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?

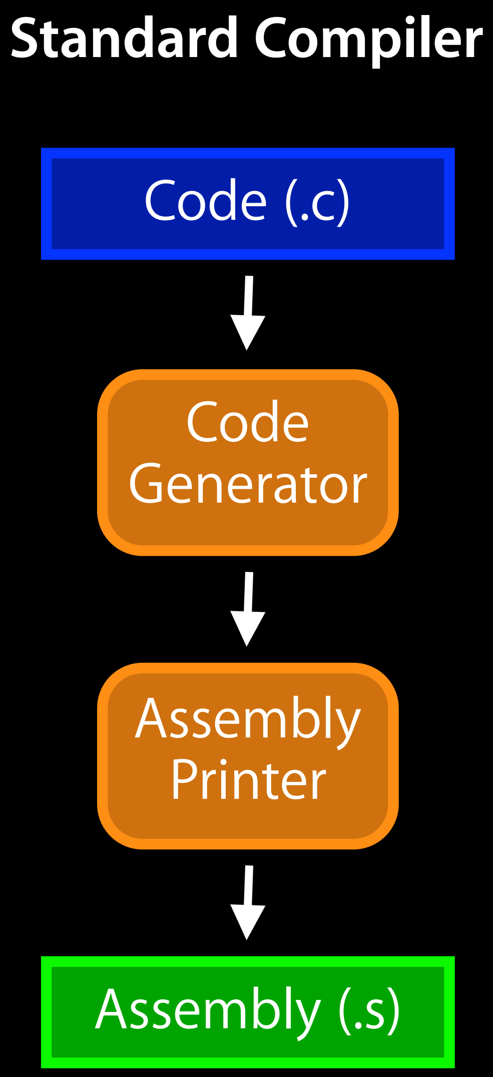
- Direct object writing
  - Simplicity, correctness, and performance
  - Single source of truth
- Advanced micro-arch optimizations

Why?

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

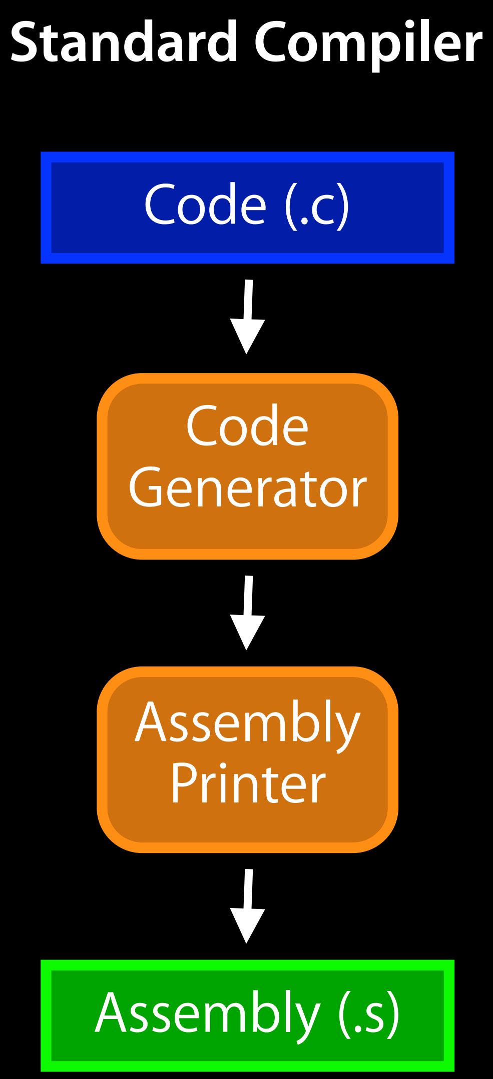
How?

How?

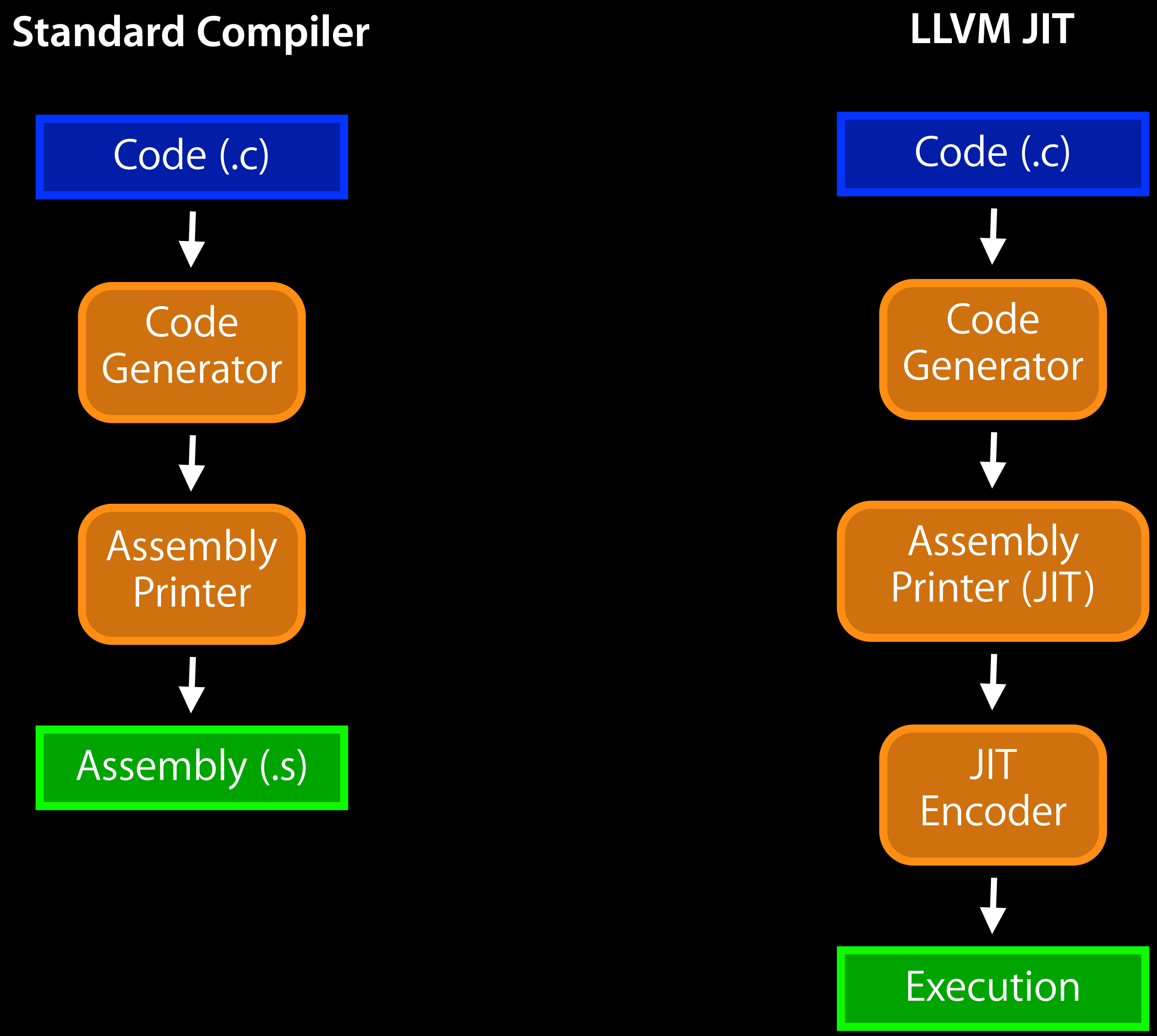


Standard Compiler
Code (.c)
Code Generator
Assembly Printer
Assembly (.s)

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)

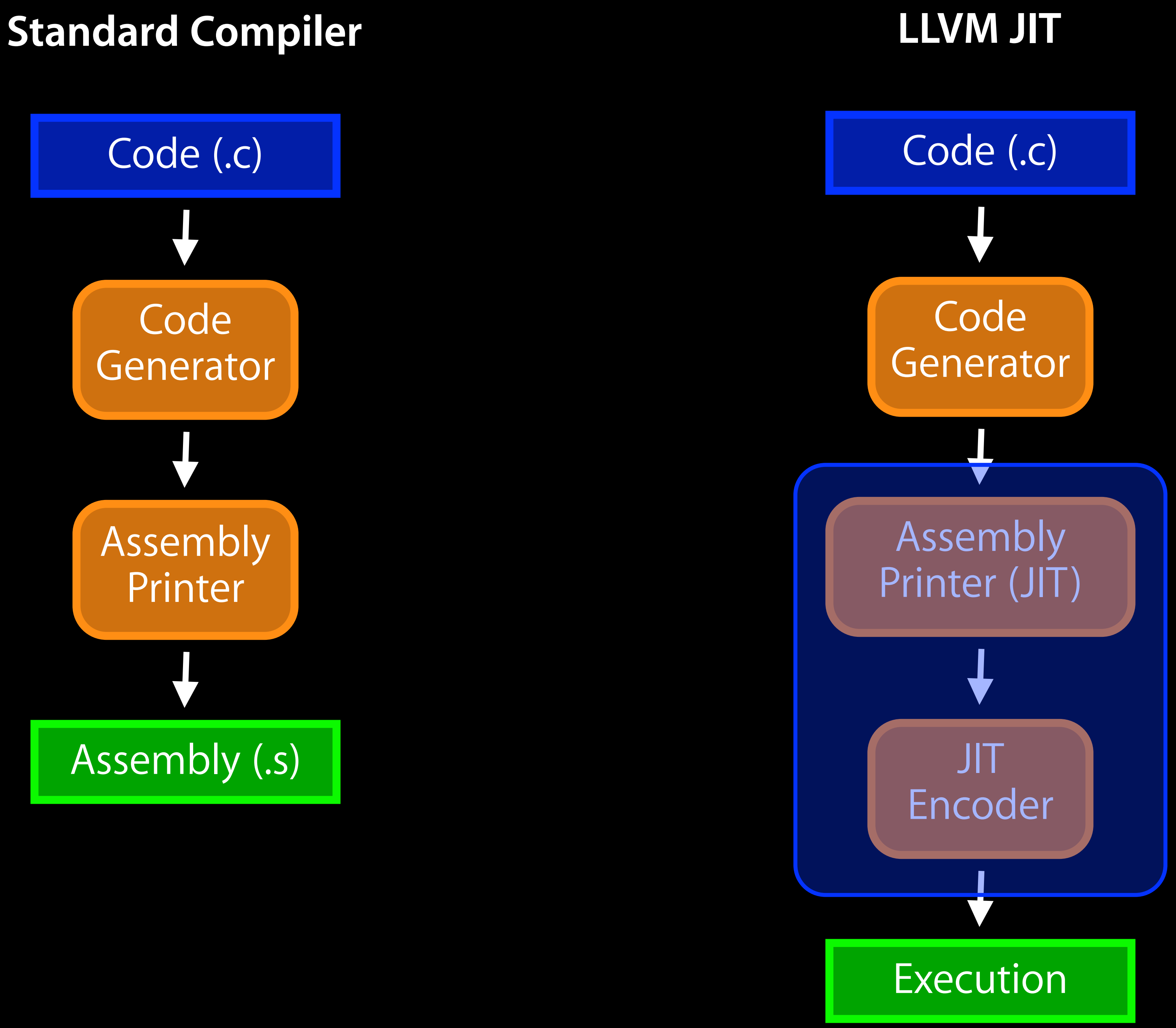
Code Generator

Assembly Printer (JIT)

JIT Encoder

Execution

How?



Standard Compiler

Code (.c)

Code Generator

Assembly Printer

Assembly (.s)

LLVM JIT

Code (.c)

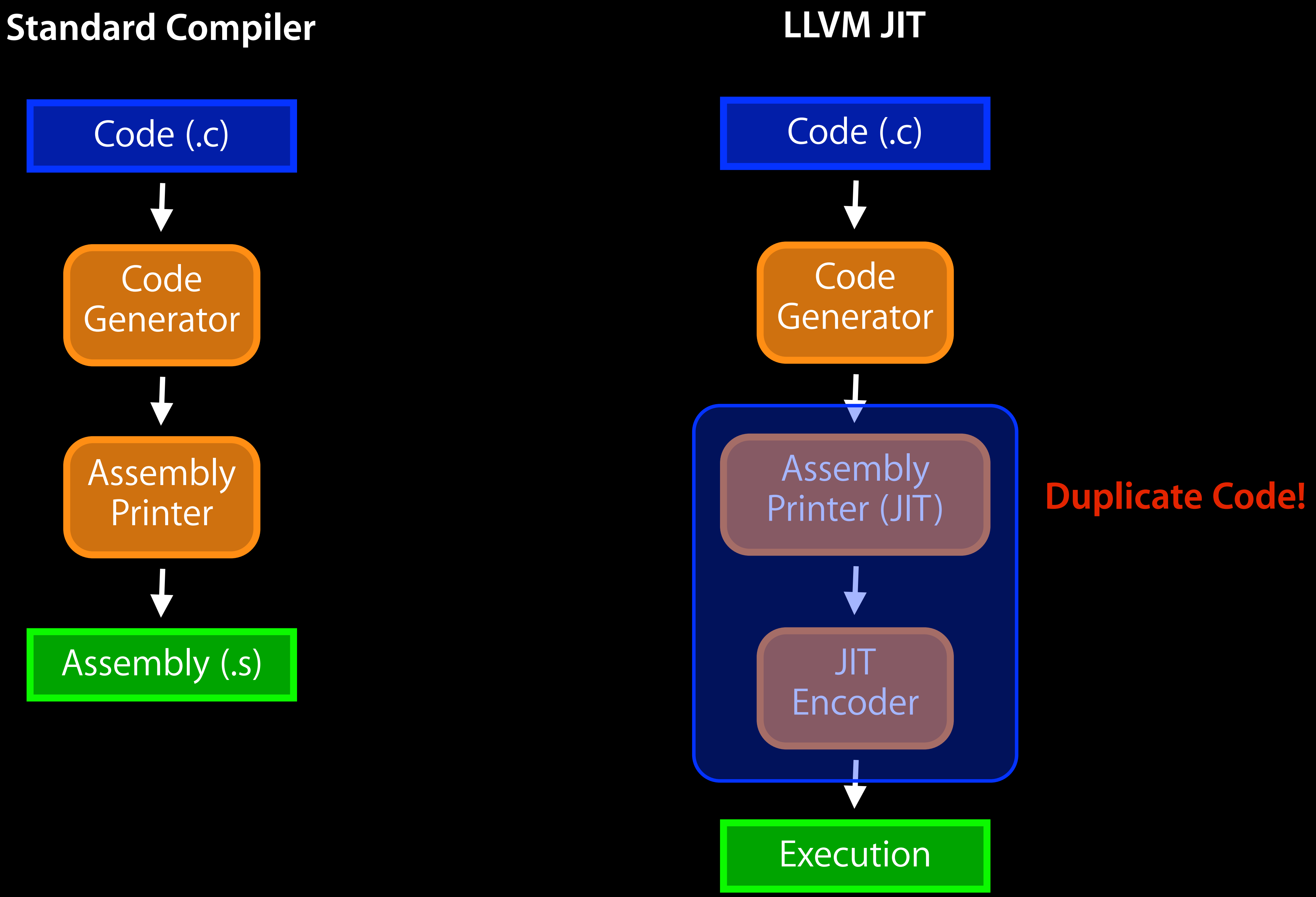
Code Generator

Assembly Printer (JIT)

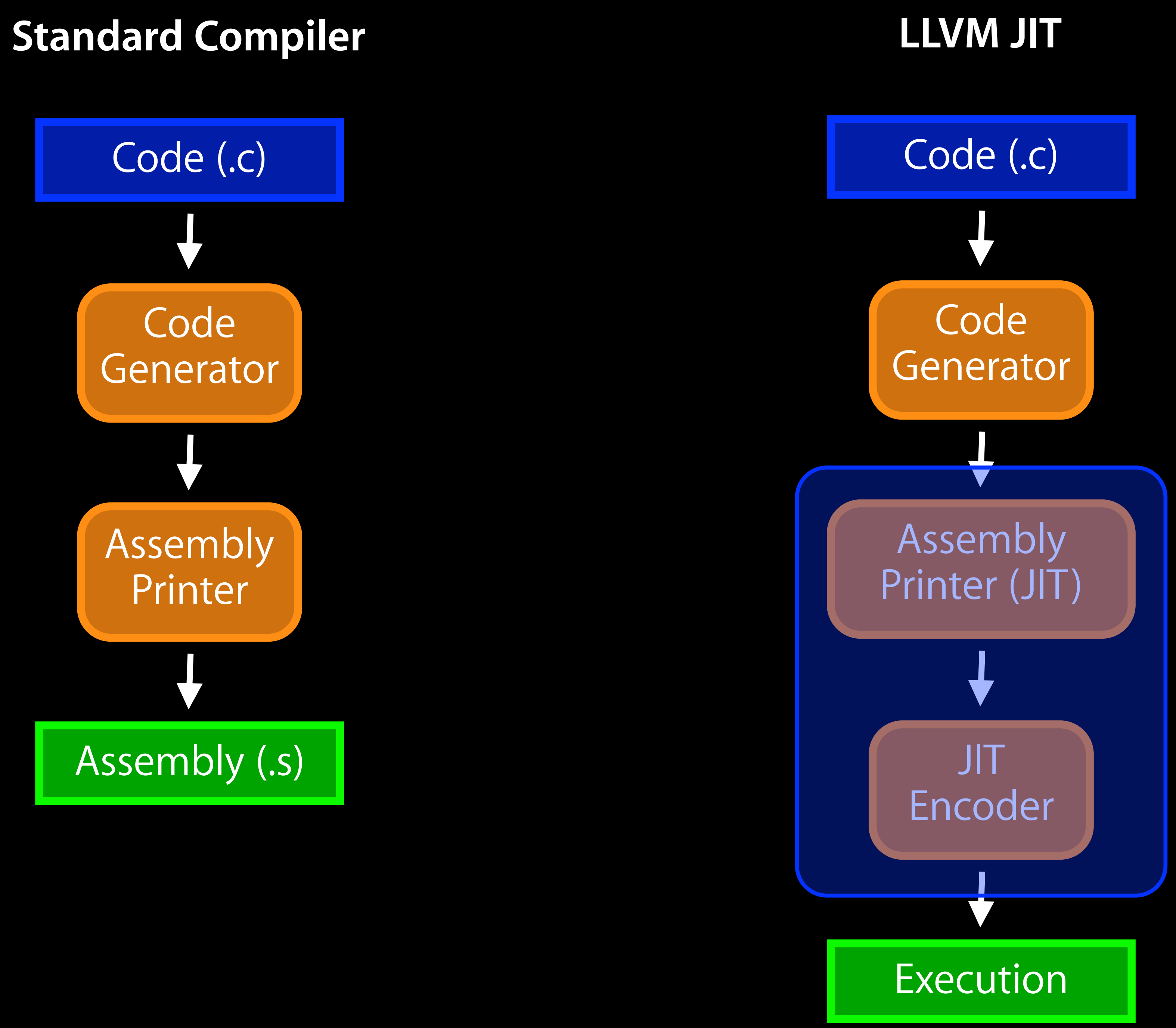
JIT Encoder

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

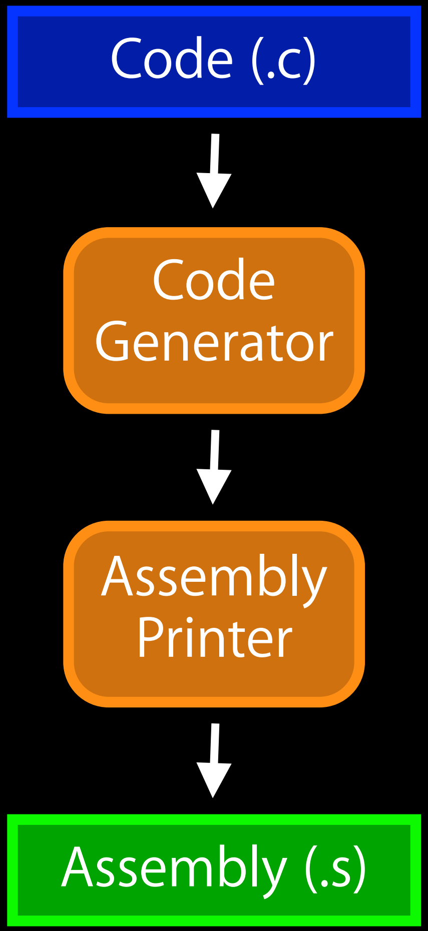
Execution

Duplicate Code!

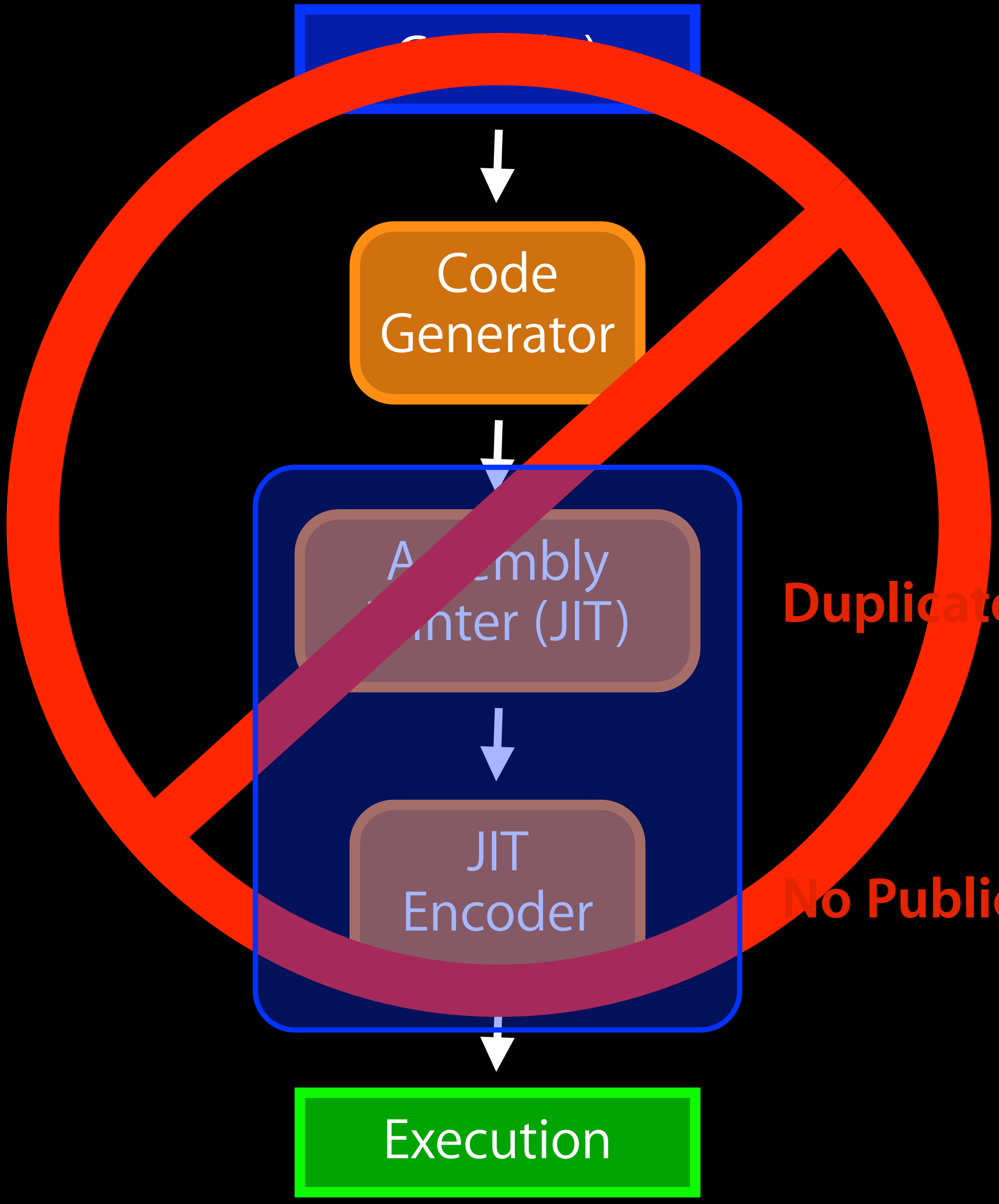
No Public API!

How?

Standard Compiler



LLVM JIT



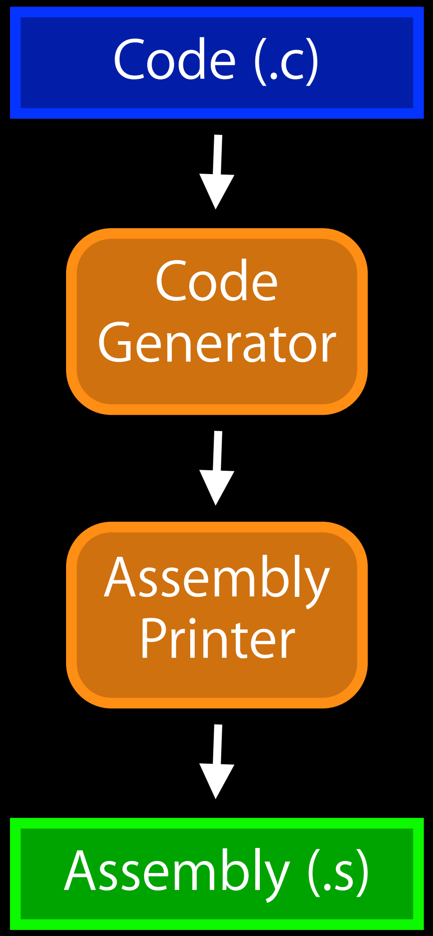
Duplicate Code!

No Public API!

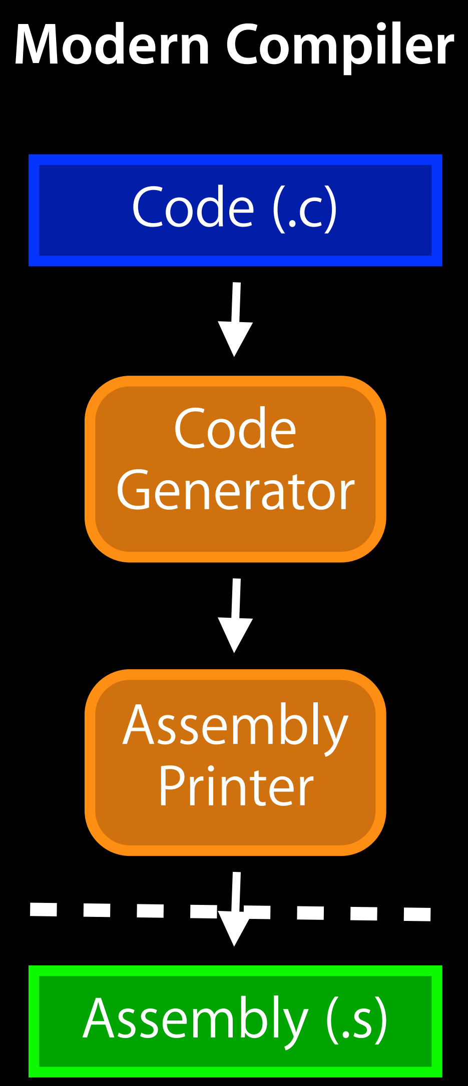
How?

How?

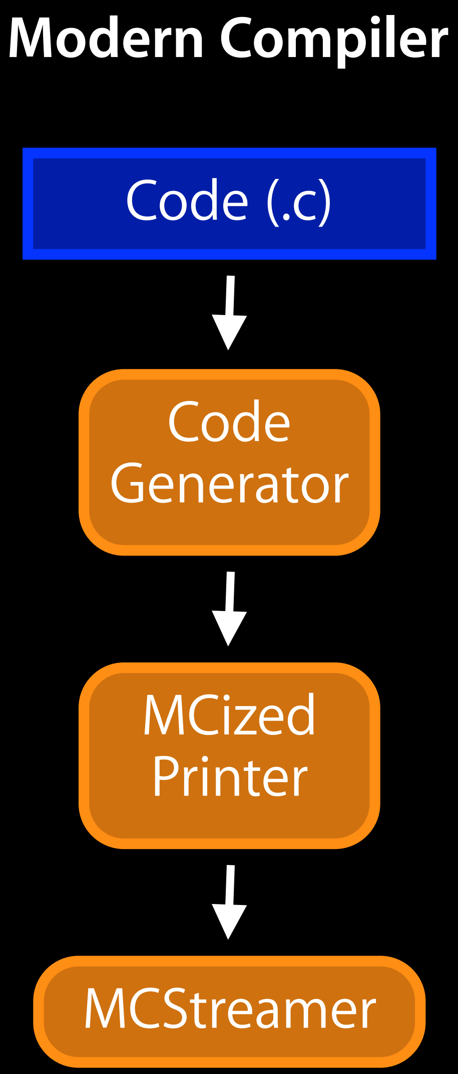
Modern Compiler



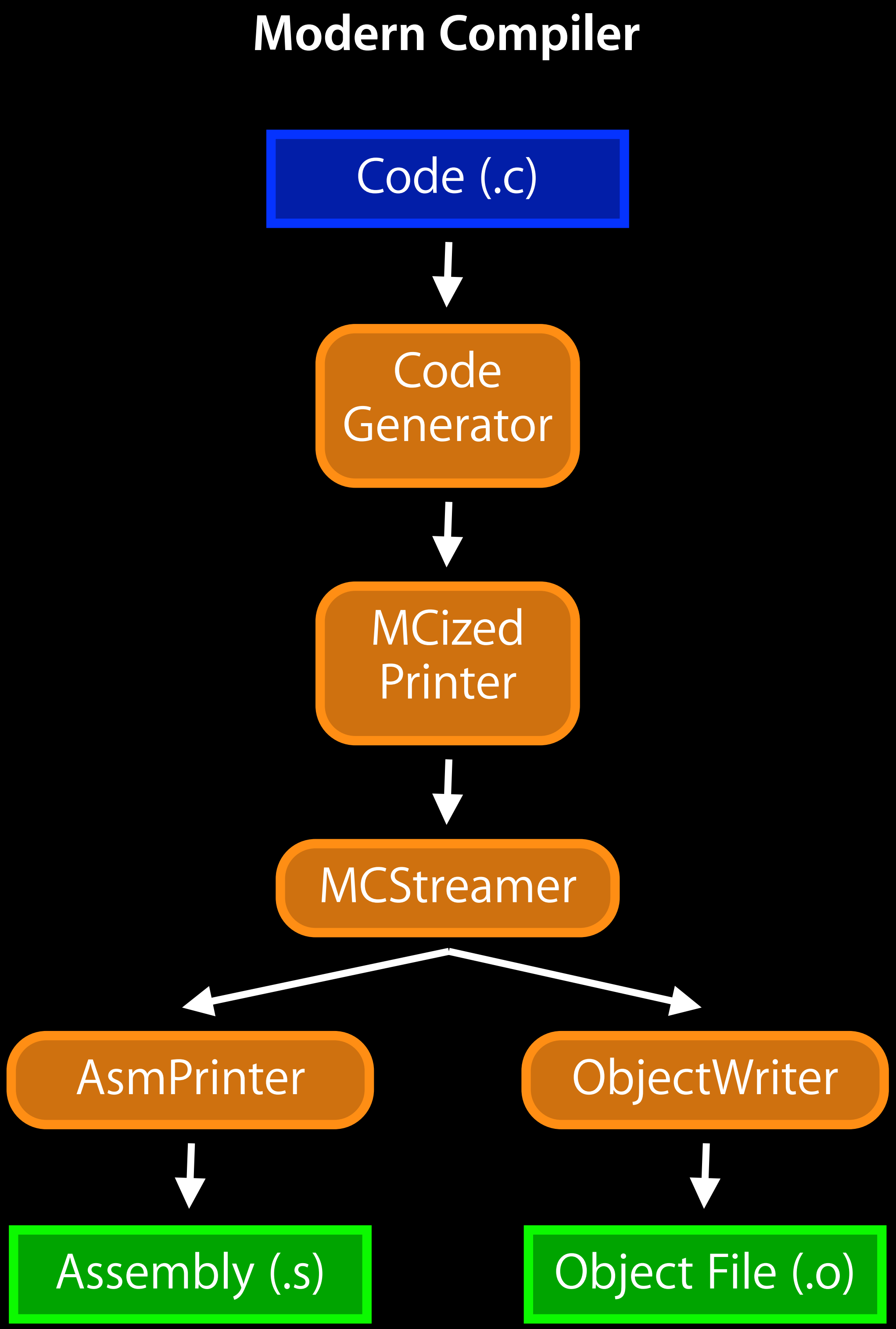
How?


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
ObjectWriter
Assembly (.s)
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

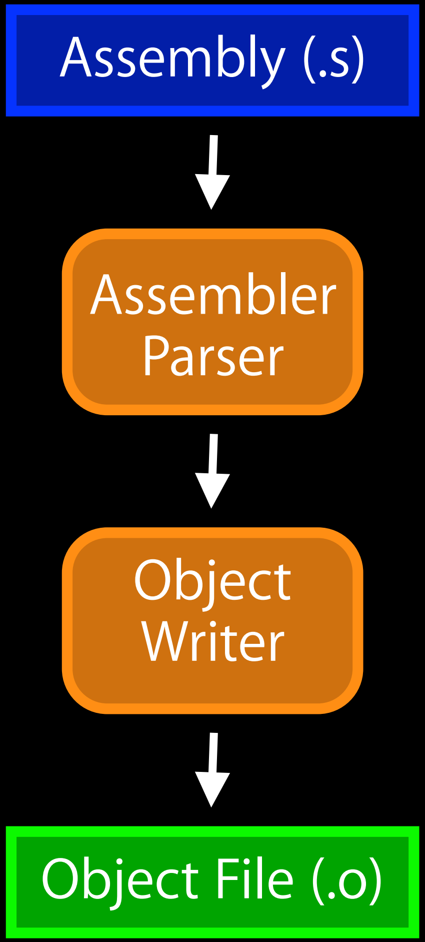
- Reuse
- Performance
  - No redundant effort
- Testability
  - Test components in isolation
- Flexibility
  - Many uses for each MC component
- Pluggable Targets

High-Level Design Goals

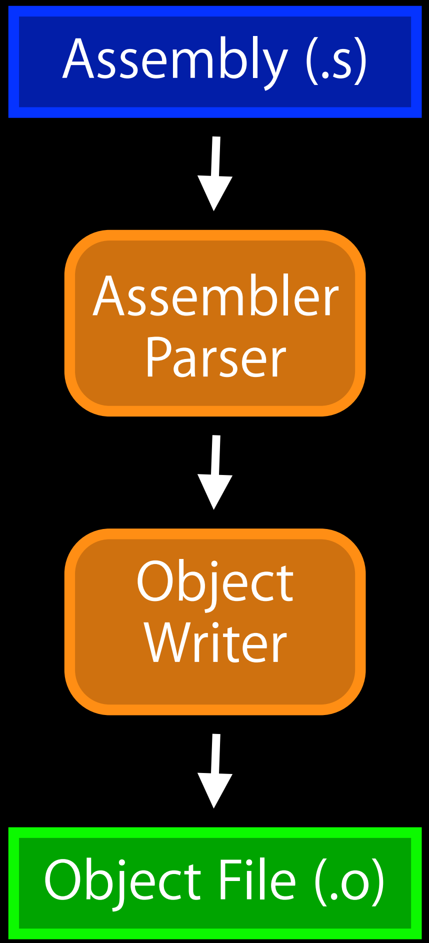
- Reuse
- Performance
  - No redundant effort
- Testability
  - Test components in isolation
- Flexibility
  - Many uses for each MC component
- Pluggable Targets
- Non-pluggable Object Formats

How is MC Used?

How is MC Used?

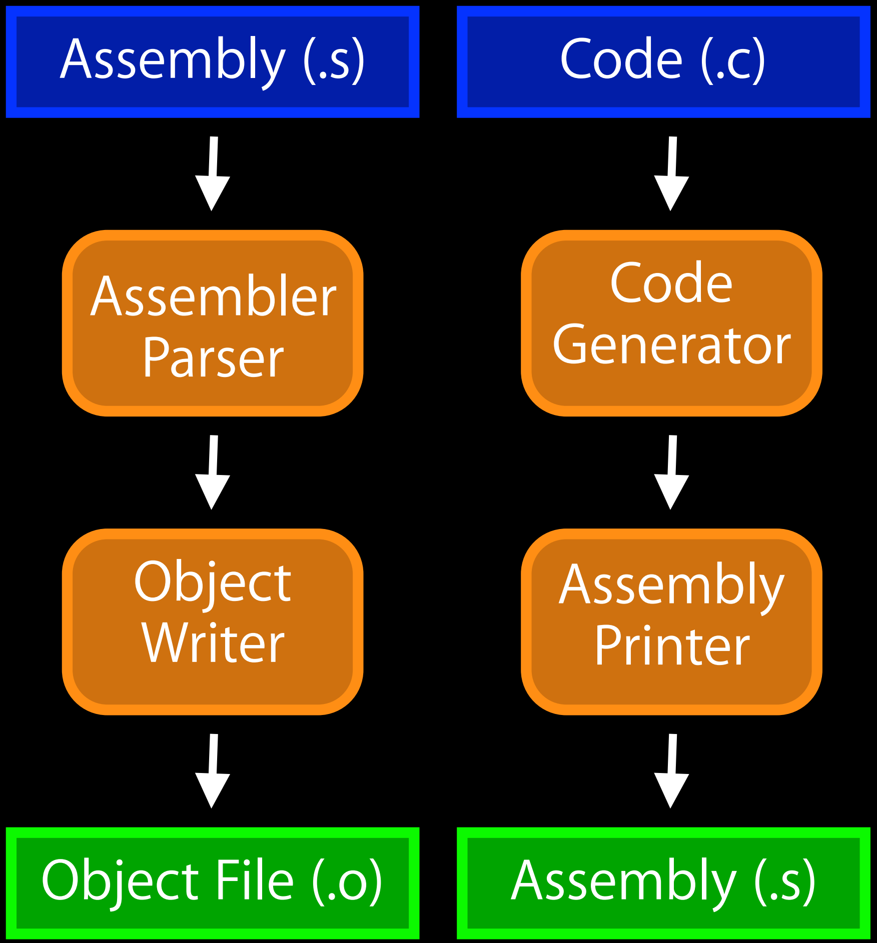


How is MC Used?



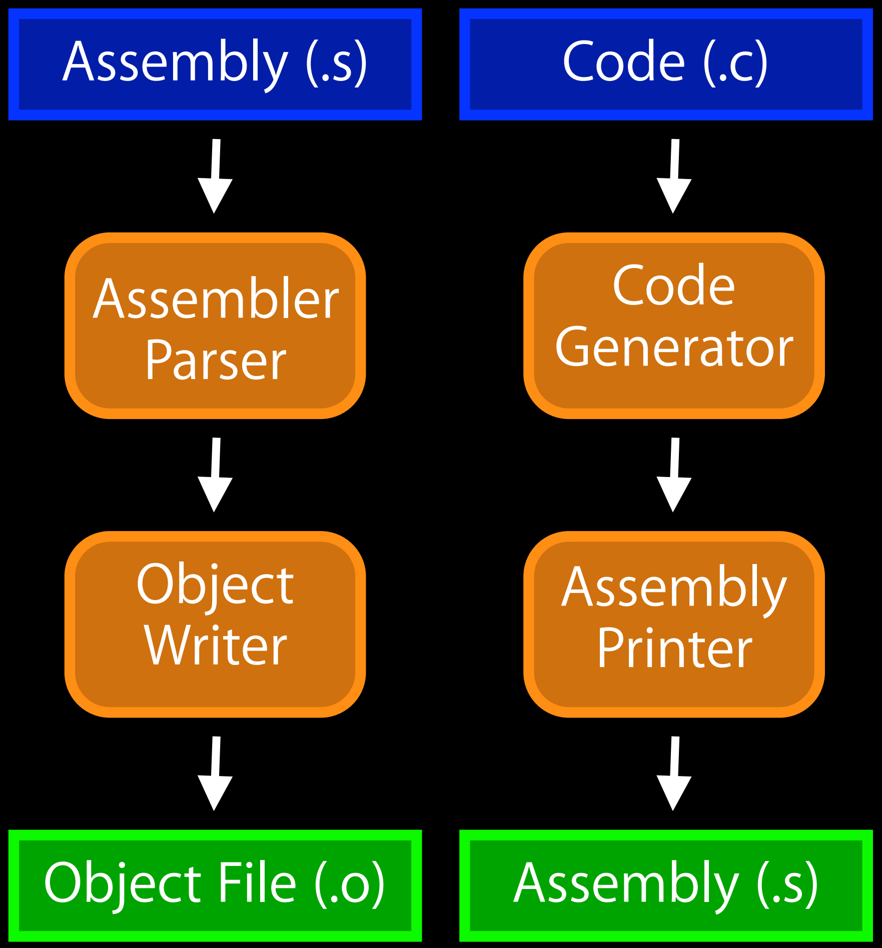
It's an Assembler!

How is MC Used?



It's an Assembler!

How is MC Used?



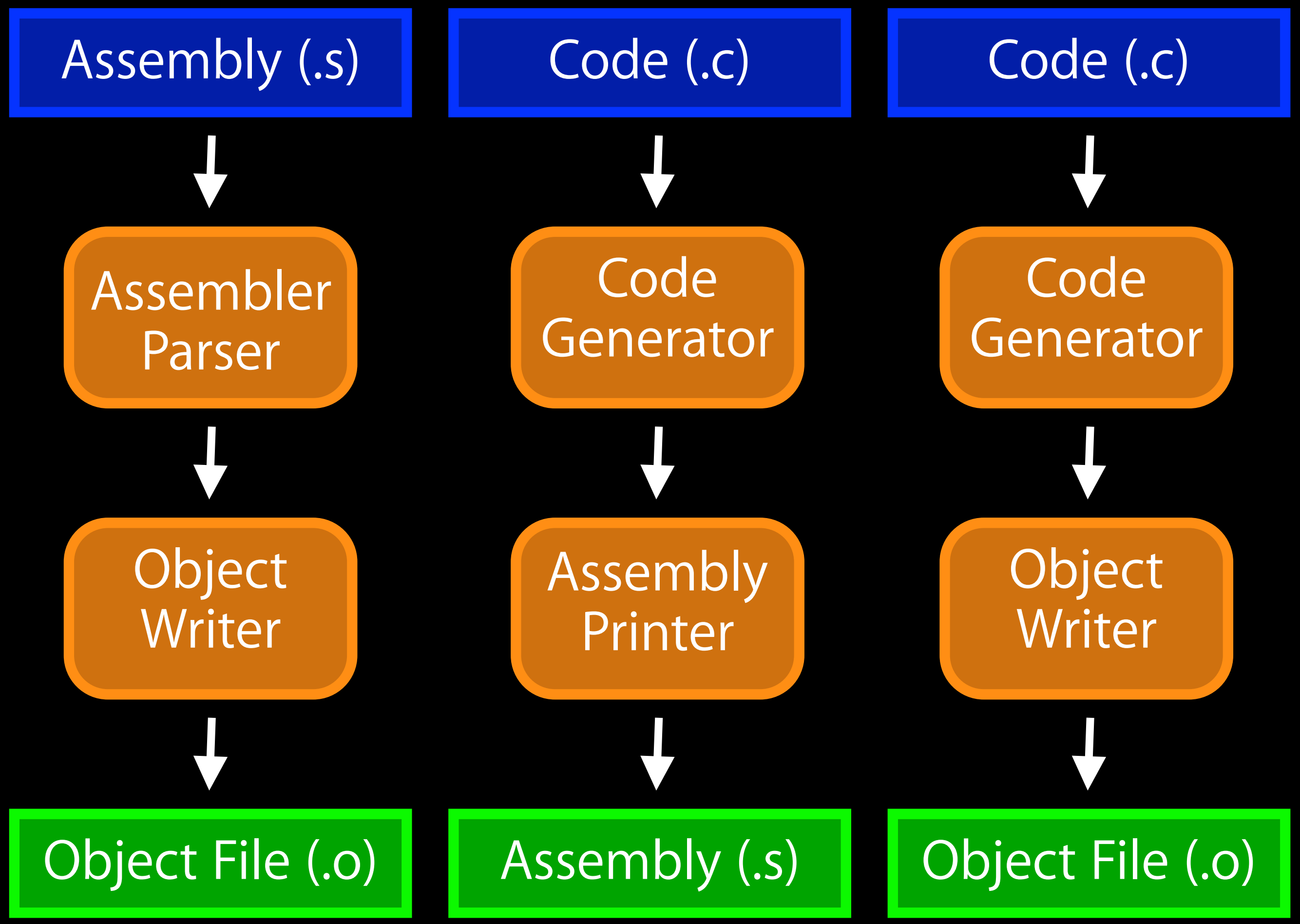
Assembly (.s)
Assembler Parser
Object Writer
Object File (.o)

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

It's an Assembler!

It's a Compiler!

How is MC Used?



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

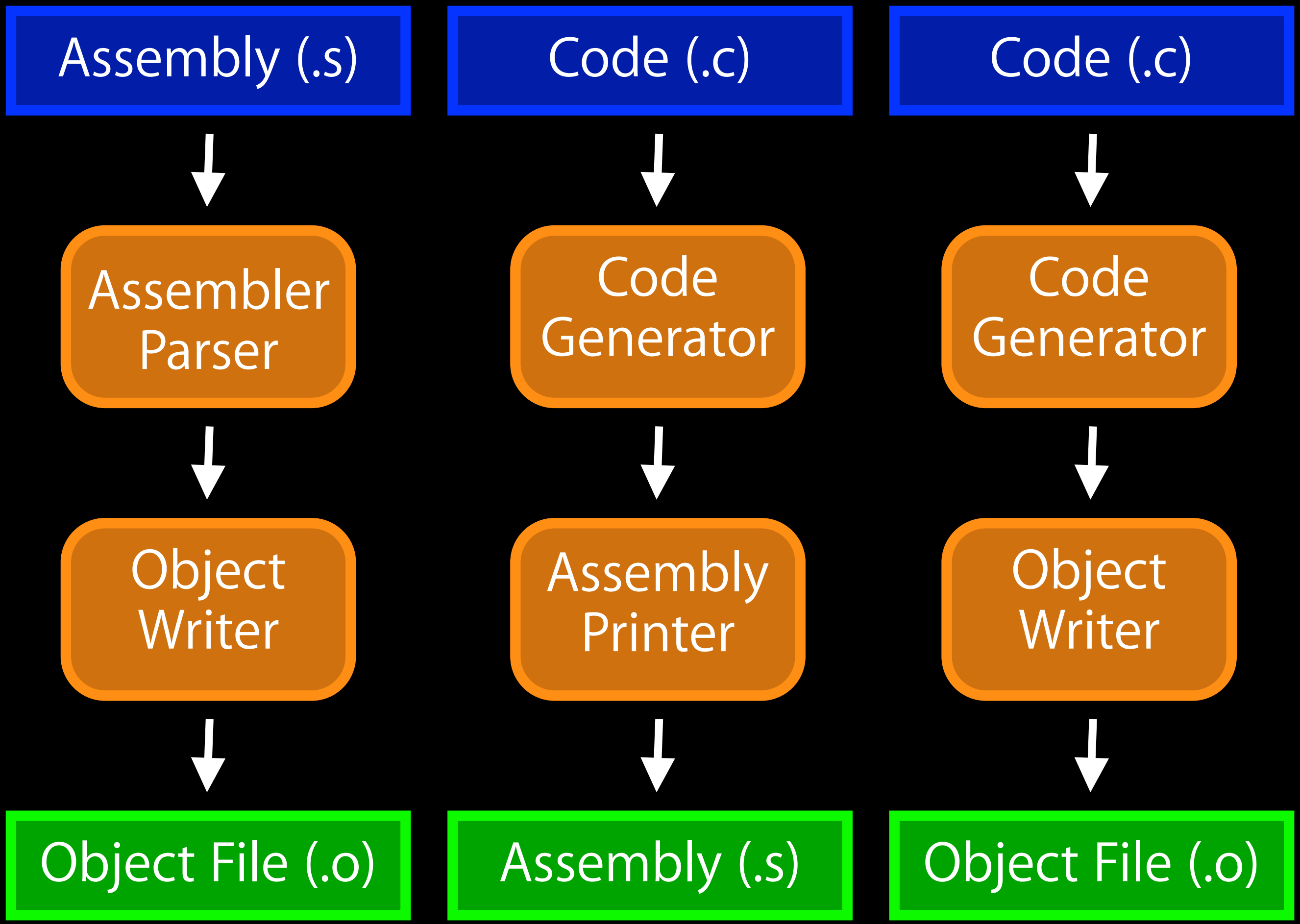
Code (.c) → Code Generator → Assembly Printer → Assembly (.s)

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

It's an Assembler!

It's a Compiler!

How is MC Used?



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

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

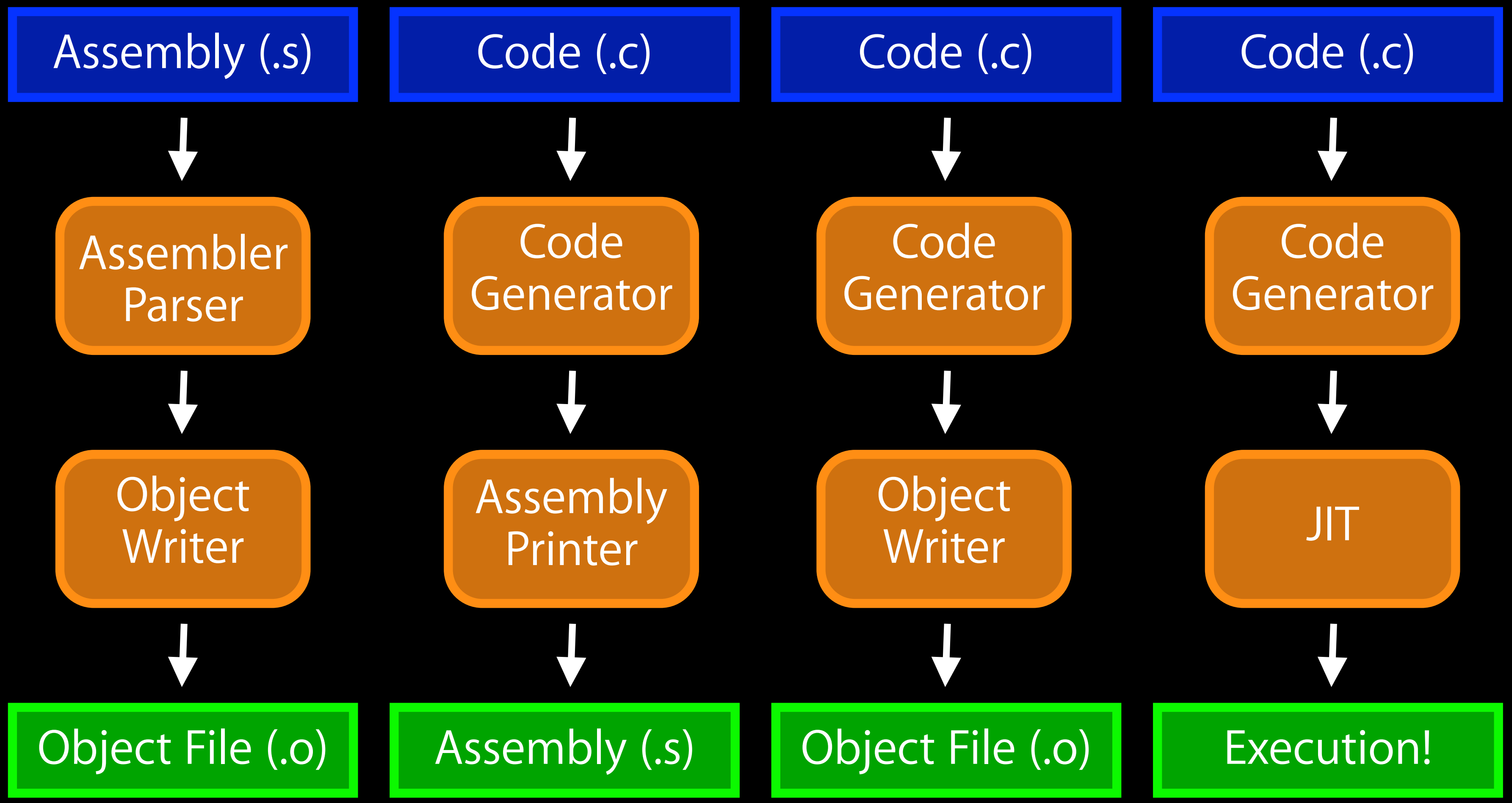
Code (.c) → Code Generator → Object Writer → Object File (.o)

It's an Assembler!

It's a Compiler!

It's a Compilassembler!

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)

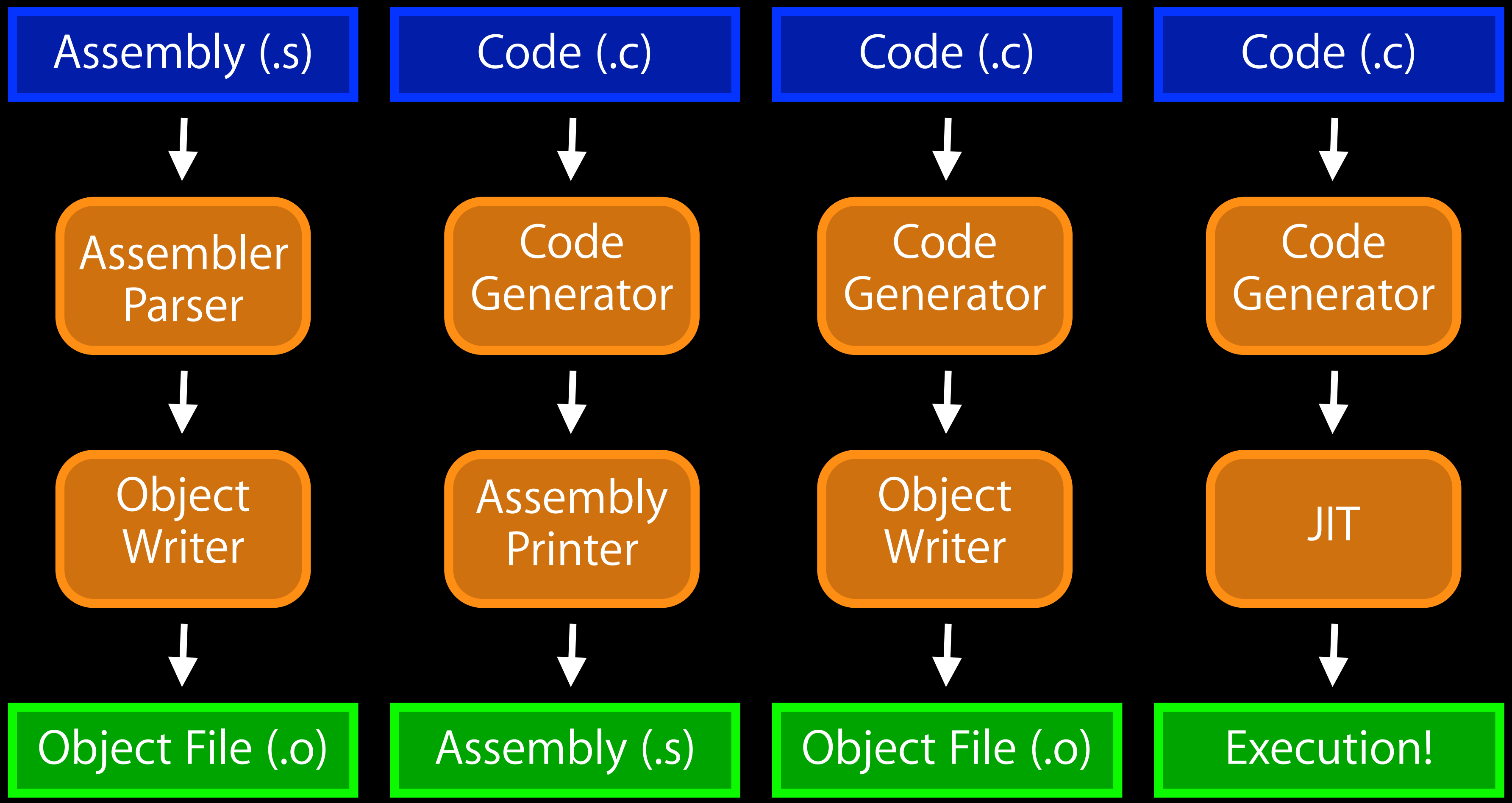
Code (.c) → Code Generator → JIT → Execution!

It’s an Assembler!

It’s a Compiler!

It’s a Compilassembler!

How is MC Used?



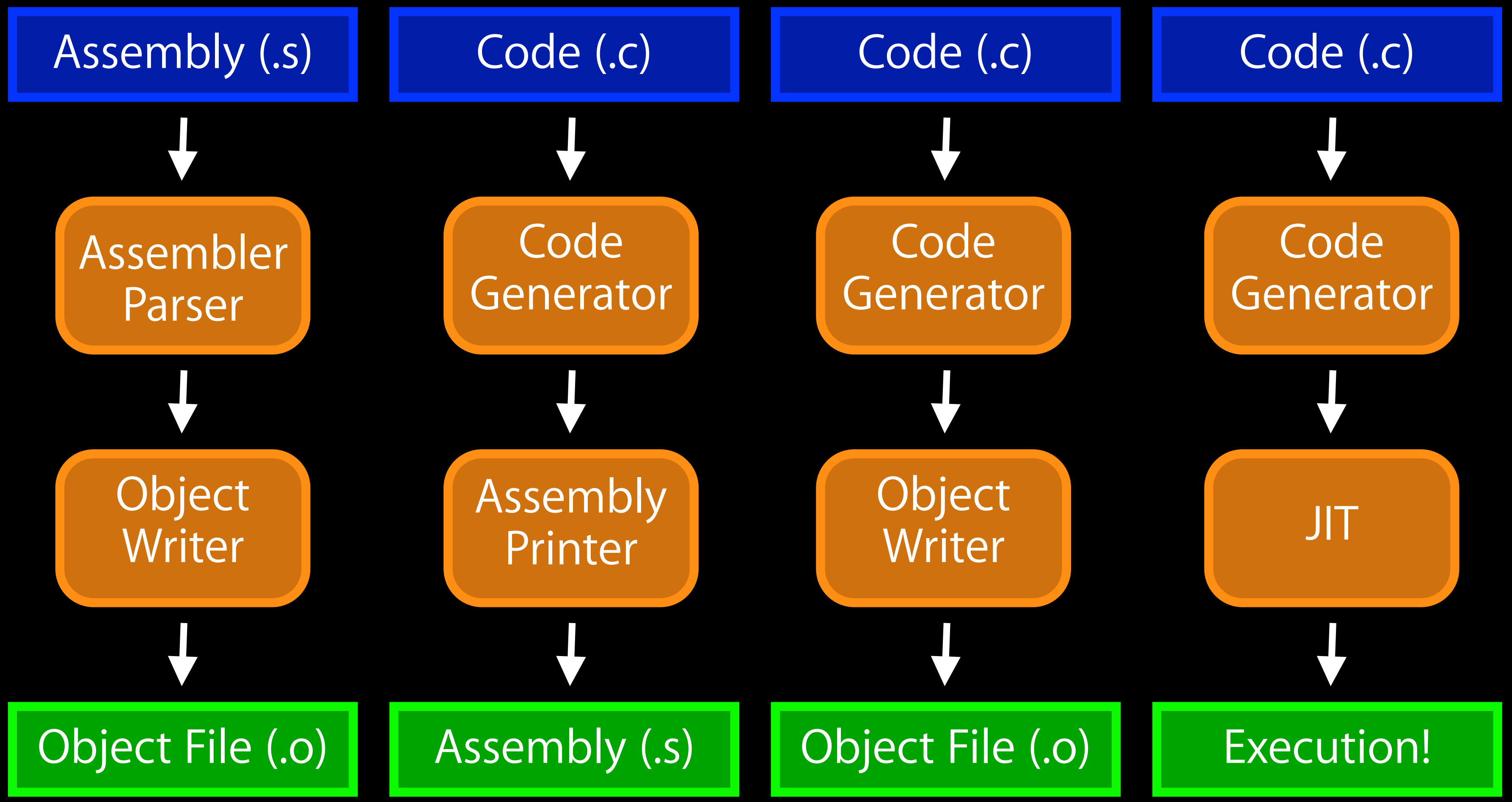
It's an Assembler!

It's a Compiler!

It's a Compilassembler!

It's a JIT!

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!

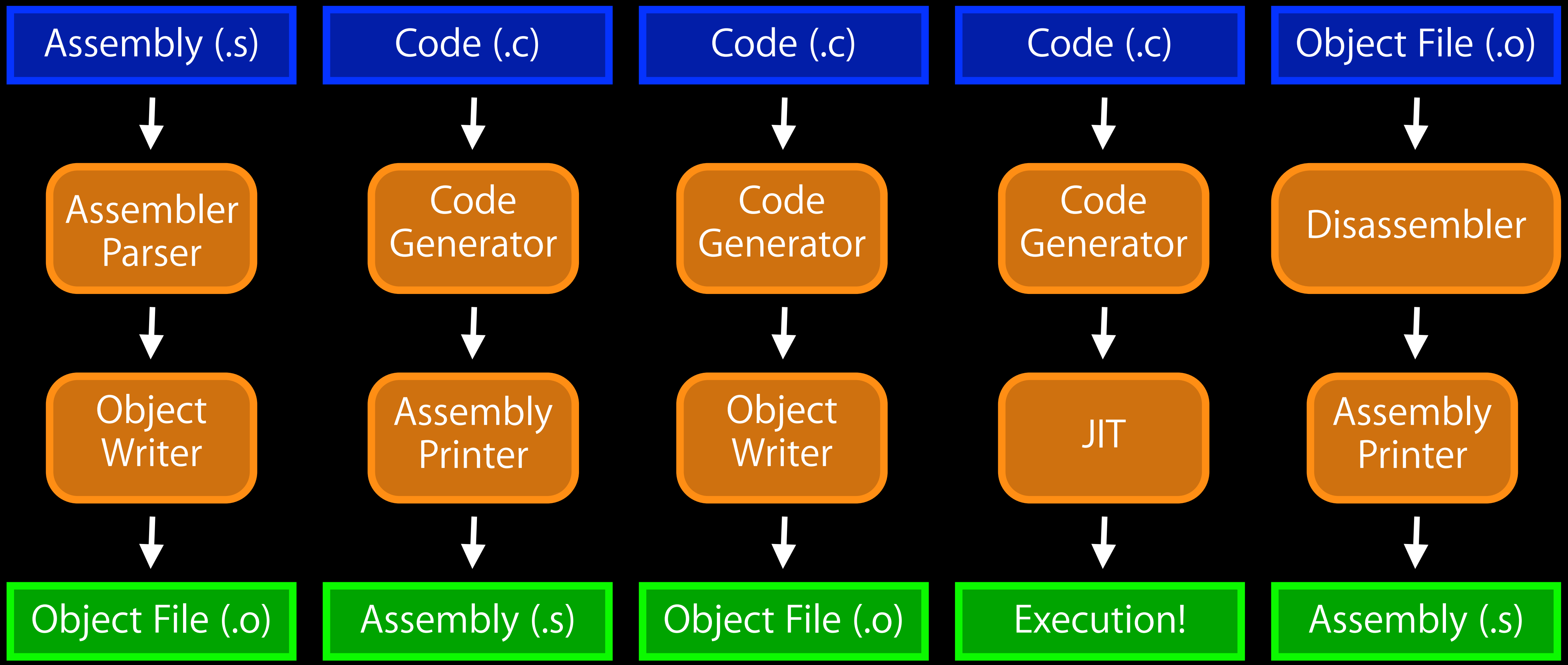
It's an Assembler!

It's a Compiler!

It's a Compilassembler!

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

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)

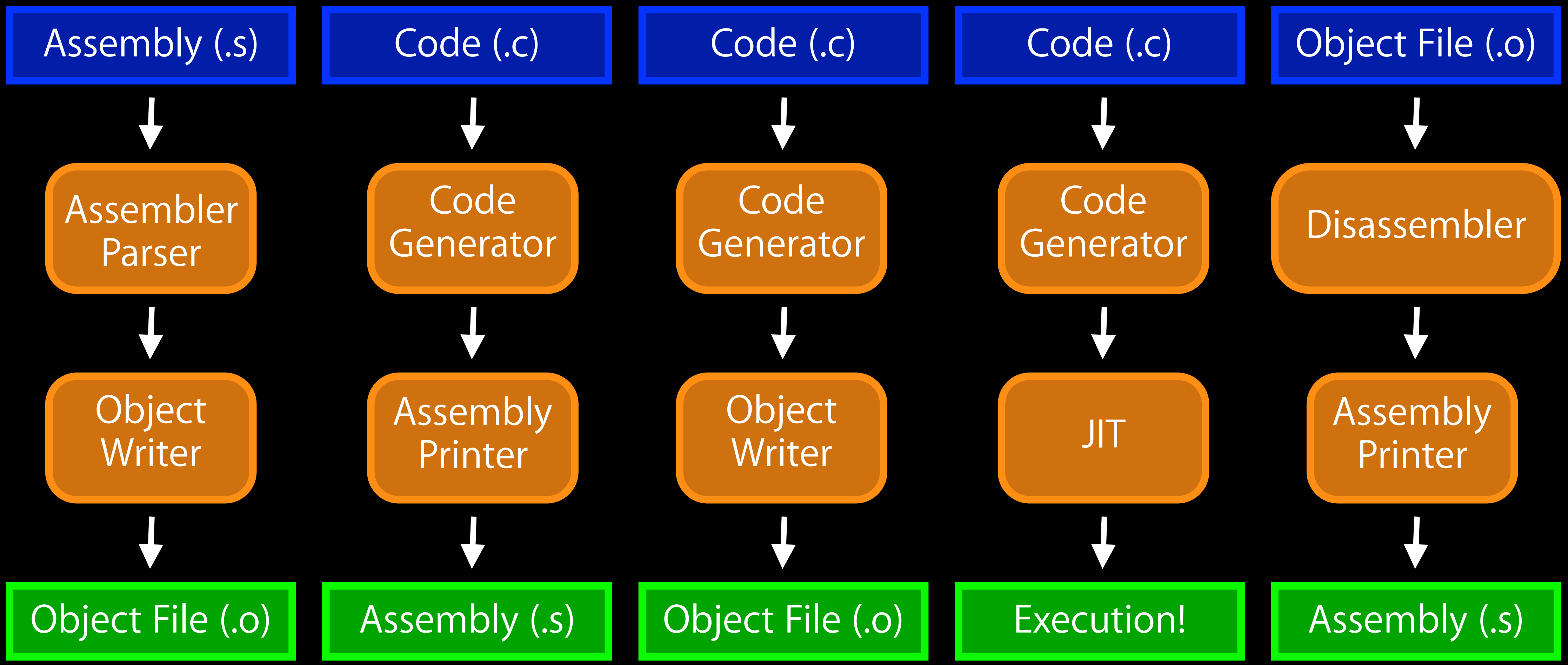
It's an Assembler!

It's a Compiler!

It's a Compilassembler!

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

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)

It's an Assembler!

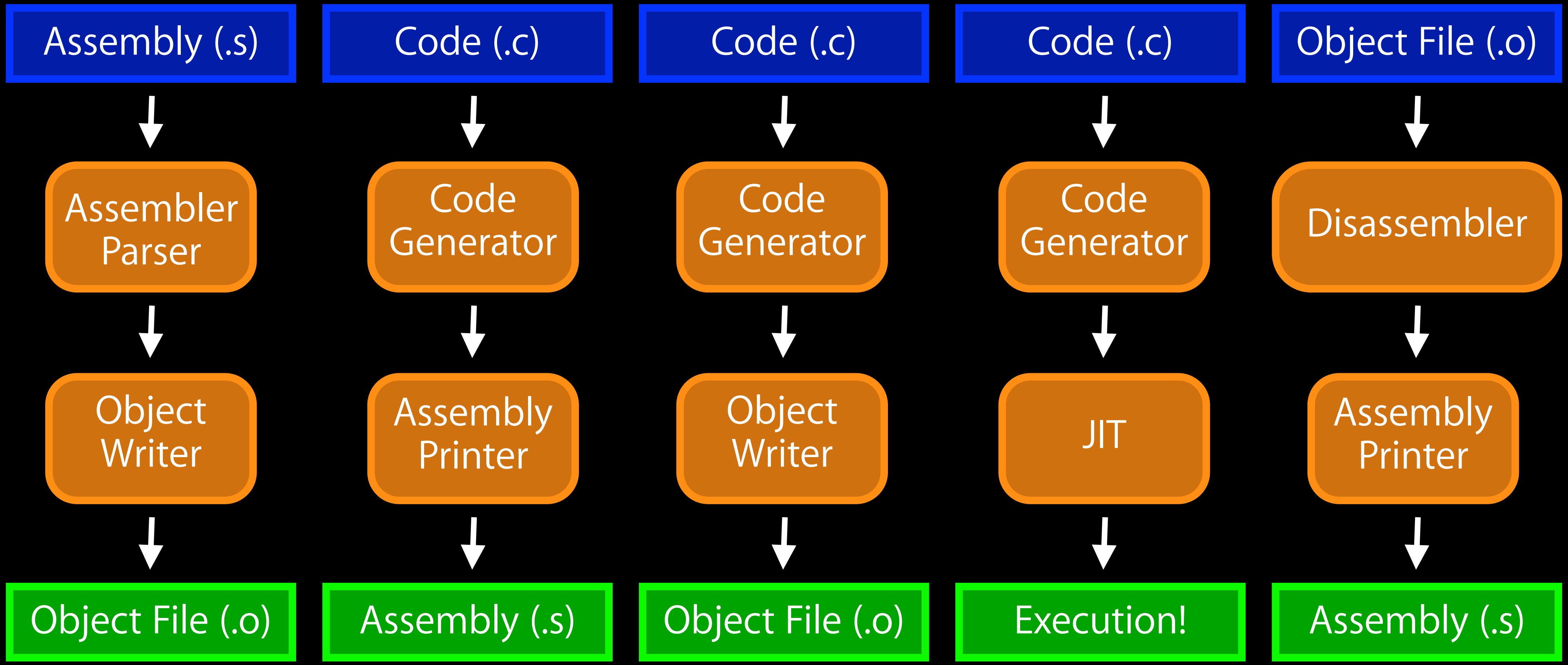
It's a Compiler!

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)

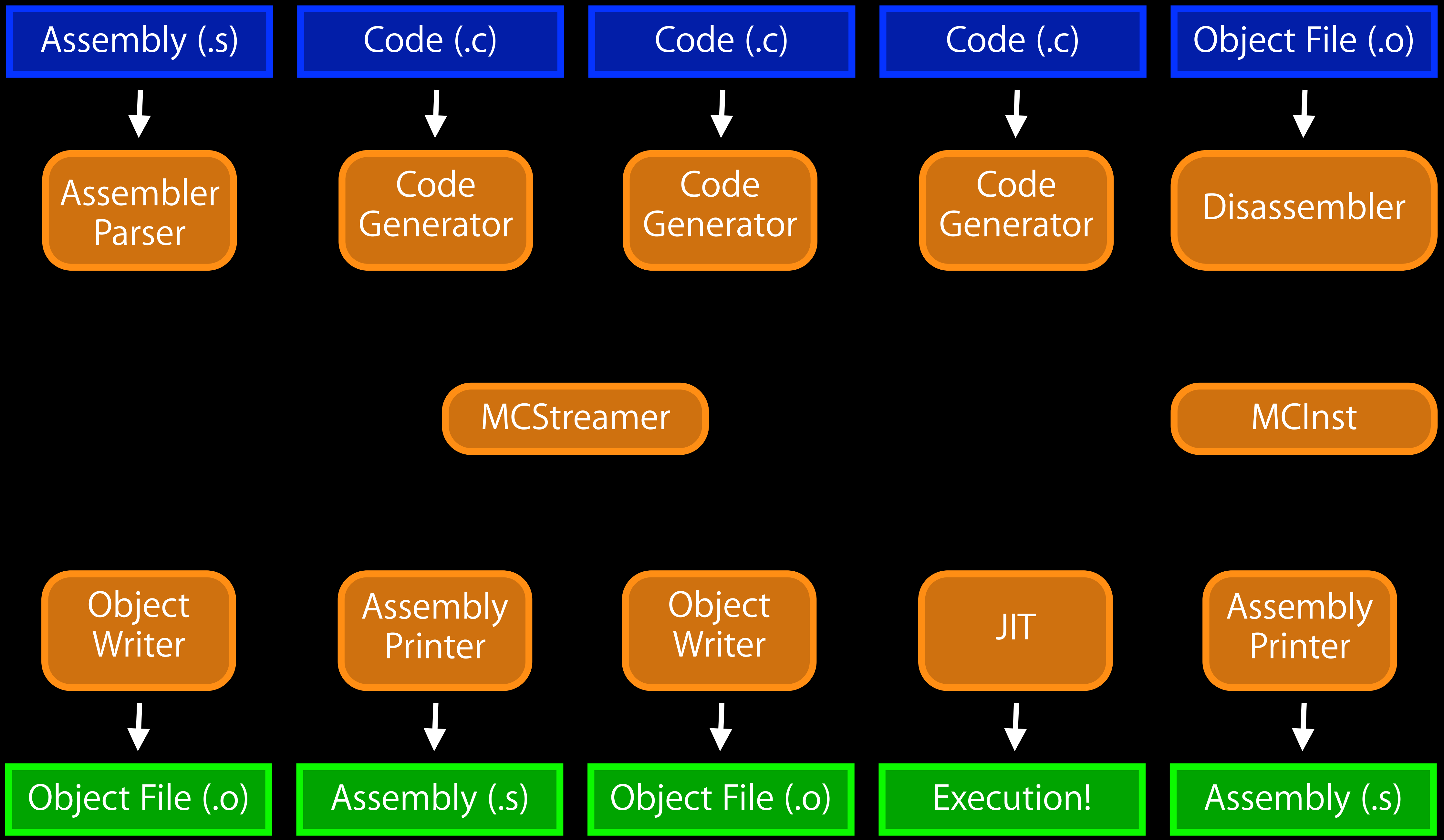
Code (.c) → Code Generator → Assembly Printer → Assembly (.s)

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)
Code (.c)
Code (.c)
Code (.c)
Object File (.o)

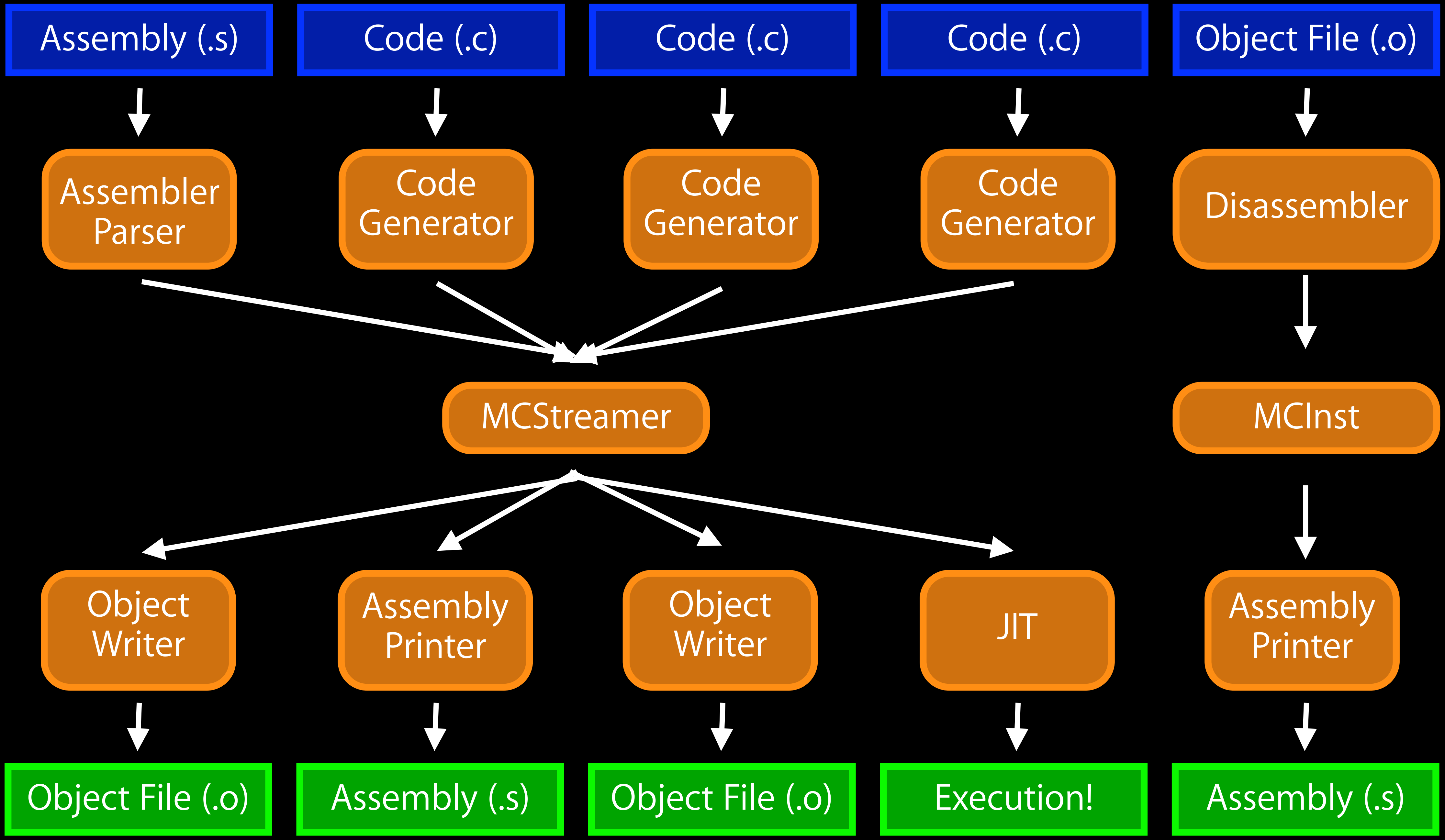
Assembler Parser
Code Generator
Code Generator
Code Generator
Disassembler

MCStreamer
MCInst

Object Writer
Assembly Printer
Object Writer
JIT
Assembly Printer

Object File (.o)
Assembly (.s)
Object File (.o)
Execution!
Assembly (.s)

How is MC Used?



MCStreamer

MCStreamer

- Core MC Component

MCStreamer

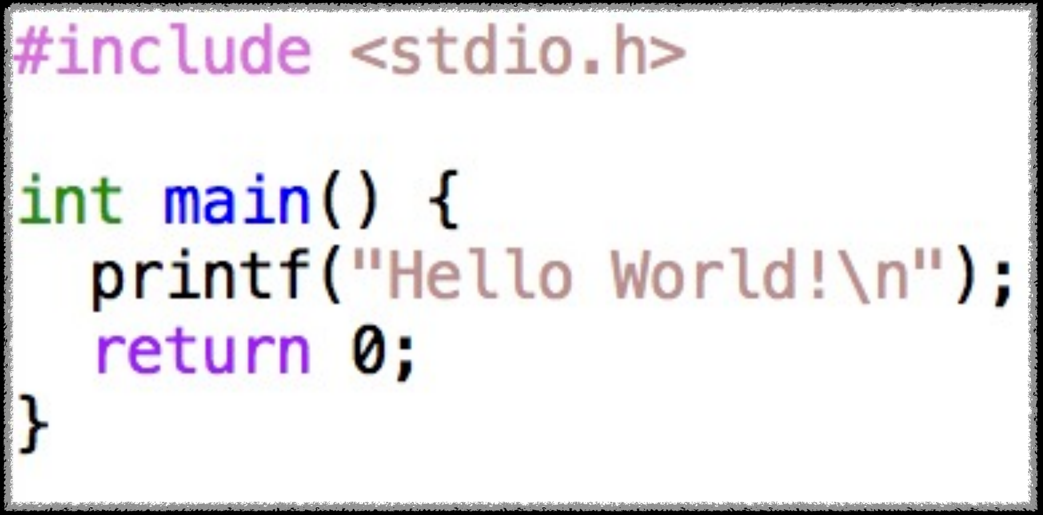
- Core MC Component
  - Programmatic Assembler API

MCStreamer

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

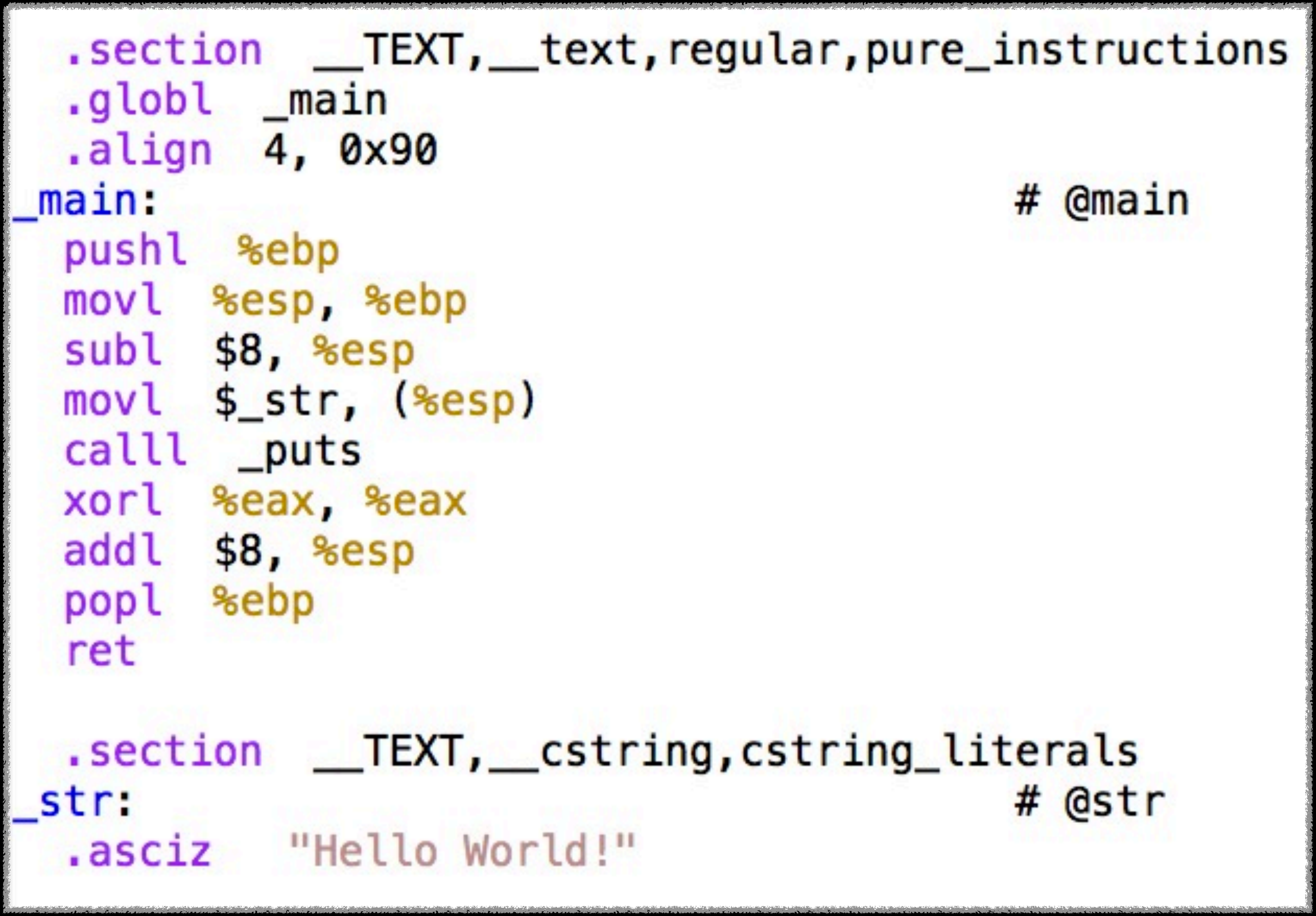
MCStreamer

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



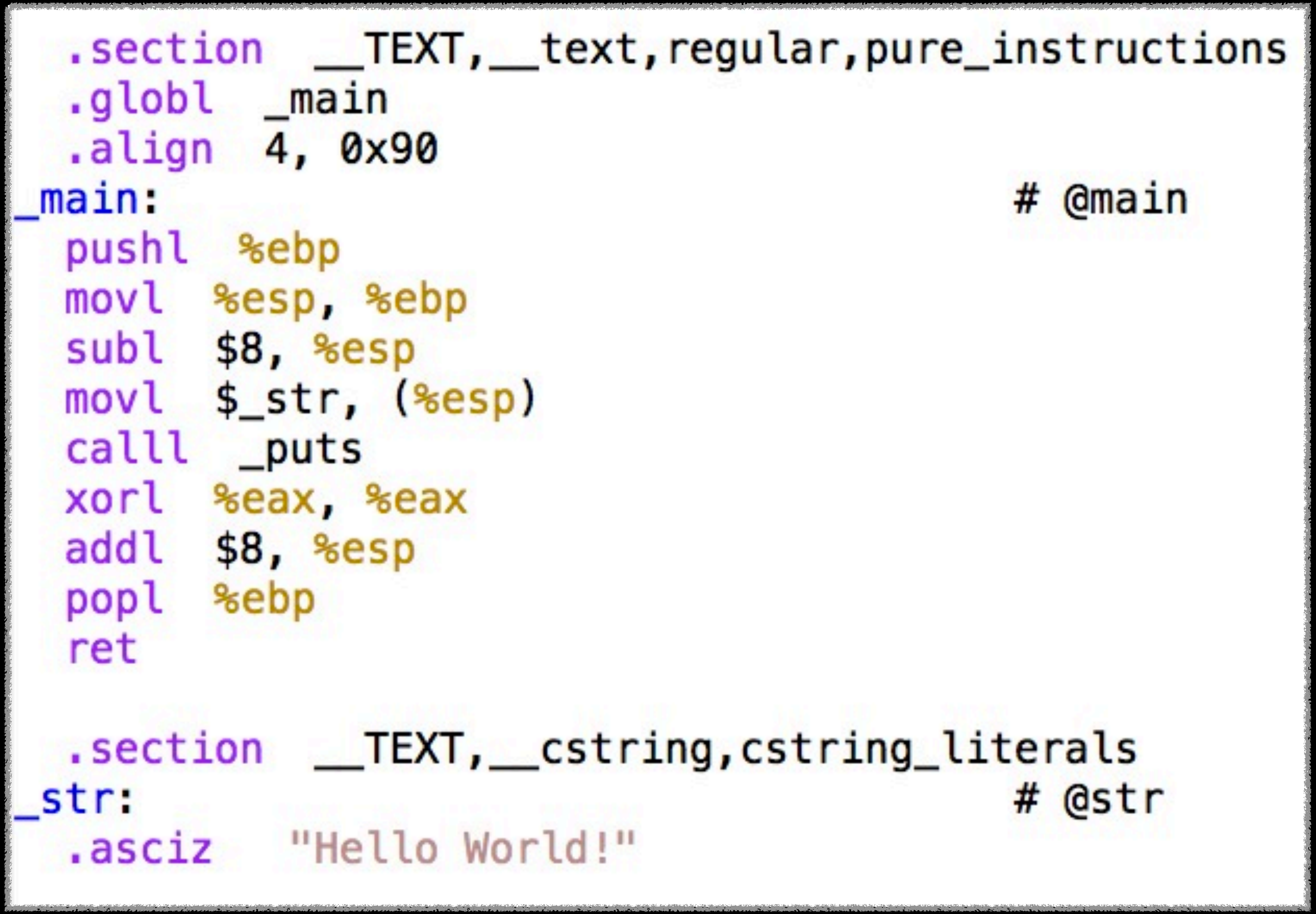
```
#include <stdio.h>

int main() {
  printf("Hello World!\n");
  return 0;
}
```

MCStreamer



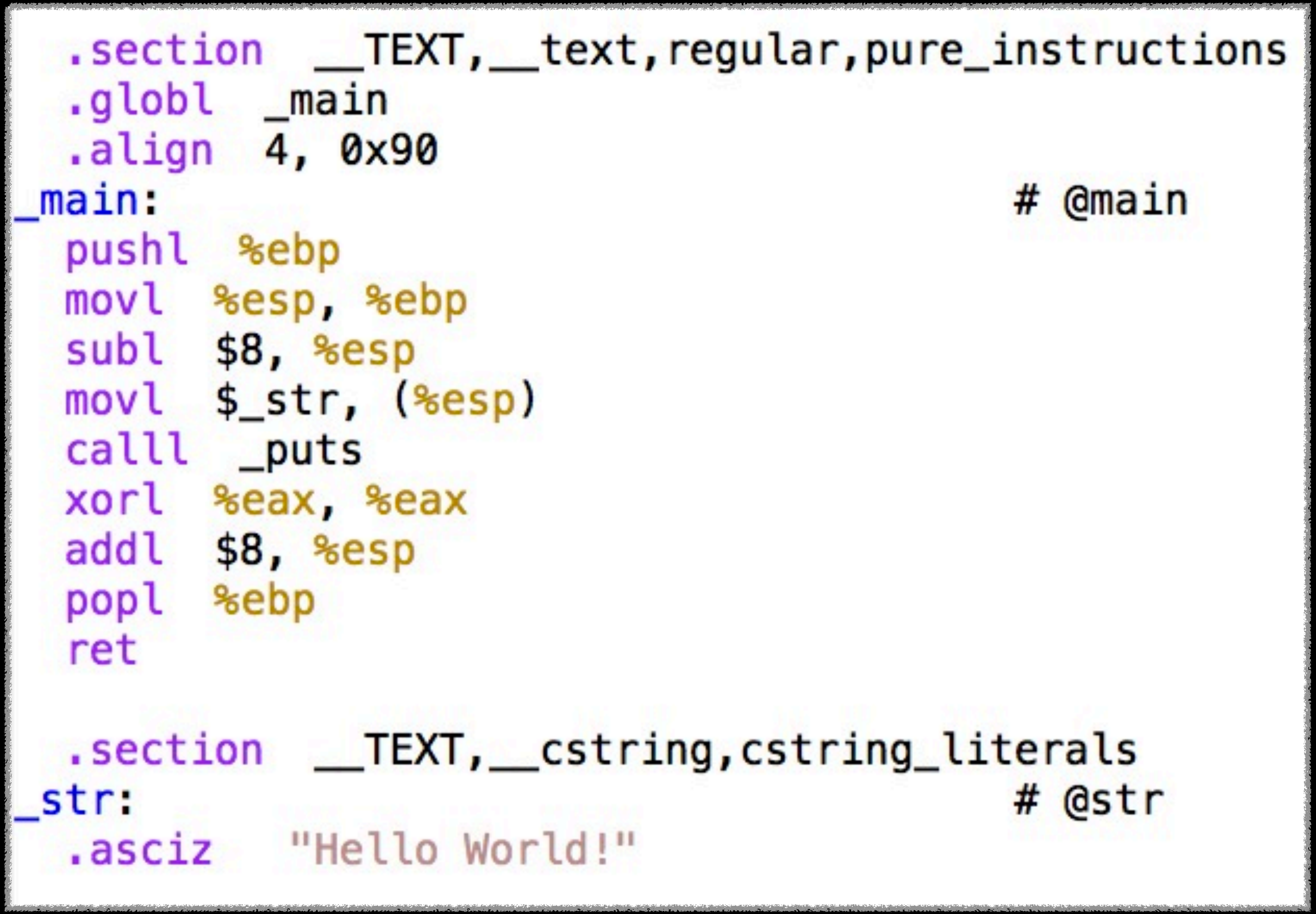
```
  .section  __TEXT,__text,regular,pure_instructions
  .globl  _main
  .align  4, 0x90
_main:                                            # @main
  pushl  %ebp
  movl  %esp, %ebp
  subl  $8, %esp
  movl  $_str, (%esp)
  calll  _puts
  xorl  %eax, %eax
  addl  $8, %esp
  popl  %ebp
  ret

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

MCStreamer



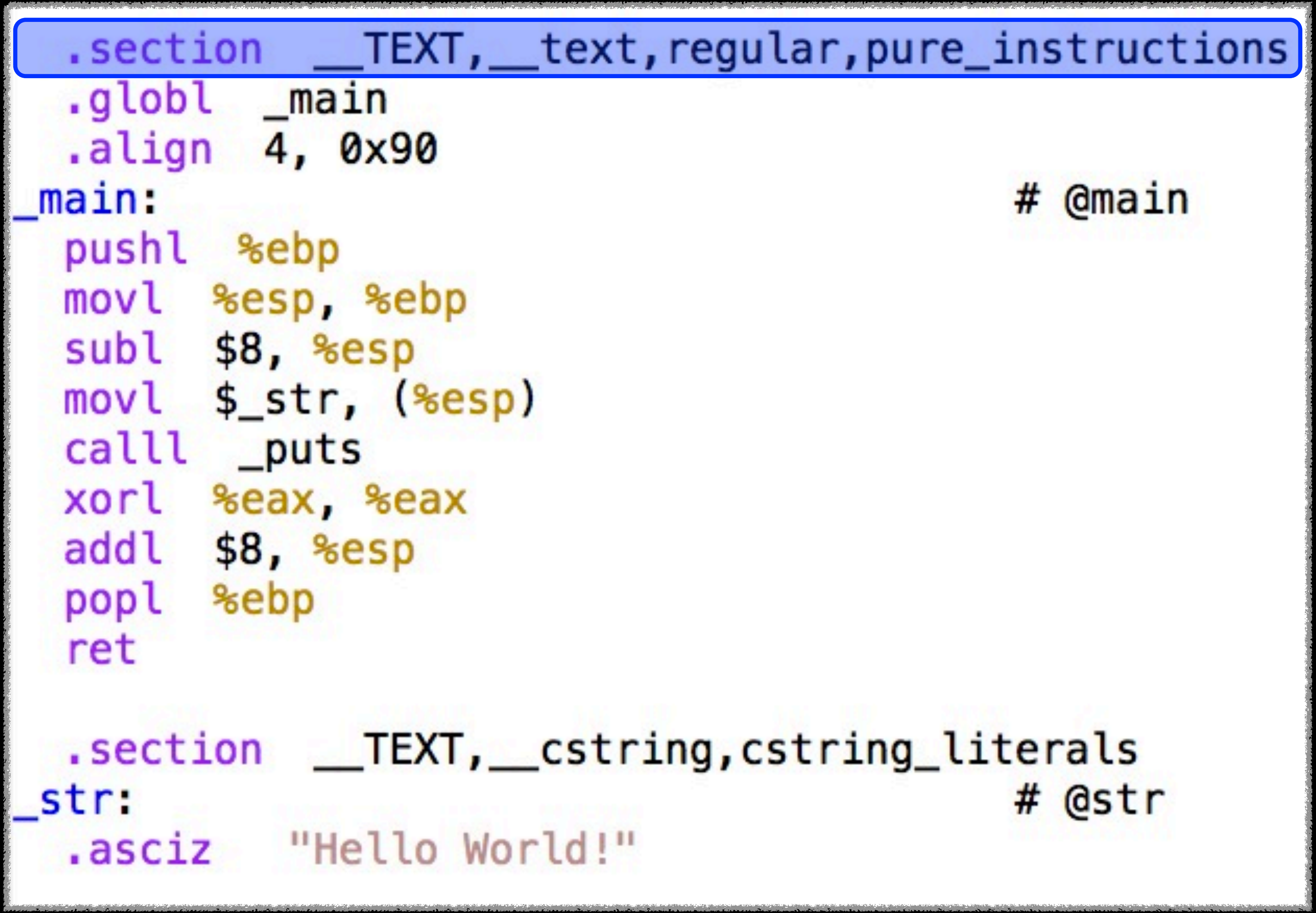
```
  .section  __TEXT,__text,regular,pure_instructions
  .globl  _main
  .align  4, 0x90
_main:                                  # @main
  pushl  %ebp
  movl  %esp, %ebp
  subl  $8, %esp
  movl  $_str, (%esp)
  calll  _puts
  xorl  %eax, %eax
  addl  $8, %esp
  popl  %ebp
  ret

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

MCStreamer



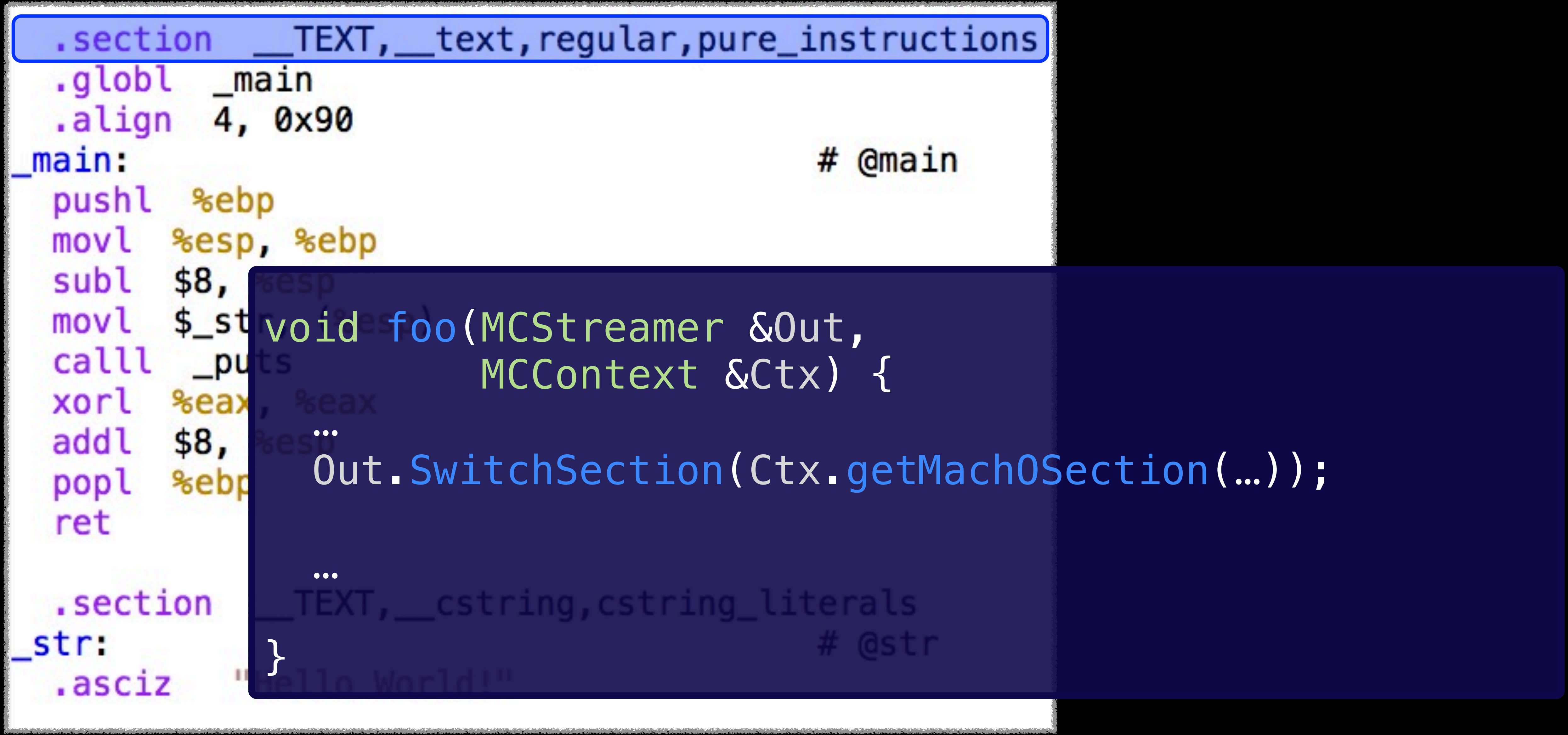
```
  .section  __TEXT,__text,regular,pure_instructions
  .globl  _main
  .align  4, 0x90
_main:                                  # @main
  pushl  %ebp
  movl  %esp, %ebp
  subl  $8, %esp
  movl  $_str, (%esp)
  calll  _puts
  xorl  %eax, %eax
  addl  $8, %esp
  popl  %ebp
  ret

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

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!"
```

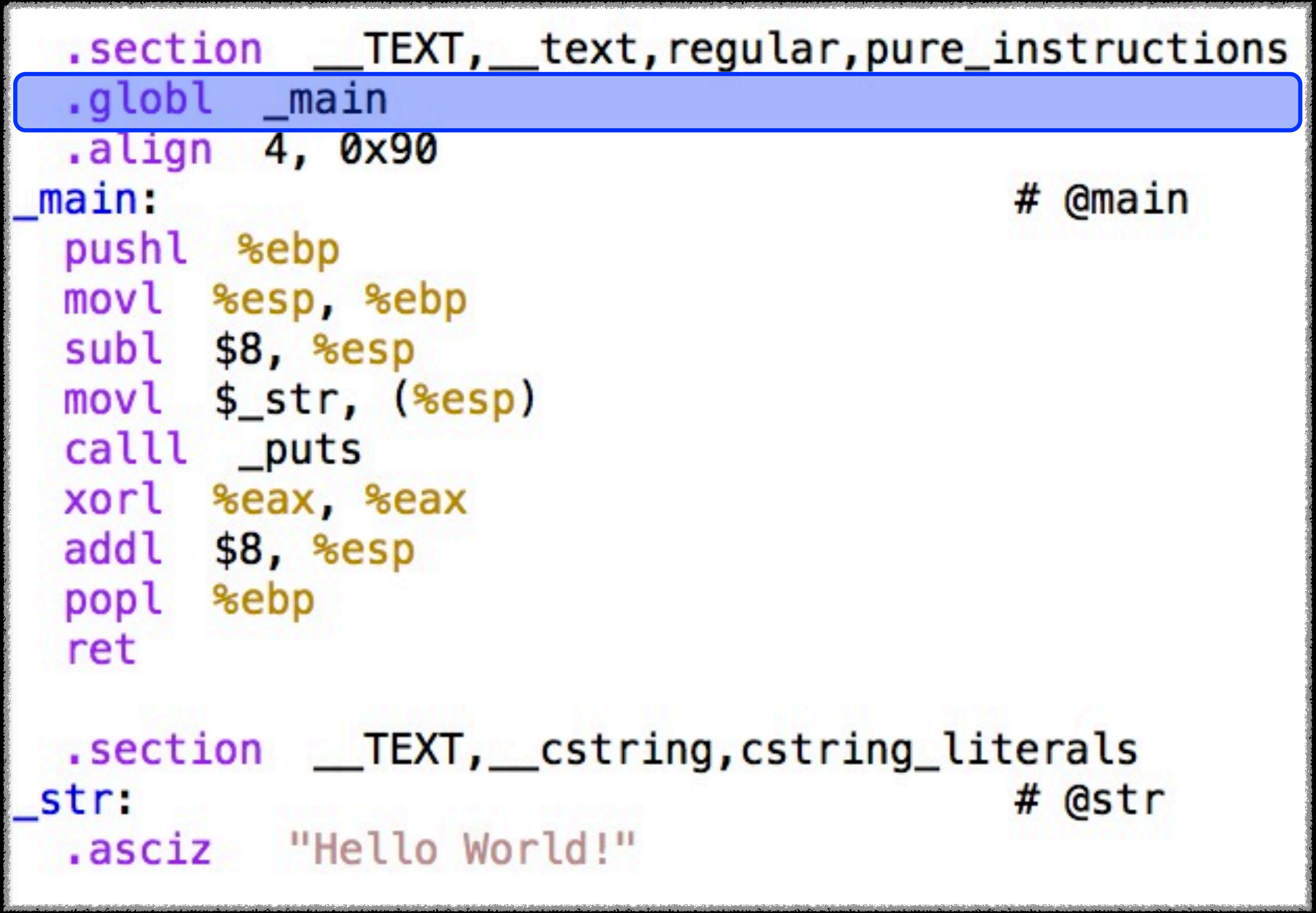
MCStreamer



```
	.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!"
```

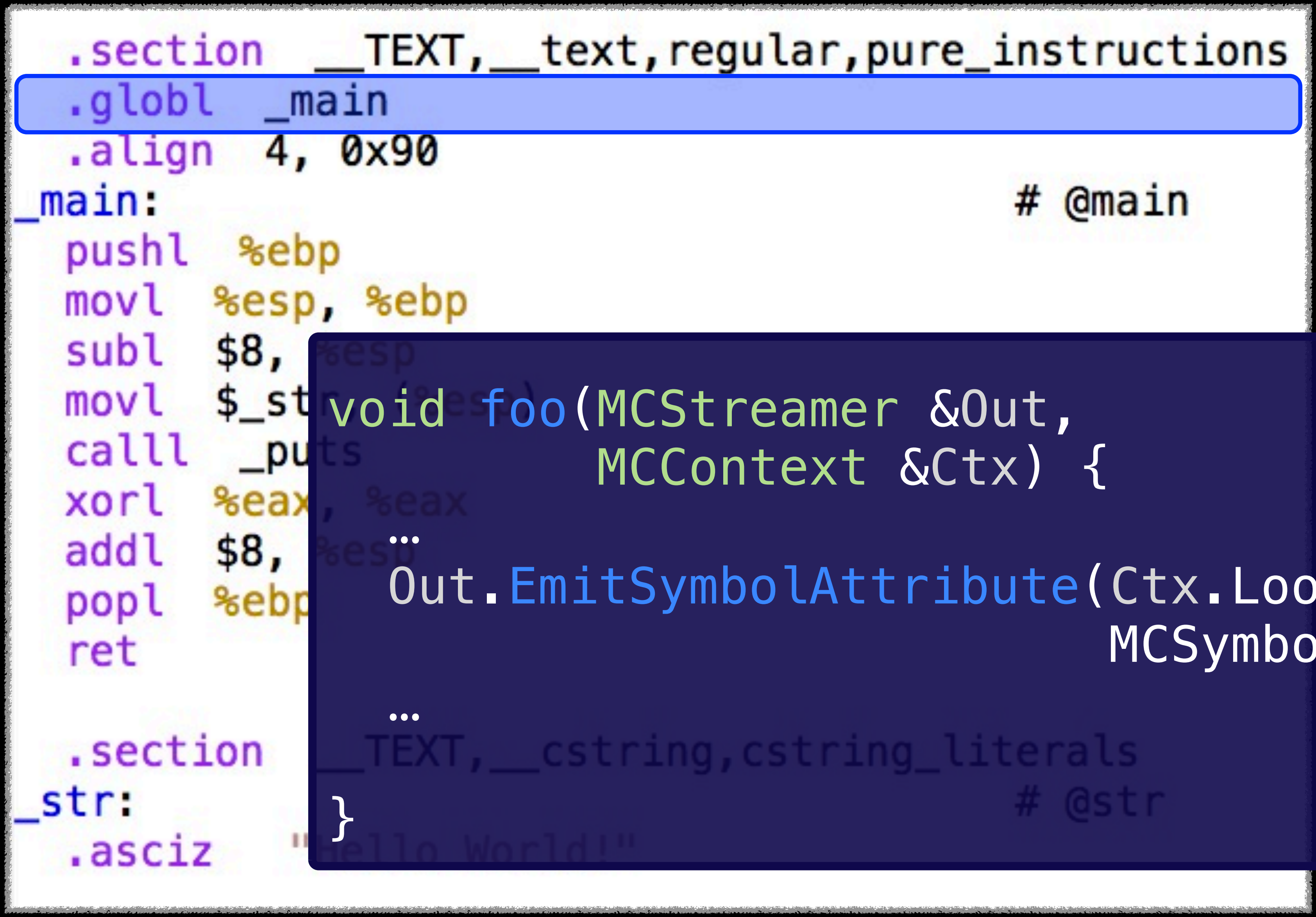
```
void foo(MCStreamer &Out,
         MCContext &Ctx) {
  …
  Out.SwitchSection(Ctx.getMachOSection(…));

  …
}
```

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!"
```

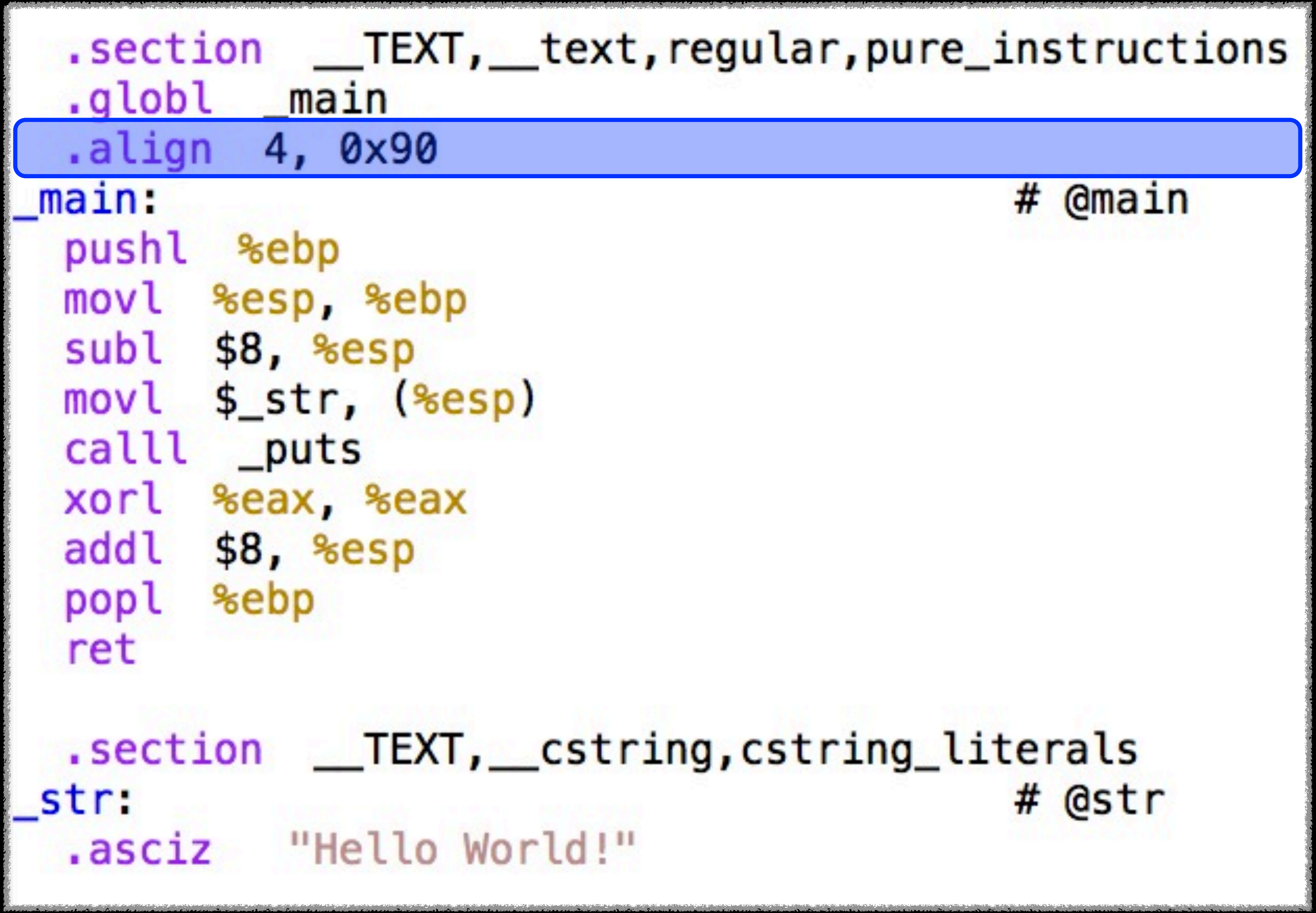
MCStreamer



```
  .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!"
```

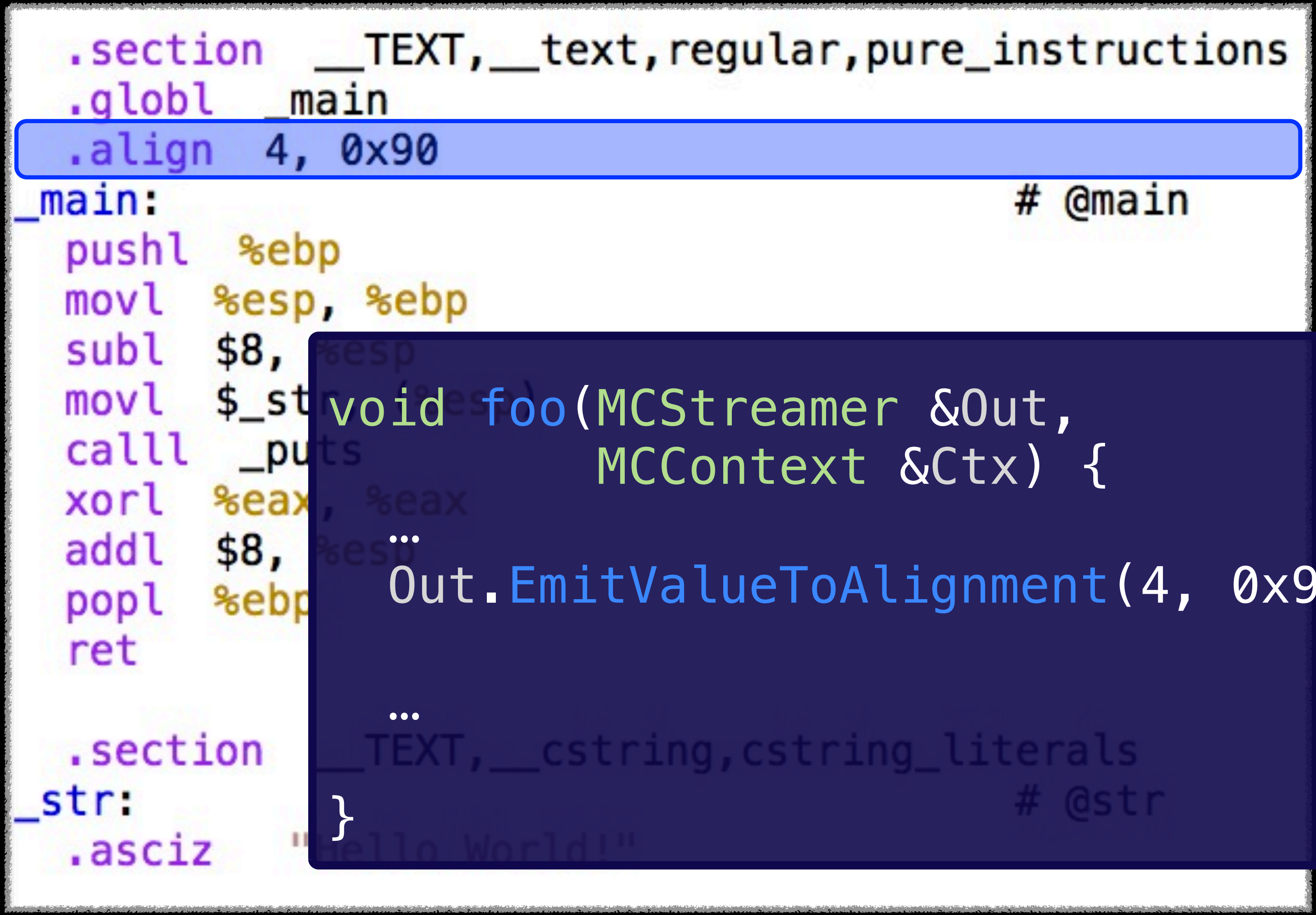
```
void foo(MCStreamer &Out,
         MCContext &Ctx) {
  …
  Out.EmitSymbolAttribute(Ctx.LookupSymbol(“_main”),
                          MCSymbolAttr::MCSA_Global);
  …
}
```

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!"
```

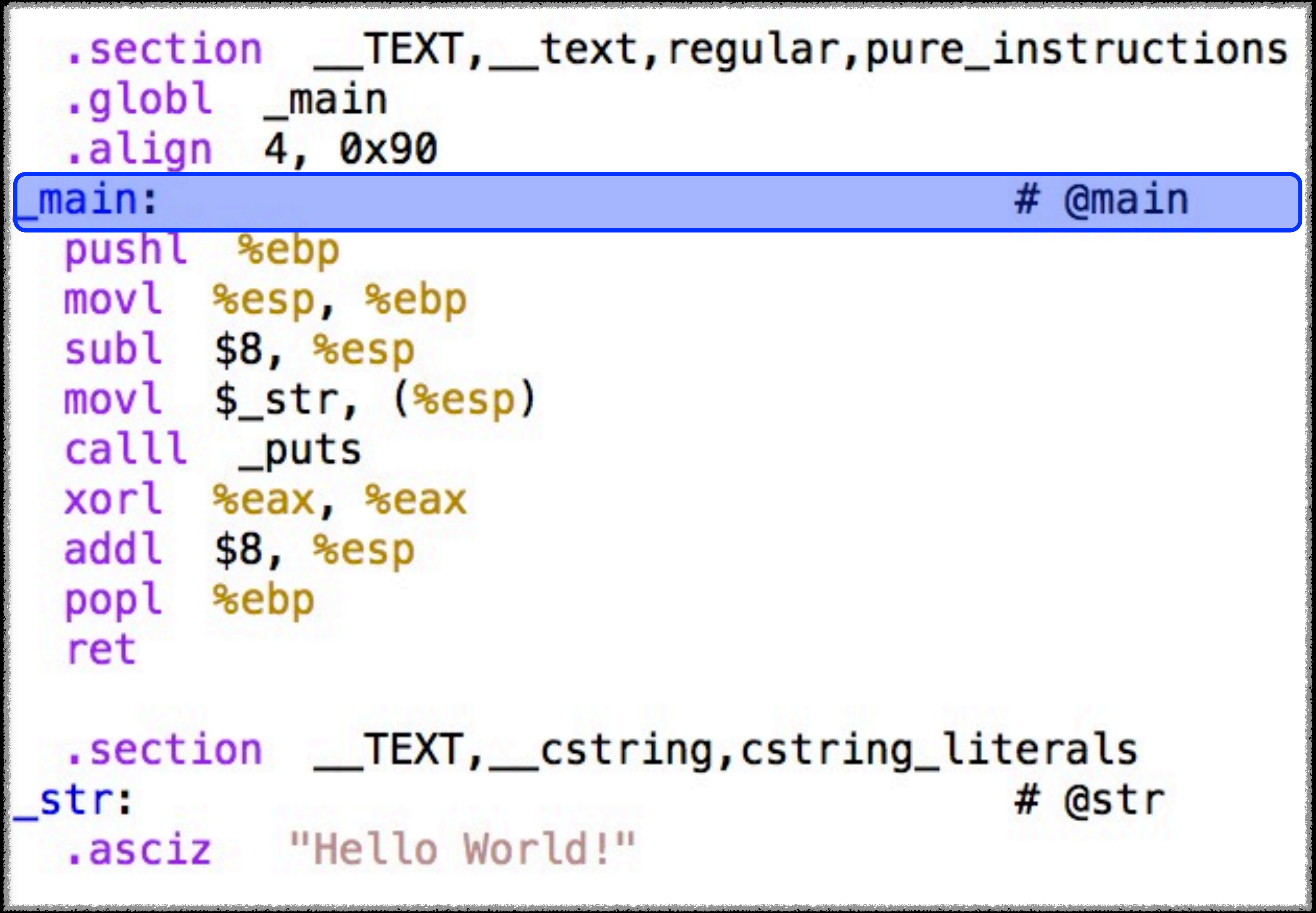
MCStreamer



```
  .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!"
```

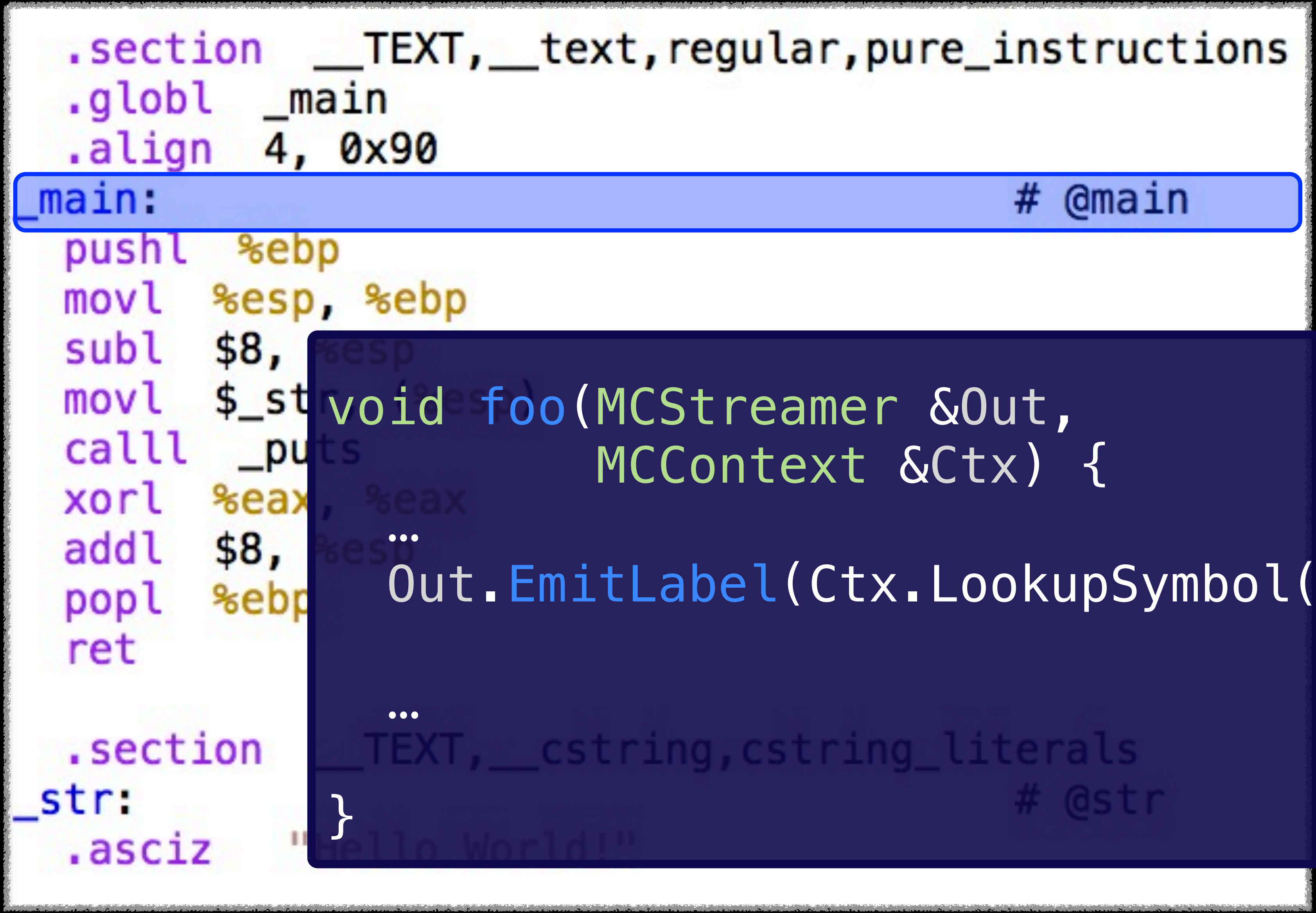
```
void foo(MCStreamer &Out,
         MCContext &Ctx) {
  …
  Out.EmitValueToAlignment(4, 0x90);

  …
}
```

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!"
```

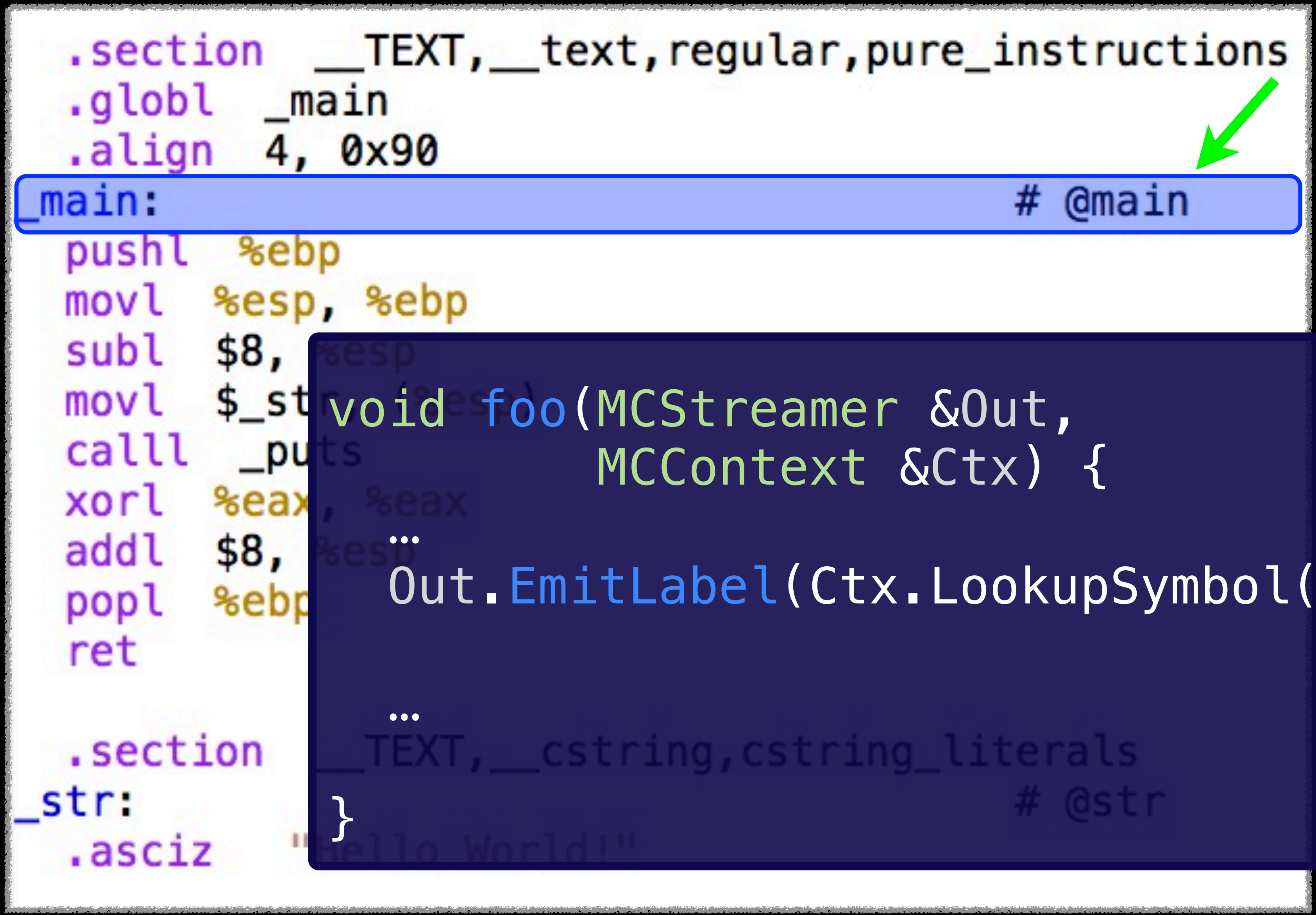
MCStreamer



```
  .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.EmitLabel(Ctx.LookupSymbol(“_main”));

  …
}
```

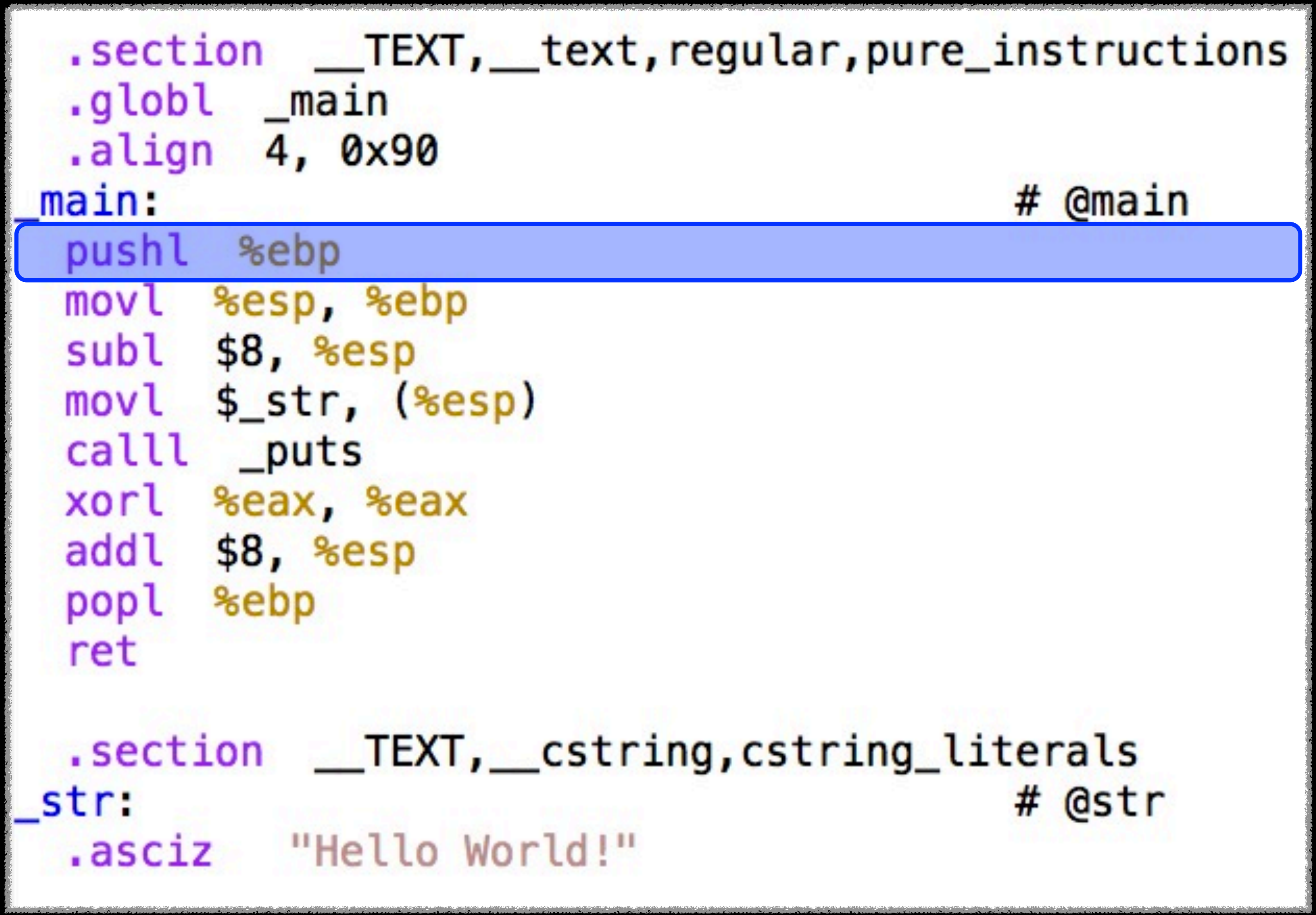
MCStreamer



```
    .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!"
```

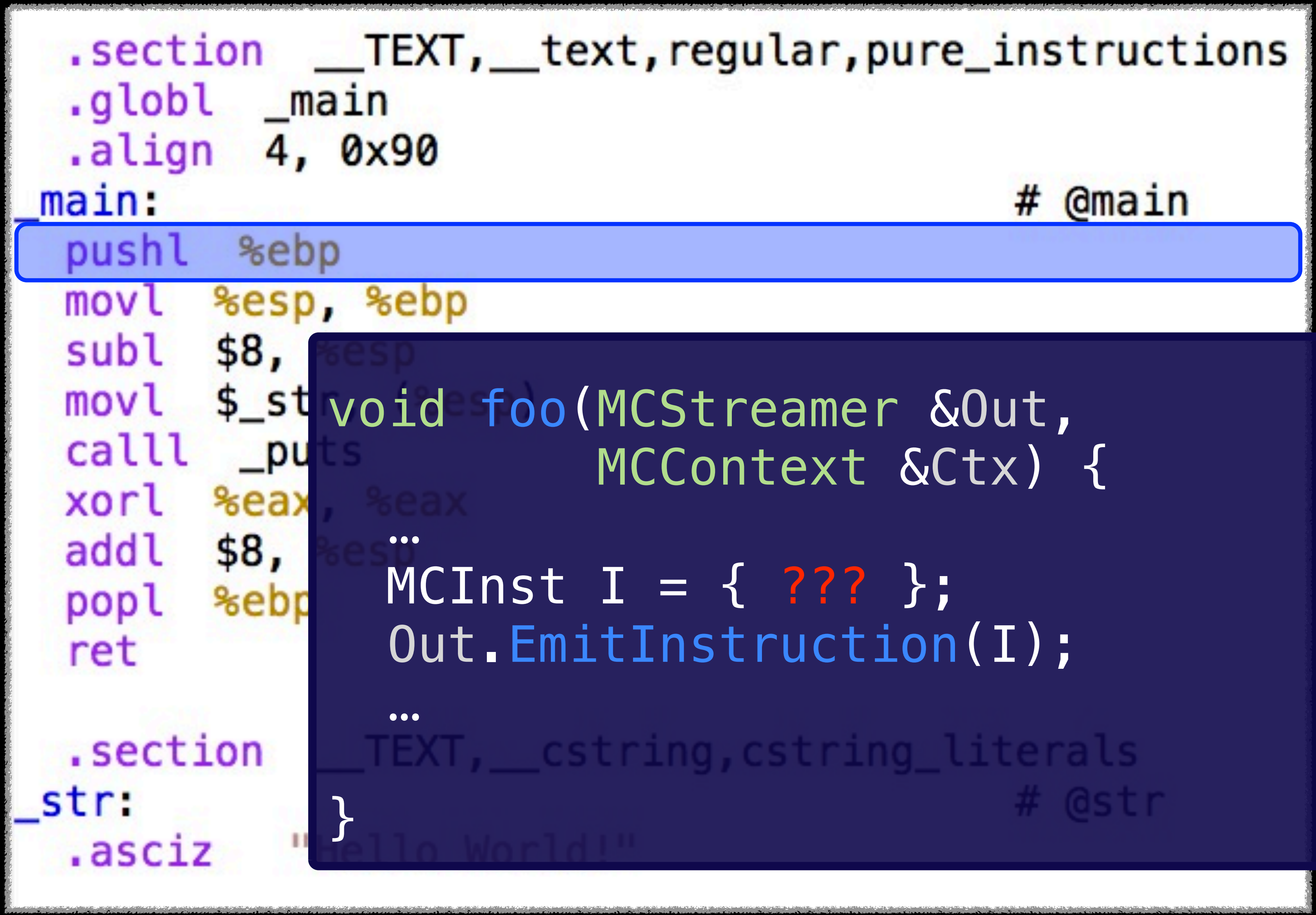
```
void foo(MCStreamer &Out,
         MCContext &Ctx) {
  …
  Out.EmitLabel(Ctx.LookupSymbol("_main"));


  …

}
```

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



```
	.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

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 `llvm-mc` tool

The `llvm-mc` tool

- `llvm-mc` is the command line tool for testing MC

The `llvm-mc` tool

- `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

```
$ 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

```
…
{ X86::PUSHF16,   "pushfw", Convert,         { },           0 },
{ X86::PUSH32r,   "pushl",  Convert__Reg1_0, { MCK_GR32 }, 0 },
{ X86::PUSH32rmr, "pushl",  Convert__Reg1_0, { MCK_GR32 }, 0 },
{ X86::PUSHCS32,  "pushl",  Convert,         { MCK_CS },
                                         Feature_In32BitMode },
…
```

Instruction Matching

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

```
…
{ X86::PUSHF16,   "pushfw", Convert,         { },           0 },
{ X86::PUSH32r,   "pushl",  Convert__Reg1_0, { MCK_GR32 }, 0 },
{ X86::PUSH32rmr, "pushl",  Convert__Reg1_0, { MCK_GR32 }, 0 },
{ X86::PUSHCS32,  "pushl",  Convert,         { MCK_CS },
                                         Feature_In32BitMode },
…
```

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

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 `-O0 -g` for i386

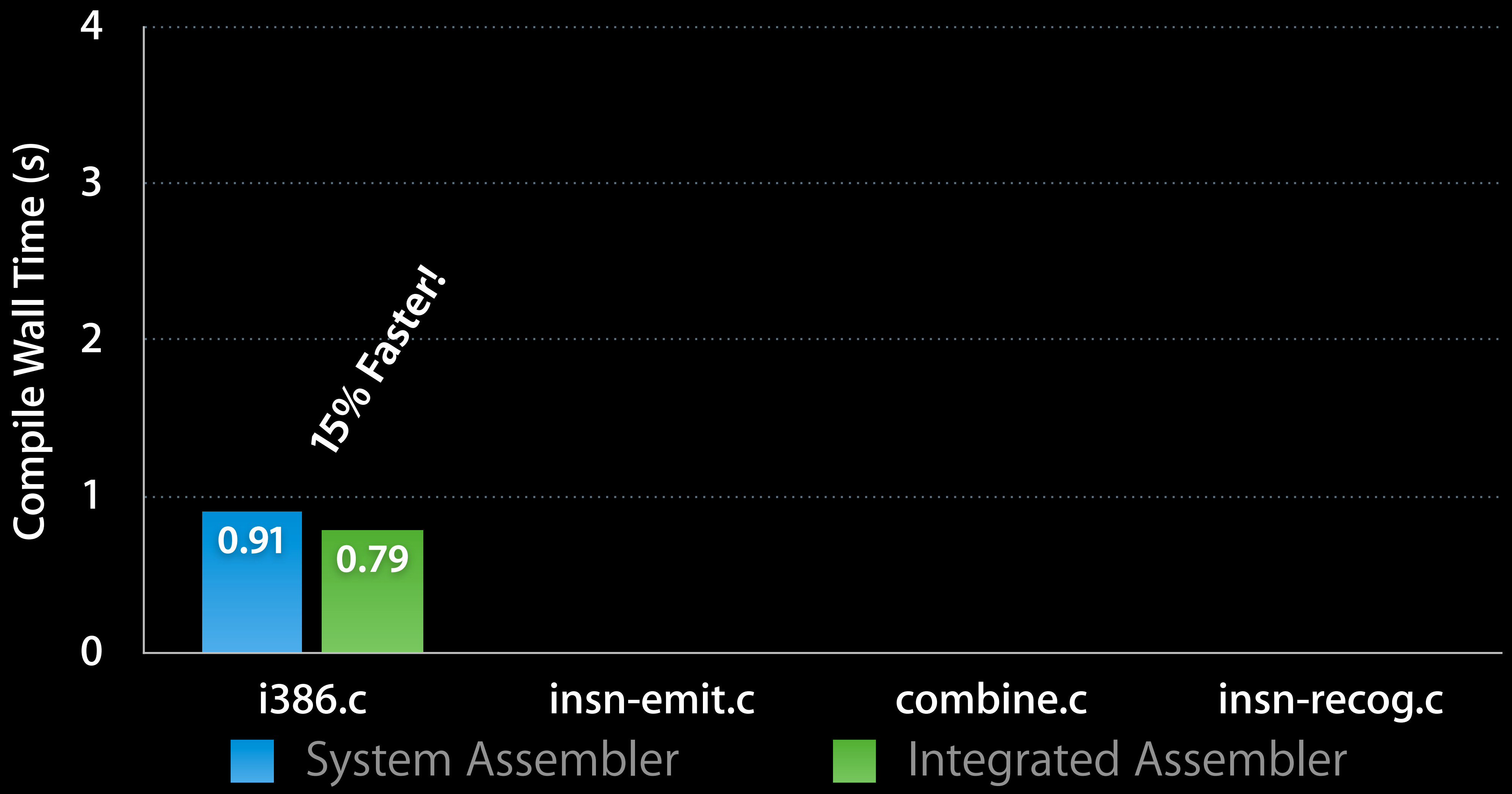
Current Status: Performance

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

Compile Wall Time (s)
4
3
2
1
0
i386.c
insn-emit.c
combine.c
insn-recog.c
🟦 System Assembler
🟩 Integrated Assembler

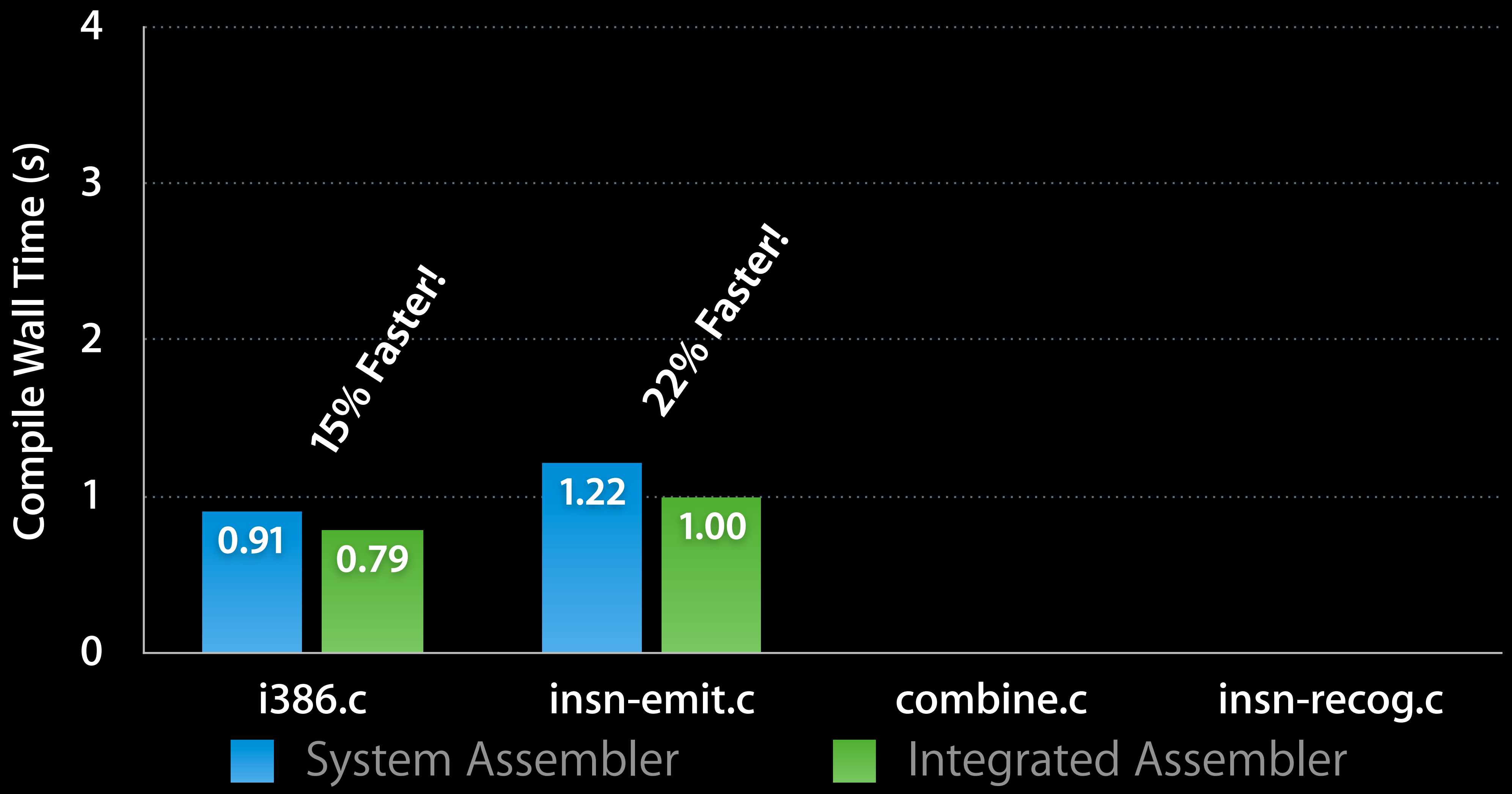
Current Status: Performance

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



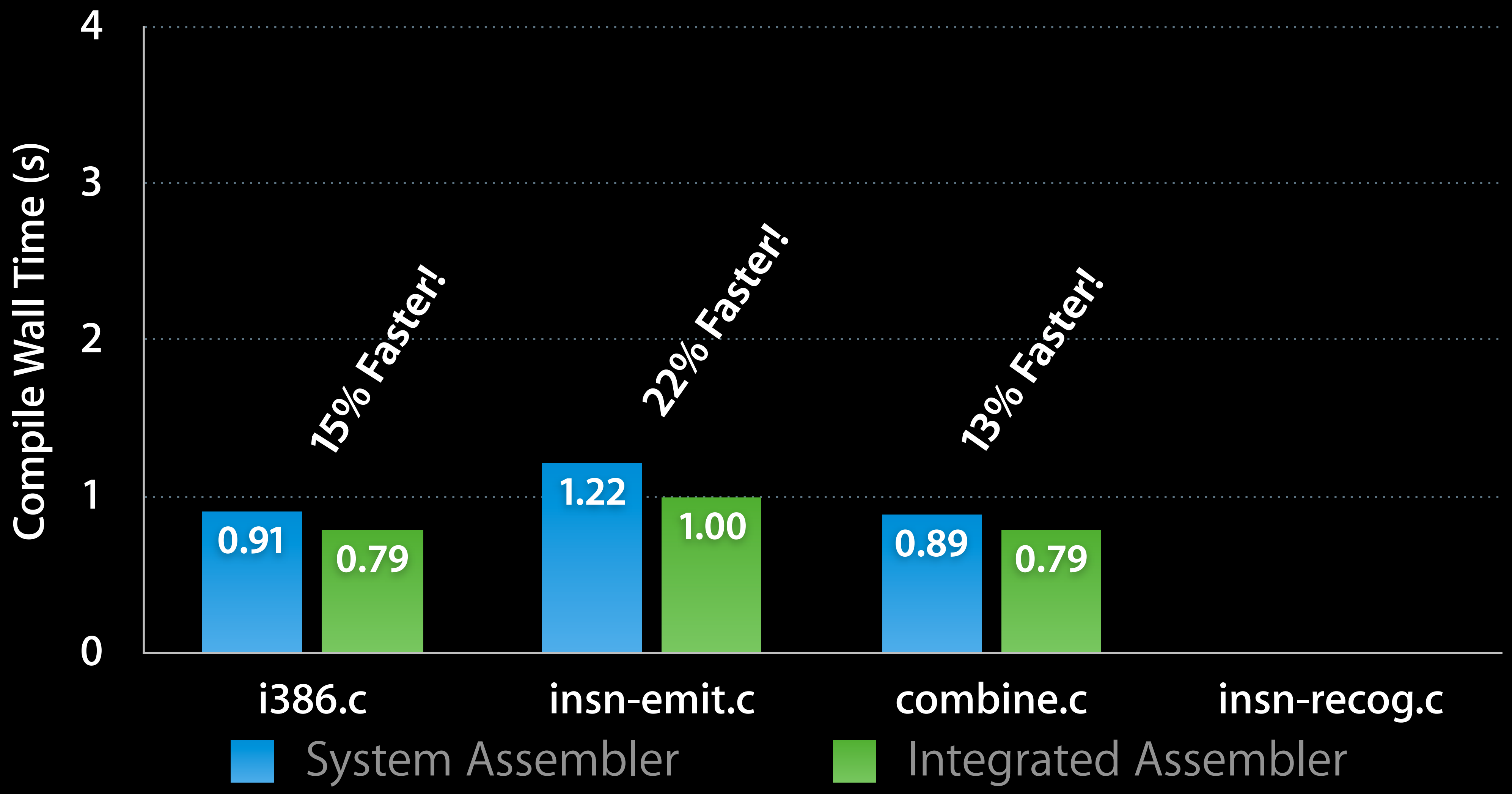
Current Status: Performance

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



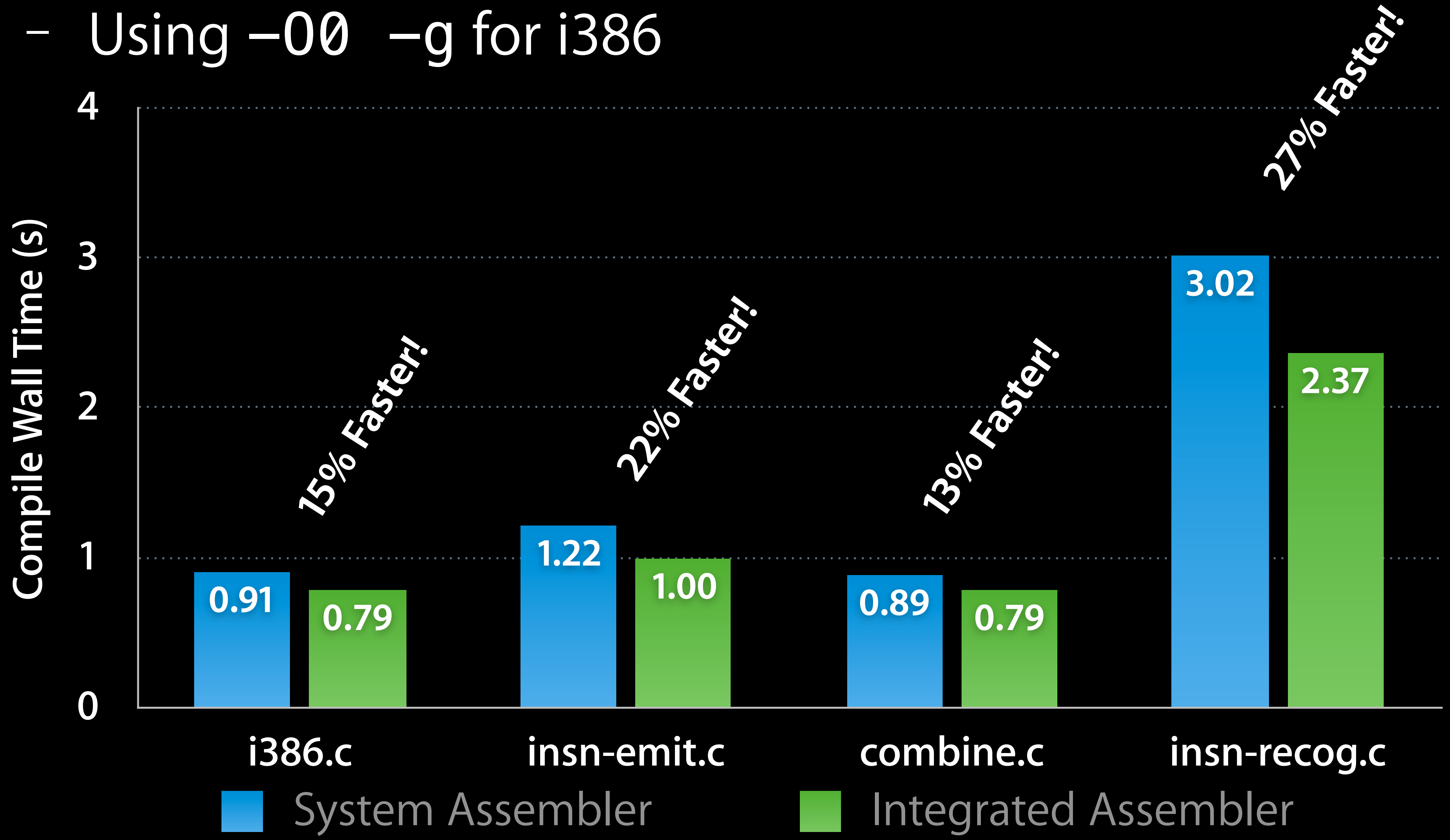
Current Status: Performance

- 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 `-O0 -g` for i386



Summary

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?