
  - **register_operand** is a predefined standard GCC predicate
  - you can write and use your own C predicates here
Machine Description Example: riscv.md (4)

```
[(set (match_operand:SI 0 "register_operand" "=r,r")
  (plus:SI (match_operand:SI 1 "register_operand" "r,r")
   (match_operand:SI 2 "arith_operand" "r,I")))]
```

- **=r,r**
  - a comma-separated list of operand constraints
  - unlike predicates if an operand constraint fails, the register allocator will try to shuffle the RTL around to try and satisfy the constraints
  - express the exact semantics of your instruction operands
Machine Description Example: riscv.md (5)

```
[(set (match_operand:SI 0 "register_operand" ";=r,r")
  (plus:SI (match_operand:SI 1 "register_operand" "r,r")
    (match_operand:SI 2 "arith_operand" "r,I"))))]
```

- List of possible constraints (alternatives)
  - = all the constraints in the list are for writing to
  - r constrained to be a register
  - I constrained to be an immediate constant

- Each operand has the same number of constraints
Machine Description Example: riscv.md (6)

```
(define_insn "addsi3"
  [(set (match_operand:SI 0 "register_operand" ";r;;;;r")
    (plus:SI (match_operand:SI 1 "register_operand" ";r;;;;r")
      (match_operand:SI 2 "arith_operand" "r,I")))]
  ""
  { return TARGET_64BIT ? "addw\%0,\%1,\%2" : "add\%0,\%1,\%2"; }
  [(set_attr "type" "arith")
   (set_attr "mode" "SI")])

• Next operand "" is a global predicate
• Typically used to disable pattern based on flag passed to the compiler
  − as a C expression
• "" means true, the pattern is always used
Machine Description Example: riscv.md (7)

```c
{ return TARGET_64BIT ? "addw\t%0,%1,%2" : "add\t%0,%1,%2"; }
```

- The code generation template
- In this case it is a C expression
  - there are many other ways it can be specified
  - can have multiple alternatives corresponding to constraints
- the % elements refer to the operand numbers above
Machine Description Example: \texttt{riscv.md} (8)

\[
[(\text{set\_attr } "\text{type}" "\text{arith}"), \\
(\text{set\_attr } "\text{mode}" "\text{SI}"))]
\]

- Attributes associated with the pattern
  - mode is mandatory – the space needed for the assembler
  - everything else is optional
    - can be queried by code generation functions
vam.md (1)

;; Register numbers
(define_constants
   [(SP_REGNUM       1)
    (HFP_REGNUM      2)
    (LR_REGNUM       3)
    (RV_REGNUM       4)
    (SR_REGNUM    32)]
)

;; nop instruction

(define_insn "nop"
   [(const_int 0)]
   ""
   "NOP")
vam.md (2)

;;; Arithmetic instructions
;;;
(define_insn "addsi3"
  [(set (match_operand:SI 0 "register_operand" "=r")
      (plus:SI
       (match_operand:SI 1 "register_operand" "r")
       (match_operand:SI 2 "register_operand" "0")))]
  ""
  "ADD\t%1, %0"
)

(define_insn "subsi3"
  [(set (match_operand:SI 0 "register_operand" "=r")
      (minus:SI
       (match_operand:SI 1 "register_operand" "r")
       (match_operand:SI 2 "register_operand" "0")))]
  ""
  "SUB\t%1, %0"
Further Reading on Machine Descriptions

Internals Manual

- **Standard Names**
  - standard MD patterns
- **Machine Description**
  - machine descriptions
- **Output Statement**
  - assembly language templates

Useful files

- `gcc/machmode.def`
  - standard machine modes
- `gcc/config/or1k/or1k.md`
  - simple(ish) example
Adding Implementation Detail:
Target Specific Options
vam.opt

• See Option Specification Files
  - for target specific options
  - VAM: -mhard-div, -msoft-div, -mhard-mul, -msoft-mul

mhard-div
Target RejectNegative InverseMask(SOFT_DIV)
Enable generation of the hardware divide (DIV) instruction. This is the default; use -msoft-div to override.

msoft-div
Target RejectNegative Mask(SOFT_DIV)
Enable generation of binaries which use functions from libgcc to perform divide operations. The default is -mhard-div.
Putting is All Together
The Full Compilation: First Attempt

$ make all-gcc

...(almost everything happens)

...build/genemit ../../gcc/gcc/common.md ../../gcc/gcc/config/vam/vam.md
insn-conditions.md
/bin/bash ../../gcc/gcc/../move-if-change tmp-emit-1.cc insn-emit-1.cc; /bin/bash ../../gcc/gcc/../move-if-change tmp-emit-2.cc insn-emit-2.cc; /bin/bash ../../gcc/gcc/../move-if-change tmp-emit-3.cc insn-emit-3.cc; /bin/bash ../../gcc/gcc/../move-if-change tmp-emit-4.cc insn-emit-4.cc; /bin/bash ../../gcc/gcc/../move-if-change tmp-emit-5.cc ...


The Full Compilation: First Attempt

emit-5.cc insn-emit-5.cc; /bin/bash ../gcc/gcc/..move-if-change
tmp-emit-6.cc insn-emit-6.cc; /bin/bash ../gcc/gcc/..move-if-change
tmp-emit-7.cc insn-emit-7.cc; /bin/bash ../gcc/gcc/..move-if-change
tmp-emit-8.cc insn-emit-8.cc; /bin/bash
../gcc/gcc/..move-if-change tmp-emit-9.cc insn-emit-9.cc;
/bin/bash ../gcc/gcc/..move-if-change tmp-emit-10.cc insn-emit-10.cc;
mv: cannot stat 'tmp-emit-10.cc': No such file or directory
make[1]: *** [Makefile:2612: s-tmp-emit] Error 1
The Full Compilation: Second Attempt

• I asked for help
• Is it a bug in `genemit`?
• Try a smaller number of partitions

```
$ ../gcc/configure --target vam-unknown-elf \
  --prefix=/opt/vam --without-headers \ 
  --enable-languages=c --disable-bootstrap \ 
  --with-insnemit-partitions=5
```
The Full Compilation: Second Attempt

$ make all-gcc
...
(everything happens)
...
/home/jeremy/gittrees/gcc-vam/bd/.gcc/xgcc
-B/home/jeremy/gittrees/gcc-vam/bd/.gcc/ -xc -nostdinc /dev/null -S
-o /dev/null -fself-test=../../gcc/gcc/testsuite/selftests
cc1: internal compiler error: in default_legitimate_address_p, at
targhooks.cc:112
0x6c3c61 default_legitimate_address_p(machine_mode, rtx_def*, bool,
code_helper)
    ..../gcc/gcc/targhooks.cc:112
0xaa1053 validize_mem(rtx_def*)
    ..../gcc/gcc/explow.cc:552
0x9af669 init_set_costs()
The Full Compilation: Second Attempt

$ make all-gcc
...
(everything happens)
...
  ../gcc/gcc/cfgloopanal.cc:441
  0xeade63 backend_init_target
  ../gcc/gcc/toplev.cc:1746
  0xeade63 initialize_rtl()
  ../gcc/gcc/toplev.cc:1811
  0xe2a096 read_rtl_function_body(char const*)
  ../gcc/gcc/read-rtl-function.cc:1632
  0xe6cf95 selftest::rtl_dump_test::rtl_dump_test(selftest::location const&, char*)
  ../gcc/gcc/selftest-rtl.cc:92
  0xe2670d test_loading_dump_fragment_1
The Full Compilation: Second Attempt

$ make all-gcc
...
(everything happens)
...
  ../gcc/gcc/read-rtl-function.cc:1761 0xe2a40e selftest::read_rtl_function_cc_tests()
  ../gcc/gcc/read-rtl-function.cc:2212 0x191e22c selftest::run_tests()
  ../gcc/gcc/selftest-run-tests.cc:90 0xeae22d toplev::run_self_tests()
  ../gcc/gcc/toplev.cc:2211
Please submit a full bug report, with preprocessed source (by using -freport-bug).
Please include the complete backtrace with any bug report.
See <https://gcc.gnu.org/bugs/> for instructions.
The Full Compilation: Second Attempt

$ make all-gcc
...
(everything happens)
...

make[1]: *** [../../gcc/gcc/c/Make-lang.in:153: s-selftest-c] Error 1
rm gfdl.pod gcc.pod gcov-dump.pod gcov-tool.pod fsf-funding.pod
gpl.pod cpp.pod gcov.pod lto-dump.pod
make[1]: Leaving directory '/home/jeremy/gittrees/gcc-vam/bd/gcc'
make: *** [Makefile:4674: all-gcc] Error 2
$
Dumping GCC internal state

• Compile a simple C file to see GIMPLE → RTL lowering
  - `gcc -S -fdump-rtl-expand ...`
  - add `-dp` to annotate the output assembly file

• Typically each line of assembly can be attributed to an RTL instruction

• More generically use
  - `-fverbose-asm`
  - `-fdump-tree-all` `-fdump-ipa-all` `-fdump-rtl-all`
Debugging GCC

- GCC can be debugged using the `-wrapper` option
- GCC will simply prefix all commands that it runs with the text that you provide as the `-wrapper` argument
  - remember, the gcc command is a wrapper

```
gcc -wrapper gdb,--args -v -o hello hello.c
```
```
...  
COLLECT_GCC_OPTIONS='-v' '-o' 'hello' '-mtune=generic' '-march=x86-64'
  gdb --args as -v --64 -o /tmp/cctyamqn.o /tmp/ccLx20Il.s
GNU assembler version 2.30 (x86_64-linux-gnu) using BFD version (GNU Binutils for Ubuntu) 2.30
```
Debugging GCC for VAM

$ gdb --args /home/jeremy/gittrees/gcc-vam/bd/./gcc/xgcc
-B/home/jeremy/gittrees/gcc-vam/bd/./gcc/ -xc -nostdinc /dev/null -S -o
/dev/null -fself-test=../../gcc/gcc/testsuite/selftests

... Type "apropos word" to search for commands related to "word"...
Reading symbols from /home/jeremy/gittrees/gcc-vam/bd/./gcc/xgcc...
(gdb) run
Starting program: /home/jeremy/gittrees/gcc-vam/bd/gcc/xgcc
-B/home/jeremy/gittrees/gcc-vam/bd/./gcc/ -xc -nostdinc /dev/null -S -o
/dev/null -fself-test=../../gcc/gcc/testsuite/selftests

... [Detaching after vfork from child process 726651]
../../gcc/gcc/selftest.cc:410: test_locate_file: FAIL: unable to open
file: ../../gcc/gcc/testsuite/selftests/example.txt
cc1: internal compiler error: in fail_formatted, at selftest.cc:63
0x1a4b260 selftest::fail_formatted(selftest::location const&, char const*, ...)
../../gcc/gcc/selftest.cc:63
What Else Did I Do?

- Created a minimal `vam.cc`
  - to define and initialize `targetm`
- Hand created `vam.opt.urls` (bug?)
- Created `gcc/common/config/vam/vam-common.cc`
  - copied from template in parent directory
- Added VAM to `doc/invoke.texi`
- Configured with `-enable-maintainer-mode`
  - regenerated some files
What Next?
What next

- This is part of our 3 month graduate training course
  - this section is given over 5 days, supported by exercises
  - distilled today into 25 minutes!
- Hopefully gives back-end context to Dave Malcolm’s talk
- One day I will create a full public tutorial on GCC
  - probably when I retire...
- In the meantime...
  - github.com/embecosm/gcc-vam
Thank You

jeremy.bennett@embecosm.com

embecosm.com