## NAME

lli - directly execute programs from LLVM bitcode

## SYNOPSIS

**lli** [*options*] [*filename*] [*program args*]

## DESCRIPTION

**lli** directly executes programs in LLVM bitcode format. It takes a program in LLVM bitcode format and executes it using a just-in-time compiler or an interpreter.

**lli** is *not* an emulator. It will not execute IR of different architectures and it can only interpret (or JIT-compile) for the host architecture.

The JIT compiler takes the same arguments as other tools, like **llc**, but they don't necessarily work for the interpreter.

If *filename* is not specified, then **lli** reads the LLVM bitcode for the program from standard input.

The optional *args* specified on the command line are passed to the program as arguments.

## GENERAL OPTIONS

**-fake-argv0=executable**

Override the **argv[0]** value passed into the executing program.

**-force-interpreter={false,true}**

If set to true, use the interpreter even if a just-in-time compiler is available for this architecture. Defaults to false.

**-help**

Print a summary of command line options.

**-load=pluginfilename**

Causes **lli** to load the plugin (shared object) named *pluginfilename* and use it for optimization.

**-stats**

Print statistics from the code-generation passes. This is only meaningful for the just-in-time compiler, at present.

**-time-passes**

Record the amount of time needed for each code-generation pass and print it to standard error.

**-version**
Print out the version of **lli** and exit without doing anything else.

## TARGET OPTIONS

**-mtriple=target triple**
Override the target triple specified in the input bitcode file with the specified string. This may result in a crash if you pick an architecture which is not compatible with the current system.

**-march=arch**
Specify the architecture for which to generate assembly, overriding the target encoded in the bitcode file. See the output of **llc -help** for a list of valid architectures. By default this is inferred from the target triple or autodetected to the current architecture.

**-mcpu=cpuname**
Specify a specific chip in the current architecture to generate code for. By default this is inferred from the target triple and autodetected to the current architecture. For a list of available CPUs, use: **llvm-as < /dev/null | llc -march=xyz -mcpu=help**

**-mattr=a1,+a2,-a3,...**
Override or control specific attributes of the target, such as whether SIMD operations are enabled or not. The default set of attributes is set by the current CPU. For a list of available attributes, use: **llvm-as < /dev/null | llc -march=xyz -mattr=help**

## FLOATING POINT OPTIONS

**-disable-excess-fp-precision**
Disable optimizations that may increase floating point precision.

**-enable-no-infs-fp-math**
Enable optimizations that assume no Inf values.

**-enable-no-nans-fp-math**
Enable optimizations that assume no NAN values.

**-enable-unsafe-fp-math**
Causes **lli** to enable optimizations that may decrease floating point precision.

**-soft-float**
Causes **lli** to generate software floating point library calls instead of equivalent hardware instructions.

## CODE GENERATION OPTIONS

**-code-model=model**
Choose the code model from:

default: Target default code model
tiny: Tiny code model
small: Small code model
kernel: Kernel code model
medium: Medium code model
large: Large code model

**-disable-post-RA-scheduler**
Disable scheduling after register allocation.

**-disable-spill-fusing**
Disable fusing of spill code into instructions.

**-jit-enable-eh**
Exception handling should be enabled in the just-in-time compiler.

**-join-liveintervals**
Coalesce copies (default=true).

**-nozero-initialized-in-bss**
Don’t place zero-initialized symbols into the BSS section.

**-pre-RA-sched=scheduler**
Instruction schedulers available (before register allocation):

=default: Best scheduler for the target
=none: No scheduling: breadth first sequencing
=simple: Simple two pass scheduling: minimize critical path and maximize processor utilization
=simple-noitin: Simple two pass scheduling: Same as simple except using generic latency
=list-burr: Bottom-up register reduction list scheduling
=list-tdrr: Top-down register reduction list scheduling

=list-td: Top-down list scheduler

**-regalloc=allocator**

Register allocator to use (default=linearscan)

=bigblock: Big-block register allocator
=linearscan: linear scan register allocator
=local: local register allocator
=simple: simple register allocator

**-relocation-model=model**

Choose relocation model from:

=default: Target default relocation model
=static: Non-relocatable code
=pic: Fully relocatable, position independent code
=dynamic-no-pic: Relocatable external references, non-relocatable code

**-spiller**

Spiller to use (default=local)

=simple: simple spiller
=local: local spiller

**-x86-asm-syntax=syntax**

Choose style of code to emit from X86 backend:

=att: Emit AT&T-style assembly
=intel: Emit Intel-style assembly

## EXIT STATUS

If **lli** fails to load the program, it will exit with an exit code of 1. Otherwise, it will return the exit code of the program it executes.

## SEE ALSO

**llc(1)**

## AUTHOR

Maintained by the LLVM Team (https://llvm.org/).

**COPYRIGHT**