LLVM Intro

Syoyo Fujita
syoyo@lucillerender.org

Agenda

Intro & history

LLVM overview

Demo

Pros & Cons

LLVM Intermediate Language

LLVM tools


LLVM

ら，

Lightweight Language ですかぁ～

No! No! No!


LLVM

ば，

Virtual Machineで

すかぁ～

No! No! No!


LLVM

りょ，

りょうほ～

ですかぁ～!

No!

No!

No!

LLVM

=

Low Level Virtual Machine

いいか

よく聞けッ!

LLVMとはッ

Low Level Virtual Machine のことッ

2000年に

Chris Lattnerら

がスタートさせ

プログラムの

ゆりかごから墓場

まで最適化をしつづける

ことをゴールとした

世界初の
公開実装
コンパイラインフラ
プロジェクト
であるッ!!!

LLVM

マシン非依存な中間言語を定義して

それを取り巻くコンパイラインフラ(C++ ライブラリ群)を提供している

Agenda

Intro & history

LLVM overview

Demo

Pros & Cons

LLVM Intermediate Language

LLVM tools

Frontend

C/C++

Java

Python

...

LLVM IR

LLVM
仮想命令セット

Backend

x86

Sparc

PPC

...

Frontend

C/C++

Java

Python

...

```
int add_func(
  int a, int b)
{
  return a + b;
}
```

LLVM IR

```
define i32 @add_func(i32 %a, i32 %b) {
entry:
     %tmp3 = add i32 %b, %a
     ret i32 %tmp3
}
```

Backend

x86

Sparc

PPC

...

```
_add_func:
  movl  8(%esp), %eax
  addl  4(%esp), %eax
    ret
```

Frontend

LLVM IR

Backend

LLVM IR への
コンパイラ、
変換ツール、API

C/C+

clang

llvm-gcc

Java

pypy

Python

LLVM
仮想命令セット

LLVM C++API

...

x86

Sparc

PPC

...

最適化、
プログラム変換

C/C++

Java

Python

LLVM IR

定数伝播

DCE

Alias 解析

User pass

LLVM
仮想命令セット

Backend

x86

Sparc

PPC

...

シリアライズ,
デシリアライズ

Bitcode writer

Bitcode reader

file

file

Frontend

C/C++

Java

Python

…

LLVM IR

Codegen,
JIT facility

LLVM
仮想命令セット

Backend

Native
CodeGen

Register
Allocation

Instruction
Scheduling

x86

Sparc

PPC

…

History

2000	Chris Latter らがコンパイラ研究のために LLVM プロジェクトを開始する
2005	ver 1.0 リリース
2007	Apple に Chris が hired され、LLVM 開発を続ける Leopard の OpenGL スタックに LLVM が利用される
20XX	iPhone のコンパイラに使われる LLVM 帝国建国?

いまここ

Agenda

Intro & history

LLVM overview

Demo

Pros & Cons

LLVM Intermediate Language

LLVM tools

Agenda

Intro & history

LLVM overview

Demo

Pros & Cons

LLVM Intermediate Language

LLVM tools

なぜ LLVM か 1/2

実務的

llvm-gcc により full C のサポート

gcc ツールチェインとの親和性

ベクトル命令(SIMD) のサポート

実践的

数多くの利用実績がある.

OpenGL on Leopard, iPhone, PhysX?, etc...

なぜ LLVM か 2/2

活発な開発

多くの開発者が参加している

(ただ、バックエンドは x86 以外あまり元気がない)

ライセンスが Illinois OSL(BSD license 的)

改変、自作コンパイラへの組み込みに向く

LLVM が向くもの 1/3

静的言語のバックエンド

最適化つきのバックエンドとして使える

LLVM が向くもの 2/3

パフォーマンス重視のアプリ

数値計算、グラフィックスなど

入力データ, プロセッサに応じてプログラムを JIT

SIMD コード出力のサポート

プロセッサパワーを最大限引き出せる

LLVM が向くもの 3/3

コンパイラ研究のツールとして

モジュラー構成なので、拡張しやすい

LLVM が向かないもの 1/2

動的言語のバックエンド(VM runtime, JIT)

実行してみないと型が分からない

そもそも VM runtime の設計が異なる.

LLVM は静的言語向け. JIT は実質 AOT

GC は optional(pypy が苦労している)

動的言語の JIT は
それはそれで深い

話題になるのでここでは扱わない

動的言語の JIT の議論についてのポインタだけ

概観

Dynamic Languages Strike Back http://steve-yegge.blogspot.com/2008/05/dynamic-languages-strike-back.html http://www.stanford.edu/class/ee380/Abstracts/080507-dynamiclanguages.pdf

Trace tree コンパイル

HotpathVM: An Effective JIT Compiler for Resource-constrained Devices http://www.usenix.org/events/vee06/full_papers/p144-gal.pdf

Andreas Gal http://andreasgal.com/

Double-dispatch で specialization

Efficient Just-In-Time Execution of Dynamically Typed Languages Via Code Specialization Using Precise Runtime Type Inference http://www.ics.uci.edu/~franz/Site/pubs-pdf/ICS-TR-07-10.pdf

VM

Parrotcode: Parrot Virtual Machine http://www.parrotcode.org/

LLVM が向かないもの 2/2

組み込み系(web, mobile, etc...)

ライブラリがでかすぎ(C++ + STL だから).

メモリが多く必要(C++ + STL だから)

リソースマネジメント機構が LLVM 中間言語で規定されてない(LowLevel だから).

Debugビルド：
120 MB!!!

Shared Folder

Name	Size	Date Modified	Kind
clang	11.6 MB	Today, 2:46 PM	Unix …ble File
llvmc2	624 KB	Today, 2:10 PM	Unix …ble File
llvm-stub	16 KB	Today, 2:… PM	Unix …ble File
llvm-bcanaly…	8	…day, 2:… PM	Unix …ble File
bugpoint	…	…day, 2:… PM	Unix …ble File
llvm-db	4.2 MB	…day, 2:09 PM	Unix …ble File
llvm-extract	4.3 MB	Today, 2:09 PM	Unix …ble File
lli	15 MB	Today, 2:09 PM	Unix …ble File
gccld	4 KB	Today, 2:09 PM	Unix …ble File
gccas	…KB	Today, 2:09 PM	Unix …ble File
llvm-link	4… MB	Today, 2:09 PM	Unix …ble File
llvm-prof	…4 MB	Today, 2:09 PM	Unix …ble File
llvm-ld	8.9 MB	Today, 2:09 PM	Unix …ble File
llvm-nm	3.9 MB	Today, 2:09 PM	Unix …ble File
llvm-ar	3.9 MB	Today, 2:09 PM	Unix …ble File
llvm-ranlib	3.8 MB	Today, 2:09 PM	
llc	19.8 MB	Today, 2:09 PM	
llvm-dis	3.8 MB	Today, 2:09 PM	
llvm-as	4.4 MB	Today, 2:09 PM	
opt	11.8 MB	Today, 2:09 PM	
llvm-config	20 KB	Today, 2:08 PM	
llvm-PerfectShuffle	44 KB	Today, 1:59 PM	
fpcmp	348 KB	Today, 1:59 PM	
tblgen	3.2 MB	Today, 1:59 PM	Unix …ble File

24 items 121 MB

▼General:

Kind: 24 documents

Size: 121 MB on disk (126,869,389 bytes)

Where: /Users/syoyo/work/llvm-trunk/Debug/bin

LLVM の漢気

ライセンスが Illinois OSL(BSD に似たもの)

改変が自由

依存ライブラリは STL のみ

必要なのがあればすべて自作している(APFloat: 浮動小数点ライブラリ, 等)

C++

gcc よりはきれいなコード

LLVM のここがよくない

IR 仕様や実装がころころ変わる

オレ様仕様. 最近は落ち着いてきた

C++

よく assert で落ちる

bitcode にバージョン間で互換性がない

Agenda

Intro & history

LLVM overview

Demo

Pros & Cons

LLVM Intermediate Language

LLVM tools

LLVM 中間言語 1/2

LLVM IR と呼んでいる

Java バイトコードみたいなもの

レジスタマシン

SSA 形式

再代入不可 => 読めなくはないが、人が手書きで書くものではない.

LLVM 中間言語 2/2

LLVM アセンブリ

テキスト形式の LLVM IR

LLVM ビットコード

ファイルなどにシリアライズされた LLVM IR(アセンブリ)

「バイトコードよりも低水準だぞ」という意思の表れか？

LLVM IR API

IR インストラクションは C++ のクラスに／へ一意に対応している

C++

```
// create fib(x-1)
Value *Sub = BinaryOperator::CreateSub(ArgX, One, "arg", RecurseBB);
CallInst *CallFibX1 = CallInst::Create(FibF, Sub, "fibx1", RecurseBB);
CallFibX1->setTailCall();
```

LLVM IR

```
%tmp2 = sub i32 %tmp1, 1        ; <i32> [#uses=1]
%tmp3 = call i32 (...)* bitcast (i32 (i32)* @fib to i32 (...)*)( i32
%tmp2 ) nounwind      ; <i32> [#uses=1]
```

```
float
add_func(float a, float b)
{
    return a + b;
}
```

がどう LLVM アセンブリに変換されるか見ていく

C

```
float
add_func(float a, float b)
{
    return a + b;
}
```

LLVM

```
define float @add_func(float %a, float %b) {
entry:
    %a_addr = alloca float     ; <float*> [#us
    %b_addr = alloca float     ; <float*> [#us
    %retval = alloca float     ; <float*> [#us
    %tmp = alloca float    ; <float*> [#uses=2
    %"alloca point" = bitcast i32 0 to i32
    store float %a, float* %a_addr
    store float %b, float* %b_addr
    %tmp1 = load float* %a_addr, align 4
    %tmp2 = load float* %b_addr, align 4
    %tmp3 = add float %tmp1, %tmp2     ; <floa
    store float %tmp3, float* %tmp, align 4
    %tmp4 = load float* %tmp, align 4      ; <
    store float %tmp4, float* %retval, align 4
    br label %return

return:     ; preds = %entry
    %retval5 = load float* %retval     ; <floa
    ret float %retval5
}
```

@ 付き: グローバル識別子(関数、大域変数)

% 付き: ローカル識別子(レジスタ名、型名)

```
define float @add_func(float %a, float %b) {
entry:
    %a_addr = alloca float      ; <float*> [#uses=2]
    %b_addr = alloca float      ; <float*> [#uses=2]
    %retval = alloca float      ; <float*> [#uses=2]
    %tmp = alloca float     ; <float*> [#uses=2]
    %"alloca point" = bitcast i32 0 to i32      ; <i32> [#uses=0]
    store float %a, float* %a_addr
    store float %b, float* %b_addr
    %tmp1 = load float* %a_addr, align 4        ; <float> [#uses=1]
    %tmp2 = load float* %b_addr, align 4        ; <float> [#uses=1]
    %tmp3 = add float %tmp1, %tmp2      ; <float> [#uses=1]
    store float %tmp3, float* %tmp, align 4
    %tmp4 = load float* %tmp, align 4       ; <float> [#uses=1]
    store float %tmp4, float* %retval, align 4
    br label %return

return:     ; preds = %entry
    %retval5 = load float* %retval      ; <float> [#uses=1]
    ret float %retval5
}
```

LLVM アセンブリはすべて型付き

```
define float @add_func(float %a, float %b) {
entry:
    %a_addr = alloca float      ; <float*> [#uses=2]
    %b_addr = alloca float      ; <float*> [#uses=2]
    %retval = alloca float      ; <float*> [#uses=2]
    %tmp = alloca float     ; <float*> [#uses=2]
    %"alloca point" = bitcast i32 0 to i32      ; <i32> [#uses=0]
    store float %a, float* %a_addr
    store float %b, float* %b_addr
    %tmp1 = load float* %a_addr, align 4        ; <float> [#uses=1]
    %tmp2 = load float* %b_addr, align 4        ; <float> [#uses=1]
    %tmp3 = add float %tmp1, %tmp2      ; <float> [#uses=1]
    store float %tmp3, float* %tmp, align 4
    %tmp4 = load float* %tmp, align 4       ; <float> [#uses=1]
    store float %tmp4, float* %retval, align 4
    br label %return

return:     ; preds = %entry
    %retval5 = load float* %retval      ; <float> [#uses=1]
    ret float %retval5
}
```

stack

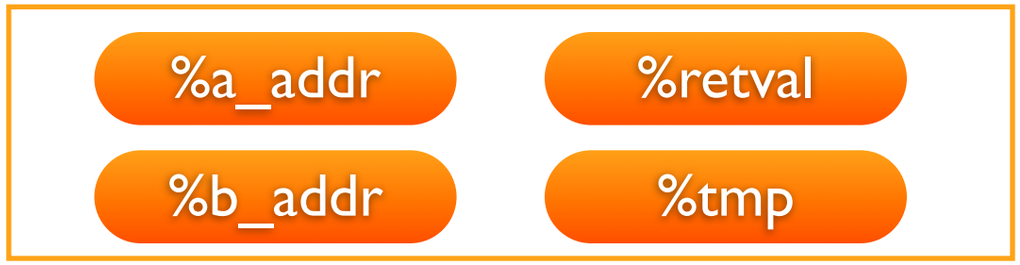
reg

%a

%b

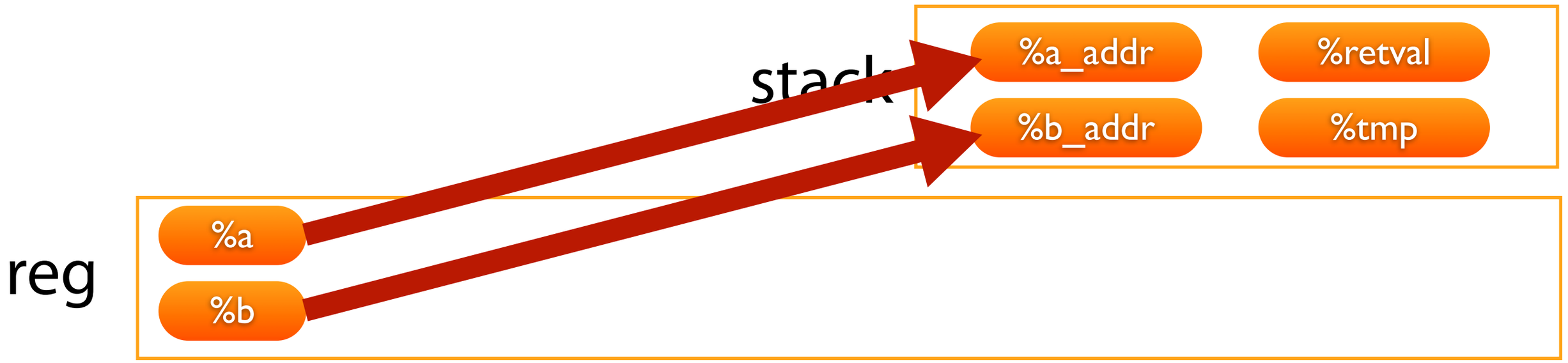
```
define float @add_func(float %a, float %b) {
entry:
    %a_addr = alloca float      ; <float*> [#uses=2]
    %b_addr = alloca float      ; <float*> [#uses=2]
    %retval = alloca float      ; <float*> [#uses=2]
    %tmp = alloca float     ; <float*> [#uses=2]
    %"alloca point" = bitcast i32 0 to i32      ; <i32> [#uses=0]
    store float %a, float* %a_addr
    store float %b, float* %b_addr
    %tmp1 = load float* %a_addr, align 4        ; <float> [#uses=1]
    %tmp2 = load float* %b_addr, align 4        ; <float> [#uses=1]
    %tmp3 = add float %tmp1, %tmp2      ; <float> [#uses=1]
    store float %tmp3, float* %tmp, align 4
    %tmp4 = load float* %tmp, align 4       ; <float> [#uses=1]
    store float %tmp4, float* %retval, align 4
    br label %return

return:     ; preds = %entry
    %retval5 = load float* %retval      ; <float> [#uses=1]
    ret float %retval5
}
```

stack

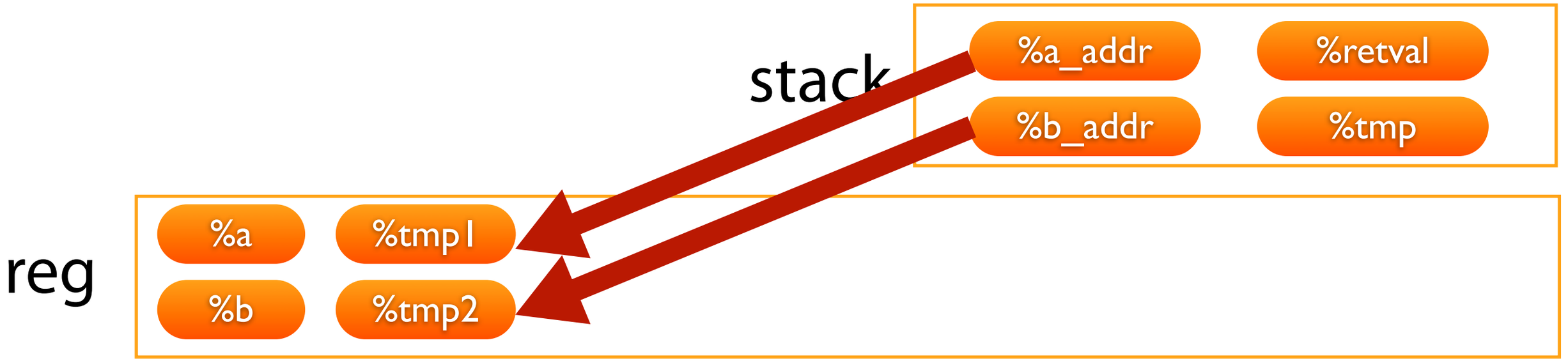


reg


```
define float @add_func(float %a, float %b) {
entry:
    %a_addr = alloca float      ; <float*> [#uses=2]
    %b_addr = alloca float      ; <float*> [#uses=2]
    %retval = alloca float      ; <float*> [#uses=2]
    %tmp = alloca float     ; <float*> [#uses=2]
    %"alloca point" = bitcast i32 0 to i32      ; <i32> [#uses=0]
    store float %a, float* %a_addr
    store float %b, float* %b_addr
    %tmp1 = load float* %a_addr, align 4        ; <float> [#uses=1]
    %tmp2 = load float* %b_addr, align 4        ; <float> [#uses=1]
    %tmp3 = add float %tmp1, %tmp2      ; <float> [#uses=1]
    store float %tmp3, float* %tmp, align 4
    %tmp4 = load float* %tmp, align 4       ; <float> [#uses=1]
    store float %tmp4, float* %retval, align 4
    br label %return

return:     ; preds = %entry
    %retval5 = load float* %retval      ; <float> [#uses=1]
    ret float %retval5
}
```




```
define float @add_func(float %a, float %b) {
entry:
    %a_addr = alloca float      ; <float*> [#uses=2]
    %b_addr = alloca float      ; <float*> [#uses=2]
    %retval = alloca float      ; <float*> [#uses=2]
    %tmp = alloca float     ; <float*> [#uses=2]
    %"alloca point" = bitcast i32 0 to i32      ; <i32> [#uses=0]
    store float %a, float* %a_addr
    store float %b, float* %b_addr
    %tmp1 = load float* %a_addr, align 4        ; <float> [#uses=1]
    %tmp2 = load float* %b_addr, align 4        ; <float> [#uses=1]
    %tmp3 = add float %tmp1, %tmp2      ; <float> [#uses=1]
    store float %tmp3, float* %tmp, align 4
    %tmp4 = load float* %tmp, align 4       ; <float> [#uses=1]
    store float %tmp4, float* %retval, align 4
    br label %return

return:     ; preds = %entry
    %retval5 = load float* %retval      ; <float> [#uses=1]
    ret float %retval5
}
```




```
define float @add_func(float %a, float %b) {
entry:
    %a_addr = alloca float      ; <float*> [#uses=2]
    %b_addr = alloca float      ; <float*> [#uses=2]
    %retval = alloca float      ; <float*> [#uses=2]
    %tmp = alloca float     ; <float*> [#uses=2]
    %"alloca point" = bitcast i32 0 to i32      ; <i32> [#uses=0]
    store float %a, float* %a_addr
    store float %b, float* %b_addr
    %tmp1 = load float* %a_addr, align 4        ; <float> [#uses=1]
    %tmp2 = load float* %b_addr, align 4        ; <float> [#uses=1]
    %tmp3 = add float %tmp1, %tmp2      ; <float> [#uses=1]
    store float %tmp3, float* %tmp, align 4
    %tmp4 = load float* %tmp, align 4       ; <float> [#uses=1]
    store float %tmp4, float* %retval, align 4
    br label %return

return:     ; preds = %entry
    %retval5 = load float* %retval      ; <float> [#uses=1]
    ret float %retval5
}
```

stack

%a_addr %retval

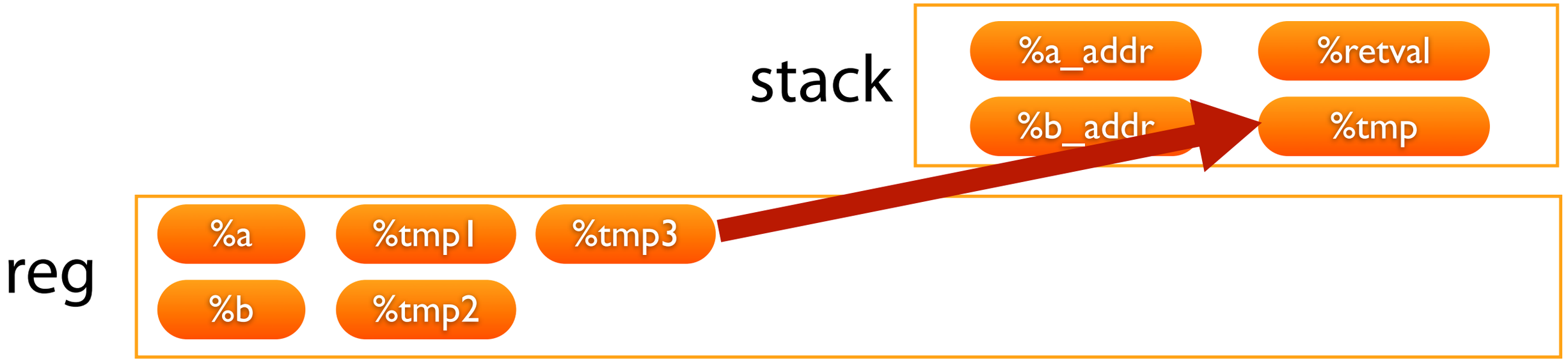
%b_addr %tmp

reg

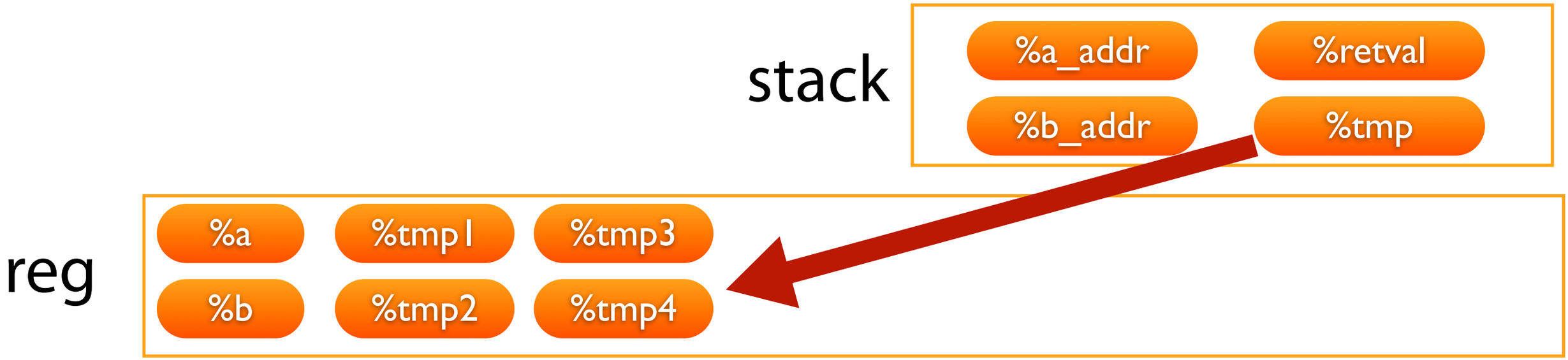
%a %tmp1 %tmp3

%b %tmp2

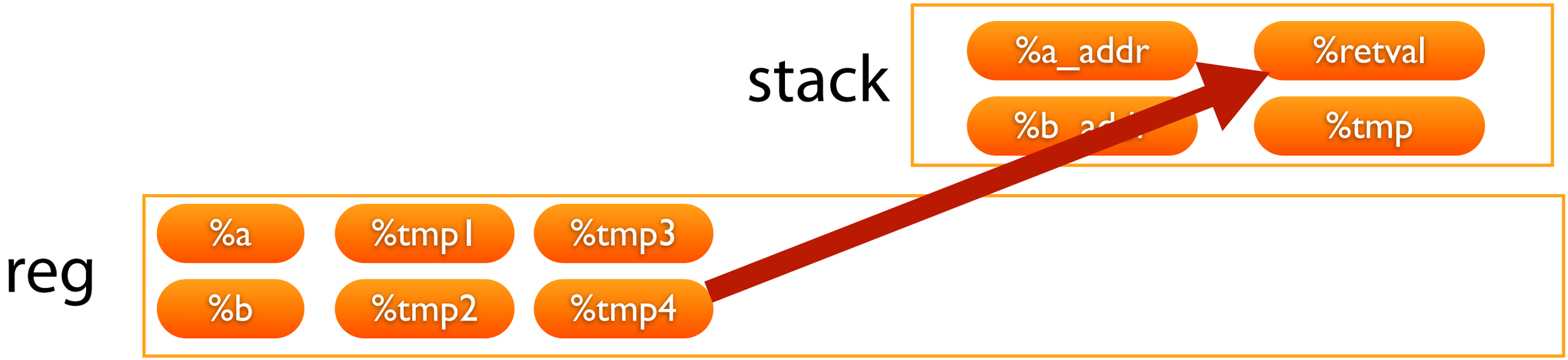
```
define float @add_func(float %a, float %b) {
entry:
    %a_addr = alloca float      ; <float*> [#uses=2]
    %b_addr = alloca float      ; <float*> [#uses=2]
    %retval = alloca float      ; <float*> [#uses=2]
    %tmp = alloca float     ; <float*> [#uses=2]
    %"alloca point" = bitcast i32 0 to i32      ; <i32> [#uses=0]
    store float %a, float* %a_addr
    store float %b, float* %b_addr
    %tmp1 = load float* %a_addr, align 4        ; <float> [#uses=1]
    %tmp2 = load float* %b_addr, align 4        ; <float> [#uses=1]
    %tmp3 = add float %tmp1, %tmp2      ; <float> [#uses=1]
    store float %tmp3, float* %tmp, align 4
    %tmp4 = load float* %tmp, align 4       ; <float> [#uses=1]
    store float %tmp4, float* %retval, align 4
    br label %return

return:     ; preds = %entry
    %retval5 = load float* %retval      ; <float> [#uses=1]
    ret float %retval5
}
```



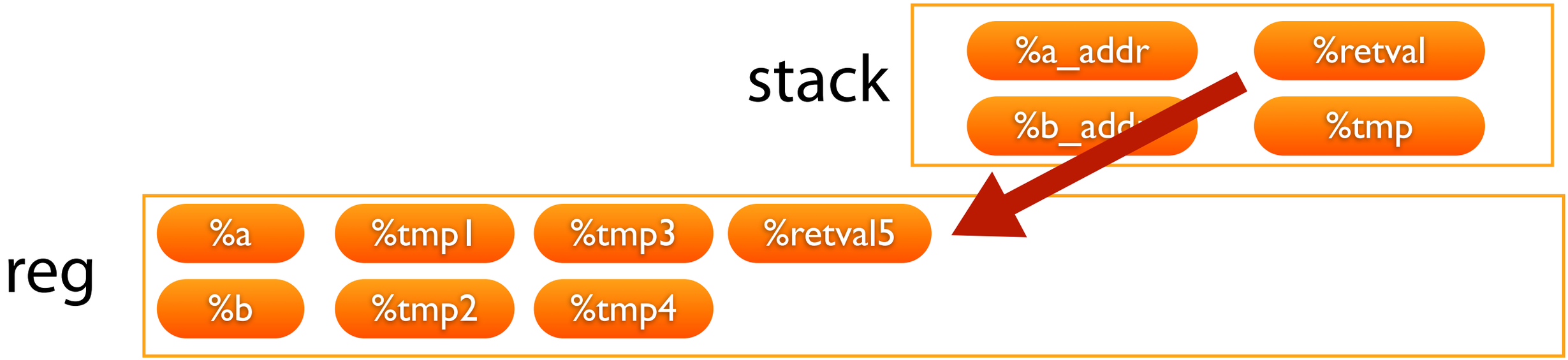

```
define float @add_func(float %a, float %b) {
entry:
    %a_addr = alloca float      ; <float*> [#uses=2]
    %b_addr = alloca float      ; <float*> [#uses=2]
    %retval = alloca float      ; <float*> [#uses=2]
    %tmp = alloca float     ; <float*> [#uses=2]
    %"alloca point" = bitcast i32 0 to i32      ; <i32> [#uses=0]
    store float %a, float* %a_addr
    store float %b, float* %b_addr
    %tmp1 = load float* %a_addr, align 4        ; <float> [#uses=1]
    %tmp2 = load float* %b_addr, align 4        ; <float> [#uses=1]
    %tmp3 = add float %tmp1, %tmp2      ; <float> [#uses=1]
    store float %tmp3, float* %tmp, align 4
    %tmp4 = load float* %tmp, align 4       ; <float> [#uses=1]
    store float %tmp4, float* %retval, align 4
    br label %return

return:     ; preds = %entry
    %retval5 = load float* %retval      ; <float> [#uses=1]
    ret float %retval5
}
```




```
define float @add_func(float %a, float %b) {
entry:
    %a_addr = alloca float      ; <float*> [#uses=2]
    %b_addr = alloca float      ; <float*> [#uses=2]
    %retval = alloca float      ; <float*> [#uses=2]
    %tmp = alloca float     ; <float*> [#uses=2]
    %"alloca point" = bitcast i32 0 to i32      ; <i32> [#uses=0]
    store float %a, float* %a_addr
    store float %b, float* %b_addr
    %tmp1 = load float* %a_addr, align 4        ; <float> [#uses=1]
    %tmp2 = load float* %b_addr, align 4        ; <float> [#uses=1]
    %tmp3 = add float %tmp1, %tmp2      ; <float> [#uses=1]
    store float %tmp3, float* %tmp, align 4
    %tmp4 = load float* %tmp, align 4       ; <float> [#uses=1]
    store float %tmp4, float* %retval, align 4
    br label %return

return:     ; preds = %entry
    %retval5 = load float* %retval      ; <float> [#uses=1]
    ret float %retval5
}
```




```
define float @add_func(float %a, float %b) {
entry:
    %a_addr = alloca float      ; <float*> [#uses=2]
    %b_addr = alloca float      ; <float*> [#uses=2]
    %retval = alloca float      ; <float*> [#uses=2]
    %tmp = alloca float     ; <float*> [#uses=2]
    %"alloca point" = bitcast i32 0 to i32      ; <i32> [#uses=0]
    store float %a, float* %a_addr
    store float %b, float* %b_addr
    %tmp1 = load float* %a_addr, align 4        ; <float> [#uses=1]
    %tmp2 = load float* %b_addr, align 4        ; <float> [#uses=1]
    %tmp3 = add float %tmp1, %tmp2      ; <float> [#uses=1]
    store float %tmp3, float* %tmp, align 4
    %tmp4 = load float* %tmp, align 4       ; <float> [#uses=1]
    store float %tmp4, float* %retval, align 4
    br label %return

return:     ; preds = %entry
    %retval5 = load float* %retval      ; <float> [#uses=1]
    ret float %retval5
}
```




```
define float @add_func(float %a, float %b) {
entry:
    %a_addr = alloca float      ; <float*> [#uses=2]
    %b_addr = alloca float      ; <float*> [#uses=2]
    %retval = alloca float      ; <float*> [#uses=2]
    %tmp = alloca float     ; <float*> [#uses=2]
    %"alloca point" = bitcast i32 0 to i32      ; <i32> [#uses=0]
    store float %a, float* %a_addr
    store float %b, float* %b_addr
    %tmp1 = load float* %a_addr, align 4        ; <float> [#uses=1]
    %tmp2 = load float* %b_addr, align 4        ; <float> [#uses=1]
    %tmp3 = add float %tmp1, %tmp2      ; <float> [#uses=1]
    store float %tmp3, float* %tmp, align 4
    %tmp4 = load float* %tmp, align 4       ; <float> [#uses=1]
    store float %tmp4, float* %retval, align 4
    br label %return

return:     ; preds = %entry
    %retval5 = load float* %retval      ; <float> [#uses=1]
    ret float %retval5
}
```

冗長?...

$ llvm-gcc -emit-llvm -S -O2 muda.c

Or

$ opt -std-compile-opts muda.bc
-f muda.opt.bc

LLVM bc ファイルに対して最適化を施し、

再度 bc ファイルに書き出す

```
define float @add_func(float %a, float %b) nounwind  {
entry:
    %tmp3 = add float %a, %b      ; <float> [#uses=1]
    ret float %tmp3
}
```

Agenda

Intro & history

LLVM overview

Demo

Pros & Cons

LLVM Intermediate Language

LLVM tools

llvm-gcc

gcc

C/C++ Parser

GIMPLE

Backend

a.out

llvm-gcc

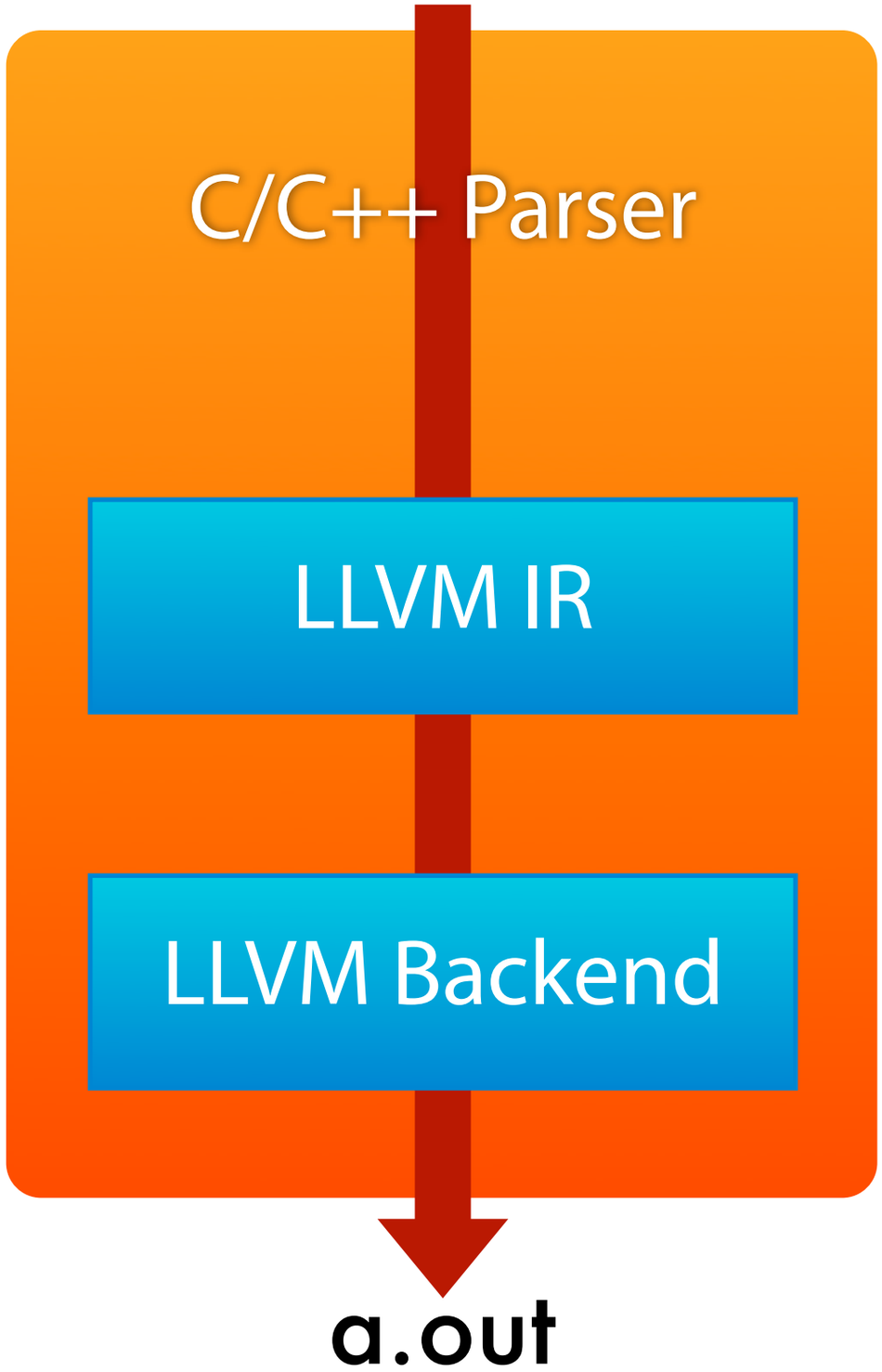
C/C++ Parser

LLVM IR

LLVM Backend

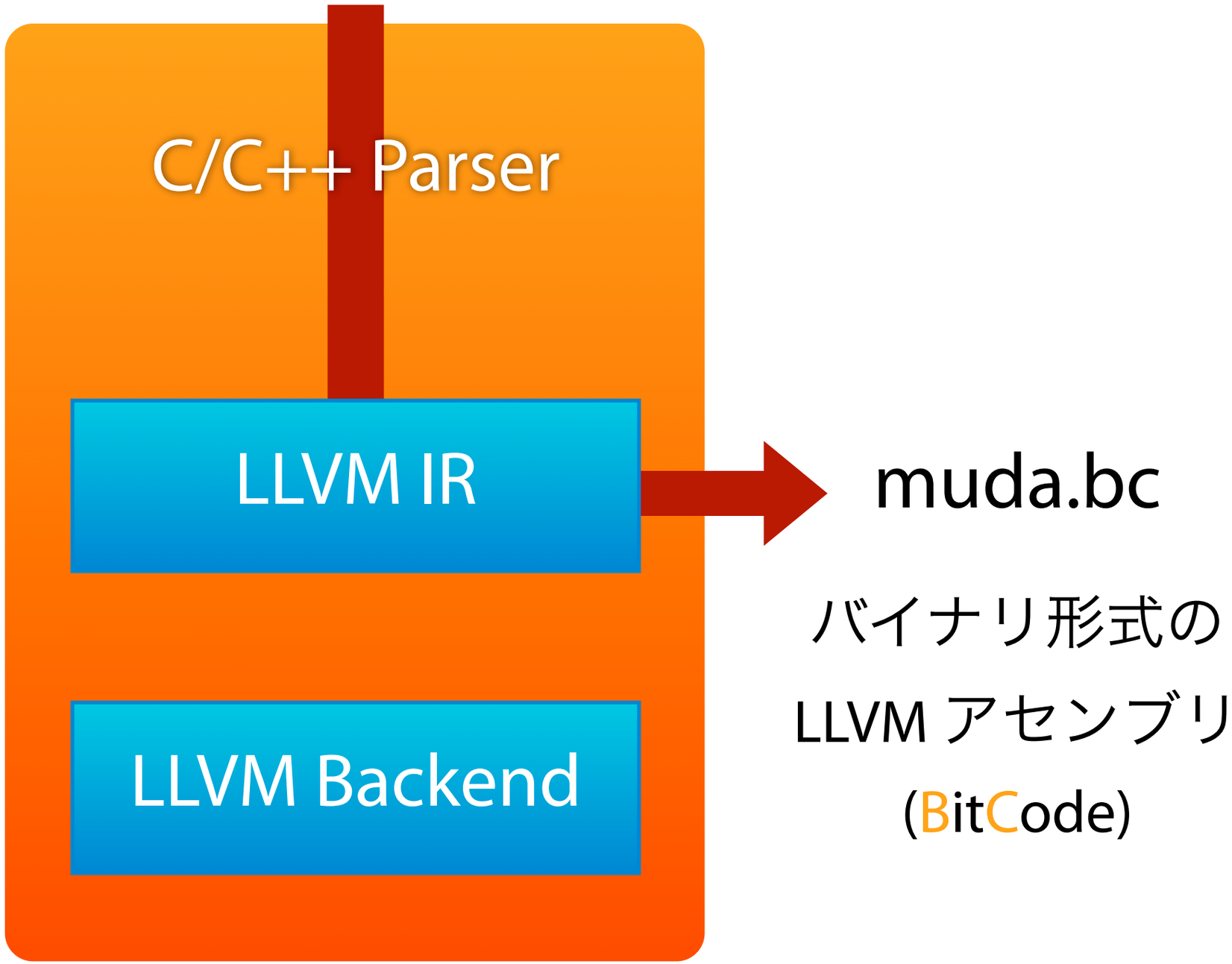
a.out

$ llvm-gcc muda.c

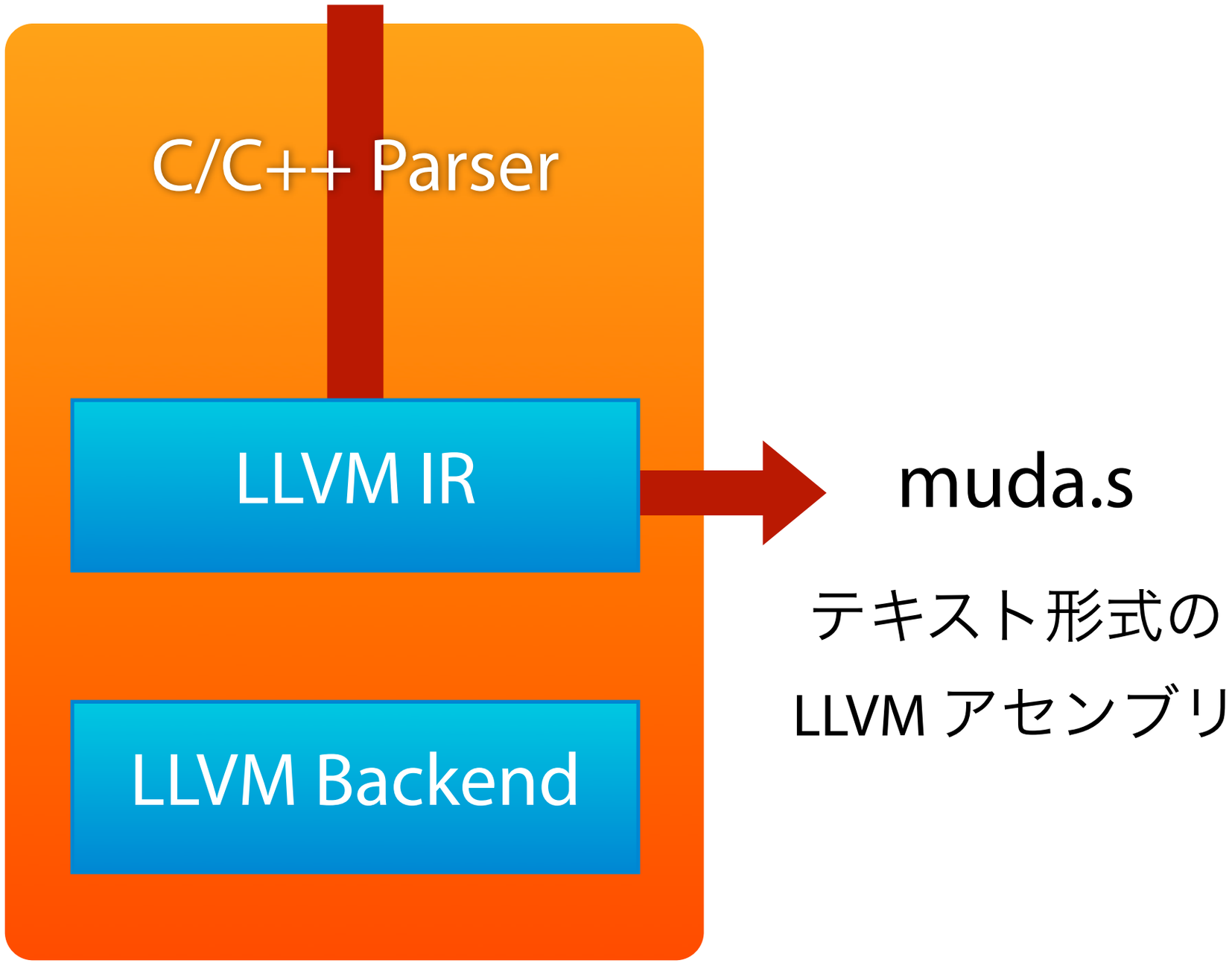


普通のコンパイラとして動く

$ llvm-gcc **-emit-llvm -c** muda.c



$ llvm-gcc **-emit-llvm -S** muda.c



最適化

LLVM IR での最適化

$ **opt** -std-compile-opts <input.bc>

最適化された bc ができる

LLVM バックエンドが行う最適化

$ llc -march=... -mcpu=... -mattr=...

lli も同様

lli

LLVM bc のインタプリタ

デフォルトは JIT コンパイル(AOTコンパイル)してから実行

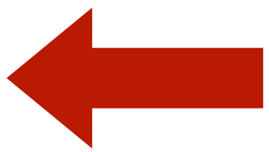
-force-interpreter でインタプリタ実行

fib.c

```
#include <stdio.h>

int
fib(int a)
{
    if (a < 2) return 1;
    return fib(a-2) + fib(a-1);
}

int
main()
{
    printf("fib(30) = %d\n", fib(30));
}
```

```
$ llvm-gcc -emit-llvm -c fib.c
$ time lli fib.o
fib(30) = 1346269

real  0m0.050s
user  0m0.044s
sys   0m0.006s
```

```
$ time lli -force-interpreter fib.o
fib(30) = 1346269

real  0m32.424s
user  0m30.889s
sys   0m0.207s
```



llc

LLVM バックエンドコンパイラ

LLVM bc -> native アセンブラ出力

obj コード出力は experimental

最適化オプション

```
$ llc --march=x86 -mcpu=help
```

-mcpu=

athlon - Select the athlon processor.
athlon-4 - Select the athlon-4 processor.
athlon-fx - Select the athlon-fx processor.
athlon-mp - Select the athlon-mp processor.
athlon-tbird - Select the athlon-tbird processor.
athlon-xp - Select the athlon-xp processor.
athlon64 - Select the athlon64 processor.
c3 - Select the c3 processor.
c3-2 - Select the c3-2 processor.
core2 - Select the core2 processor.
generic - Select the generic processor.
i386 - Select the i386 processor.
i486 - Select the i486 processor.
i686 - Select the i686 processor.
k6 - Select the k6 processor.
k6-2 - Select the k6-2 processor.
k6-3 - Select the k6-3 processor.
k8 - Select the k8 processor.
nocona - Select the nocona processor.
opteron - Select the opteron processor.
penryn - Select the penryn processor.
pentium - Select the pentium processor.
pentium-m - Select the pentium-m processor.
pentium-mmx - Select the pentium-mmx processor.
pentium2 - Select the pentium2 processor.
pentium3 - Select the pentium3 processor.
pentium4 - Select the pentium4 processor.
pentiumpro - Select the pentiumpro processor.
prescott - Select the prescott processor.
winchip-c6 - Select the winchip-c6 processor.
winchip2 - Select the winchip2 processor.
x86-64 - Select the x86-64 processor.
yonah - Select the yonah processor.

-mattr=

3dnow - Enable 3DNow! instructions.
3dnowa - Enable 3DNow! Athlon instructions.
64bit - Support 64-bit instructions.
mmx   - Enable MMX instructions.
sse   - Enable SSE instructions.
sse2   - Enable SSE2 instructions.
sse3   - Enable SSE3 instructions.
sse41 - Enable SSE 4.1 instructions.
sse42 - Enable SSE 4.2 instructions.
ssse3 - Enable SSSE3 instructions.

```
define void @t1(float* %R, <4 x float>* %P1) {
   %X = load <4 x float>* %P1
   %tmp = extractelement <4 x float> %X, i32 3
   store float %tmp, float* %R
   ret void
}
```

```
$ llvm-as < input.ll | llc -march=x86 -mattr=+sse41

...

_t1:
Leh_func_begin1:
Llabel1:
    movl 8(%esp), %eax
    movaps  (%eax), %xmm0
    movl 4(%esp), %eax
    extractps $3, %xmm0, (%eax)
    ret
Leh_func_end1:

...
```

?