The LLVM Assembler and Machine Code Infrastructure
Overview
Overview

• What?
Overview

• What?
• Why?
Overview

- What?
- Why?
- How?
Overview

• What?
• Why?
• How?
• High-Level Design Goals
Overview

• What?
• Why?
• How?
• High-Level Design Goals
• Architecture
Overview

- What?
- Why?
- How?
- High-Level Design Goals
- Architecture
- Status
What?
What?

- What is MC?
What?

- What is MC?
  - "Machine code"
What?

• What is MC?
  – “Machine code”
  – Focus is working with “object files”
What?

• What is MC?
  – "Machine code"
  – Focus is working with "object files"
• Project started late 2009
What?

- What is MC?
  - “Machine code”
  - Focus is working with “object files”

- Project started late 2009
  - Enabled for production in LLVM 2.8 (Oct 2010)
Why?
Why?

• Direct object writing
Why?

- Direct object writing
  - Simplicity, correctness, and performance
Why?

• Direct object writing
  – Simplicity, correctness, and performance
  – Single source of truth
Why?

• Direct object writing
  – Simplicity, correctness, and performance
  – Single source of truth

• Advanced micro-arch optimizations
Why?

- Direct object writing
  - Simplicity, correctness, and performance
  - Single source of truth
- Advanced micro-arch optimizations
- Platform for advancing low-level tools
How?
How?

Standard Compiler

Code (.c)

Code Generator

Assembly Printer

Assembly (.s)
How?

Standard Compiler

Code (.c)

Code Generator

Assembly Printer

Assembly (.s)
How?

**Standard Compiler**

- **Code (.c)**
  - Code Generator
  - Assembly Printer
  - Assembly (.s)

**LLVM JIT**

- **Code (.c)**
  - Code Generator
  - Assembly Printer (JIT)
  - JIT Encoder
  - Execution
How?

Standard Compiler

- Code (.c)
  - Code Generator
  - Assembly Printer
  - Assembly (.s)

LLVM JIT

- Code (.c)
  - Code Generator
  - Assembly Printer (JIT)
  - JIT Encoder
  - Execution
How?

Standard Compiler

- Code (.c)
  - Code Generator
    - Assembly Printer
      - Assembly (.s)

LLVM JIT

- Code (.c)
  - Code Generator
    - Assembly Printer (JIT)
      - JIT Encoder
        - Execution

Duplicate Code!
How?

**Standard Compiler**

- Code (.c)
  - Code Generator
    - Assembly Printer
      - Assembly (.s)

**LLVM JIT**

- Code (.c)
  - Code Generator
    - Assembly Printer (JIT)
      - JIT Encoder
        - Execution

**Issues**

- Duplicate Code!
- No Public API!
How?
How?

Modern Compiler

- Code (.c)
  - Code Generator
  - Assembly Printer
  - Assembly (.s)
How?

Modern Compiler

Code (.c)

Code Generator

Assembly Printer

Assembly (.s)
How?

Modern Compiler

Code (.c)

Code Generator

MCized Printer

MCStreamer
How?

Modern Compiler

Code (.c)

Code Generator

MCized Printer

MCStreamer

AsmPrinter

Assembly (.s)

ObjectWriter

Object File (.o)
High-Level Design Goals
High-Level Design Goals

- Reuse
High-Level Design Goals

- Reuse
- Performance
High-Level Design Goals

- Reuse
- Performance
  - No redundant effort
High-Level Design Goals

- Reuse
- Performance
  - No redundant effort
- Testability
High-Level Design Goals

- Reuse
- Performance
  - No redundant effort
- Testability
  - Test components in isolation
High-Level Design Goals

- Reuse
- Performance
  - No redundant effort
- Testability
  - Test components in isolation
- Flexibility
High-Level Design Goals

• Reuse
• Performance
  – No redundant effort
• Testability
  – Test components in isolation
• Flexibility
  – Many uses for each MC component
High-Level Design Goals

- Reuse
- Performance
  - No redundant effort
- Testability
  - Test components in isolation
- Flexibility
  - Many uses for each MC component
- Pluggable Targets
High-Level Design Goals

- Reuse
- Performance
  - No redundant effort
- Testability
  - Test components in isolation
- Flexibility
  - Many uses for each MC component
- Pluggable Targets
- Non-pluggable Object Formats
How is MC Used?
How is MC Used?

- Assembly (.s)
- Assembler Parser
- Object Writer
- Object File (.o)
How is MC Used?

- Assembly (.s)
  - Assembler Parser
  - Object Writer
  - Object File (.o)

It's an Assembler!
How is MC Used?

It’s an Assembler!
How is MC Used?

It's an Assembler!

It's a Compiler!

- Assembly (.s)
- Code (.c)
- Assembler Parser
- Code Generator
- Object Writer
- Assembly Printer
- Object File (.o)
- Assembly (.s)
How is MC Used?

- Assembly (.s)
  - Assembler Parser
    - Object Writer
      - Object File (.o)
  - Code Generator
    - Assembly Printer
      - Assembly (.s)
- Code (.c)
  - Code Generator
    - Object Writer
      - Object File (.o)

It's an Assembler!

It's a Compiler!
How is MC Used?

It's an Assembler!

It's a Compiler!

It's a Compil assembler!
How is MC Used?

- Assembly (.s)
  - Assembler Parser
    - Object Writer
      - Object File (.o)
  - Assembly (.s)
- Code (.c)
  - Code Generator
    - Assembly Printer
      - Assembly (.s)
  - Code (.c)
- Code (.c)
  - Code Generator
    - Object Writer
      - Object File (.o)
  - Code (.c)
  - JIT
  - Execution!

It's an Assembler!
It's a Compiler!
It's a Compilassembler!
How is MC Used?

Assembler (.s) → Assembler Parser → Object Writer → Object File (.o)

Code (.c) → Code Generator → Assembly Printer → Assembly (.s)

Code (.c) → Code Generator → Object Writer → Object File (.o)

Code (.c) → Code Generator → JIT → Execution!

It’s an Assembler!
It’s a Compiler!
It’s a Comilasembler!
It’s a JIT!
How is MC Used?

It's an Assembler!

It's a Compiler!

It's a Compilassembler!

It's a JIT!
(with inline asm support!)
How is MC Used?

It's an Assembler!

It's a Compiler!

It's a Compilassembler! (with inline asm support!)

It's a JIT!
How is MC Used?

It's an Assembler!

It's a Compiler!

It's a Compilassembler!

It's a JIT!

(with inline asm support!)

It's a Disassembler!
How is MC Used?

Assembly (.s) -> Assembler Parser -> Object Writer -> Object File (.o)

Code (.c) -> Code Generator -> Assembly Printer -> Assembly (.s)

Code (.c) -> Code Generator -> Object Writer -> Object File (.o)

Code (.c) -> Code Generator -> JIT -> Execution!

Object File (.o) -> Disassembler -> Assembly Printer -> Assembly (.s)
How is MC Used?

- Assembly (.s) -> Assembler Parser
- Code (.c) -> Code Generator
- Code (.c) -> Code Generator
- Code (.c) -> Code Generator
- Object File (.o) -> Disassembler

- MCStreamer
- MCInst

- Object Writer -> Object File (.o)
- Assembly Printer -> Assembly (.s)
- Object Writer -> Object File (.o)
- JIT
- Assembly Printer -> Assembly (.s)
How is MC Used?

- Assembly (.s)
- Code (.c)
- Code (.c)
- Code (.c)
- Object File (.o)

Assembler Parser

Code Generator

Code Generator

Code Generator

Disassembler

MCStreamer

Object Writer

Assembly Printer

Object Writer

JIT

Assembly Printer

Object File (.o)

Assembly (.s)

Object File (.o)

Execution!

Assembly (.s)
MCStreamer
MCStreamer

- Core MC Component
MCStreamer

- Core MC Component
  - Programmatic Assembler API
MCStreamer

- Core MC Component
  - Programmatic Assembler API
  - Best explained by example
MCStreamer

- Core MC Component
  - Programmatic Assembler API
  - Best explained by example

```c
#include <stdio.h>

int main() {
    printf("Hello World!\n");
    return 0;
}
```
.section __TEXT,__text,regular,pure_instructions
.globl _main
.align 4, 0x90

_main:
    # @main
    pushl %ebp
    movl %esp, %ebp
    subl $8, %esp
    movl $_str, (%esp)
    calll _.puts
    xorl %eax, %eax
    addl $8, %esp
    popl %ebp
    ret

.section __TEXT,__cstring,cstring_literals
_str:
    # @str
    .asciz "Hello World!"
.section __TEXT,__text,regular,pure_instructions
.globl _main
.align 4, 0x90

_main:
    pushl %ebp
    movl %esp, %ebp
    subl $8, %esp
    movl $-_str, (%esp)
    calll _puts
    xorl %eax, %eax
    addl $8, %esp
    popl %ebp
    ret

.section __TEXT,__cstring,cstring_literals
_str:
    .asciz "Hello World!"
.section  __TEXT,__text,regular,pure_instructions
.globl  _main
.align  4, 0x90

_main:
    # @main
    pushl %ebp
 movl %esp, %ebp
 subl $8, %esp
 movl $_str, (%esp)
 calll _puts
 xorl %eax, %eax
 addl $8, %esp
 popl %ebp
 ret

.section  __TEXT,__cstring,cstring_literals
_str:
    # @str
.asciz  "Hello World!"
MCStreamer

```
.section __TEXT,__text,regular,pure_instructions
.globl _main
.align 4, 0x90
_main:
    # @main
    pushl %ebp
    movl %esp, %ebp
    subl $8, %esp
    movl $_str, (%esp)
    calll _.puts
    xorl %eax, %eax
    addl $8, %esp
    popl %ebp
    ret

.section __TEXT,__cstring,cstring_literals
_str:
    # @str
    .asciz "Hello World!"
```
void foo(MCStreamer &Out, MCContext &Ctx) {
    Out.SwitchSection(Ctx.getMachOSection(...));

    ...

    .section __TEXT,__cstring,cstring_literals
    .asciz "Hello World!"
    }
MCStreamemr

.mc
  .section __TEXT,__text,regular,pure_instructions
  .globl __main
  .align 4, 0x90
  __main:
  # @main
  pushl %ebp
  movl %esp, %ebp
  subl $8, %esp
  movl $__str, (%esp)
  calll __puts
  xorl %eax, %eax
  addl $8, %esp
  popl %ebp
  ret

.mc
  .section __TEXT,__cstring,cstring_literals
  __str:
  # @str
  .asciz "Hello World!"
void foo(MCStreamer &Out, MCContext &Ctx) {
    Out.EmitSymbolAttribute(Ctx.LookupSymbol("_main"), MCSymbolAttr::MCSA_Global);
}
```assembly
.section   __TEXT,__text,regular,pure_instructions
.globl   main
.align   4, 0x90
_main:
    pushl  %ebp
    movl  %esp, %ebp
    subl  $8, %esp
    movl  $__str, (%esp)
    call   __.puts
    xorl  %eax, %eax
    addl  $8, %esp
    popl  %ebp
    ret

.section   __TEXT,__cstring,cstring_literals
_str:        # @str
    .asciz   "Hello World!"
```
void foo(MCStreamer &Out, MCContext &Ctx) {
    ... 
    Out.EmitValueToAlignment(4, 0x90);
    ...
    _str: 
    .section __TEXT,__cstring,cstring_literals
    .asciz "Hello World!"
}
MCStreamer

```
.section __TEXT,__text,regular,pure_instructions
.globl _main
.align 4, 0x90

main:
  # @main
  pushl %ebp
  movl %esp, %ebp
  subl $8, %esp
  movl $-_str, (%esp)
  calll _puts
  xorl %eax, %eax
  addl $8, %esp
  popl %ebp
  ret

.section __TEXT,__cstring,cstring_literals
_str:
  # @str
  .asciz "Hello World!"
```
MCStreamer

```c
void foo(MCStreamer &Out, MCContext &Ctx) {
    Out.EmitLabel(Ctx.LookupSymbol("_main"));
    ...
}
```

```assembly
.main:
    pushl  %ebp
    movl  %esp, %ebp
    subl  $8, %esp
    movl  $-_str, %esp
    calll  _puts
    xorl  %eax, %eax
    addl  $8, %esp
    popl  %ebp
    ret

.section  __TEXT,__cstring,cstring_literals
    .asciz "Hello World!"
```

```
.globl _main
.align  4, 0x90
```

```
.section   __TEXT,__text,regular,pure_instructions
```

```
.pushl   %ebp
    .globl   _main
    .align   4, 0x90
```

```
pushl   %ebp
    movl   %esp, %ebp
    subl   $8, %esp
    movl   $-_str, %esp
    calll  _puts
    xorl   %eax, %eax
    addl   $8, %esp
    popl   %ebp
    ret

.section  __TEXT,__cstring,cstring_literals
    .asciz "Hello World!"
```
void foo(MCStreamer &Out, MCContext &Ctx) {
    Out.EmitLabel(Ctx.LookupSymbol("_main"));
}

...
.section  __TEXT,__text,regular,pure_instructions
.globl  _main
.align  4, 0x90
main:
    # @main
        pushl  %ebp
        movl  %esp, %ebp
        subl  $8, %esp
        movl  $-_str, (%esp)
        calll  _puts
        xorl  %eax, %eax
        addl  $8, %esp
        popl  %ebp
        ret

.section  __TEXT,__cstring,cstring_literals
_str:
    # @str
        .asciz  "Hello World!"
void foo(MCStreamer &Out, MCContext &Ctx) {
    MCInst I = { ??? };
    Out.EmitInstruction(I);

    ...
MCInst
MCInst

- Second major MC abstraction
MCInst

- Second major MC abstraction
- MCInst is a simple representation of a machine instruction
MCInst

- Second major MC abstraction
- MCInst is a simple representation of a machine instruction
  - Consists of opcode and operands
MCInst

- Second major MC abstraction
- MCInst is a simple representation of a machine instruction
  - Consists of opcode and operands
  - Operands:
MCInst

- Second major MC abstraction
- MCInst is a simple representation of a machine instruction
  - Consists of opcode and operands
  - Operands:
    - Registers
MCInst

- Second major MC abstraction
- MCInst is a simple representation of a machine instruction
  - Consists of opcode and operands
  - Operands:
    - Registers
    - Immediates (constants and expressions)
MCInst

- Second major MC abstraction
- MCInst is a simple representation of a machine instruction
  - Consists of opcode and operands
  - Operands:
    - Registers
    - Immediates (constants and expressions)
    - Floating point immediates
MCInst

- Second major MC abstraction
- MCInst is a simple representation of a machine instruction
  - Consists of opcode and operands
  - Operands:
    - Registers
    - Immediates (constants and expressions)
    - Floating point immediates
  - Affords simple C API
The \texttt{llvm-mc} tool
The `llvm-mc` tool

- `llvm-mc` is the command line tool for testing MC
The \texttt{llvm-mc} tool

- \texttt{llvm-mc} is the command line tool for testing MC
  - Includes assembler, object file writer, and disassembler
The `llvm-mc` tool

- `llvm-mc` is the command line tool for testing MC
  - Includes assembler, object file writer, and disassembler
- Can use it to show encoding and MCInst structure
The **llvm-mc** tool

- **llvm-mc** is the command line tool for testing MC
  - Includes assembler, object file writer, and disassembler
- Can use it to show encoding and MCInst structure

```bash
$ llvm-mc --show-inst t.s
pushl %ebp

## <MCInst #2044 PUSH32r
## <MCOperand Reg:44>>
```
The **llvm-mc** tool

- **llvm-mc** is the command line tool for testing MC
  - Includes assembler, object file writer, and disassembler
- Can use it to show encoding and MCInst structure

```
$ llvm-mc --show-inst t.s
  pushl  %ebp         ## <MCInst #2044 PUSH32r
  ##   <MCOperand Reg:44>>

$ llvm-mc --show-encoding t.s
  pushl  %ebp         ## encoding: [0x55]
```
Instruction Matching
Instruction Matching

- Ties together the parsed instruction with target `.td` files
Instruction Matching

- Ties together the parsed instruction with target .td files
- Uses a custom tblgen backend to generate match tables
Instruction Matching

• Ties together the parsed instruction with target .td files
• Uses a custom tblgen backend to generate match tables

```plaintext
...,
{ X86::PUSHF16, "pushfw", Convert, { }, MCK_CS, 0 },,
{ X86::PUSH32r, "pushl", Convert__Reg1_0, { MCK_GR32 }, 0 },,
{ X86::PUSH32rmm, "pushl", Convert__Reg1_0, { MCK_GR32 }, 0 },,
{ X86::PUSHCS32, "pushl", Convert, Feature_In32BitMode },
... 
```
Instruction Matching

- Ties together the parsed instruction with target .td files
- Uses a custom tblgen backend to generate match tables
Current Status
Current Status

- Integrated assembler is default for X86 for Darwin
Current Status

- Integrated assembler is default for X86 for Darwin
- Lots of testing and qualification for X86
Current Status

- Integrated assembler is default for X86 for Darwin
- Lots of testing and qualification for X86
- ELF/X86-64 support is done
  - On by default in Clang top-of-tree

Monday, November 29, 2010
Current Status

- Integrated assembler is default for X86 for Darwin
- Lots of testing and qualification for X86
- ELF/X86-64 support is done
  - On by default in Clang top-of-tree
- COFF support is well underway
  - Passes many programs in LLVM test-suite repository
Current Status

- Integrated assembler is default for X86 for Darwin
- Lots of testing and qualification for X86
- ELF/X86-64 support is done
  - On by default in Clang top-of-tree
- COFF support is well underway
  - Passes many programs in LLVM test-suite repository
- ARM support is ongoing
Current Status: Performance
Current Status: Performance

- Example numbers from SPECCPU's 403.gcc
  - clang with and without -integrated-as
  - Using -00 -g for i386
Current Status: Performance

- Example numbers from SPECCPU’s 403.gcc
  - clang with and without `-integrated-as`
  - Using `-O0 -g` for i386
Current Status: Performance

- Example numbers from SPECCPU’s 403.gcc
  - clang with and without `-integrated-as`
  - Using `-00 -g` for i386

Compile Wall Time (s)

- System Assembler
- Integrated Assembler

15% Faster!

Monday, November 29, 2010
Current Status: Performance

- Example numbers from SPECCPU’s 403.gcc
  - clang with and without `-integrated-as`
  - Using `-00 -g` for i386

![Compiled Wall Time Graph]

- i386.c
  - System Assembler: 0.91
  - Integrated Assembler: 0.79
- insn-emit.c
  - System Assembler: 1.22
  - Integrated Assembler: 1.00
- combine.c
  - System Assembler: 1.00
  - Integrated Assembler: 1.00
- insn-recog.c
  - System Assembler: 1.00
  - Integrated Assembler: 1.00

15% Faster!
22% Faster!

Monday, November 29, 2010
Current Status: Performance

- Example numbers from SPECCPU’s 403.gcc
  - clang with and without \texttt{-integrated-as}
  - Using \texttt{-O0 \ -g} for i386

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{chart.png}
\caption{Compile Wall Time (s)}
\end{figure}

\begin{itemize}
\item i386.c: 0.91 (System Assembler), 0.79 (Integrated Assembler)
\item insn-emit.c: 1.22 (System Assembler), 1.00 (Integrated Assembler)
\item combine.c: 0.89 (System Assembler), 0.79 (Integrated Assembler)
\item insn-recog.c: 0.79 (System Assembler), 0.79 (Integrated Assembler)
\end{itemize}

15\% Faster!
22\% Faster!
13\% Faster!
Current Status: Performance

- Example numbers from SPECCPU’s 403.gcc
  - clang with and without -integrated-as
  - Using -O0 -g for i386

<table>
<thead>
<tr>
<th></th>
<th>System Assembler</th>
<th>Integrated Assembler</th>
</tr>
</thead>
<tbody>
<tr>
<td>i386.c</td>
<td>0.91</td>
<td>0.79</td>
</tr>
<tr>
<td>insn-emit.c</td>
<td>1.22</td>
<td>1.00</td>
</tr>
<tr>
<td>combine.c</td>
<td>0.89</td>
<td>0.79</td>
</tr>
<tr>
<td>insn-recog.c</td>
<td>3.02</td>
<td>2.37</td>
</tr>
</tbody>
</table>

15% Faster! 22% Faster! 13% Faster! 27% Faster!

Monday, November 29, 2010
Summary

• Good compile-time improvements
Summary

- Good compile-time improvements
- Reduced system complexity
Summary

- Good compile-time improvements
- Reduced system complexity
- Many new tools and opportunities
Summary

- Good compile-time improvements
- Reduced system complexity
- Many new tools and opportunities
- What’s next?
Summary

- Good compile-time improvements
- Reduced system complexity
- Many new tools and opportunities
- What’s next?
  - JIT needs to be converted
Summary

• Good compile-time improvements
• Reduced system complexity
• Many new tools and opportunities
• What’s next?
  – JIT needs to be converted
  – User-level disassembler
Questions?