中国科学院软件研究所 Institute of Software Chinese Academy of Sciences

智能软件研究中心 Intelligent Software Research Center

软件所智能软件中心PLCT实验室 王鹏 实习生

2020//04/29

目 录

01 clang 和 rvv-benchmark 编写

02 SelectionDAG 介绍

• 01 rvv-benchmark 编写

将rvv spec中的例子封装成测试用例

https://github.com/riscv/riscv-v-spec

```
riscv-v-spec$ ls example/
memcpy.s      sgemm.S       strlen.s      vvaddint32.s
saxpy.s       strcpy.s      strncpy.s
```

- 编写一个有main函数的C文件，调用汇编文件中实现的函数
- 编写Makefile，能够使用rvv-llvm编译C文件和汇编文件，然后使用链接器链接成可执行程序
- 能够在spike/qemu模拟器上成功运行生成的可执行程序

```
u@u-virtual-machine:~/tools/test/c-assemble/riscv-v-spec/example$ clang -o rvv-test main.c
u@u-virtual-machine:~/tools/test/c-assemble/riscv-v-spec/example$ ls
h        Makefile  rvv-test  sgemm_nn.S  strlen.s   vvaddint32.s
main.c  memcpy.s  saxpy.s   strcpy.s    strncpy.s
u@u-virtual-machine:~/tools/test/c-assemble/riscv-v-spec/example$ ./rvv-test
 rvv-benchmark

//strcpy.s --------------------//

 srcstr[] = ABCDEFGH, dststr[] = abcdefgh, after calling strcpy.s
 dststr[] = ABCDEFGH
//strncpy.s--------------------//

 srcnstr[] = ABCDEFGH, dstnstr[] = abcdefgh, n=3,  after calling strcpy.s
 dstnstr[] = ABCdefgh
//vvaddint32.s--------------------//

2, 3, 4, 5, 6,
//memcpy.s and strlen.s---------//

hellowilliam
//saxpy.s--------------------//

15.000000, 15.000000, 15.000000, 15.000000, 15.000000, 15.000000, 15.000000, 0.000000, 7.000000,
 7.000000, 7.000000, 7.000000, 22862829672416478815248711 68.000000, 22579498013620981379775856 64
.000000, 8720196804555290806012870656.000000, 0.000000, -0.000000, 0.000000, 0.000000,
//sgemm,S--------------------//

//--------------------------//
```

```
# rvv-benchmark test
#
#

objects = main.c

rvv-test : $(objects)
	clang -o rvv-test $(objects)
clean :
	rm -f rvv-test $(objects)
~
~
~
~
```

Makefile

```
u@u-virtual-machine:~/tools/test/c-assemble/riscv-v-spec/zz$ vim Makefile
u@u-virtual-machine:~/tools/test/c-assemble/riscv-v-spec/zz$ make
clang -o rvv-test main.c
u@u-virtual-machine:~/tools/test/c-assemble/riscv-v-spec/zz$ ls
main.c  Makefile  memcpy.s  rvv-test  saxpy.s  sgemm.S  strcpy.s  strlen.s  strncpy.s  vvaddint32.s
u@u-virtual-machine:~/tools/test/c-assemble/riscv-v-spec/zz$ ./rvv-test
 rvv-benchmark

//strcpy.s --------------------//

 srcstr[] = ABCDEFGH, dststr[] = abcdefgh, after calling strcpy.s
 dststr[] = ABCDEFGH
//strncpy.s-------------------//

 srcnstr[] = ABCDEFGH, dstnstr[] = abcdefgh, n=3,  after calling strcpy.s
 dstnstr[] = ABCdefgh
//vvaddint32.s-------------------//

2, 3, 4, 5, 6,
//memcpy.s and strlen.s--------//

hellowilliam
//saxpy.s-------------------//

15.000000, 15.000000, 15.000000, 15.000000, 15.000000, 15.000000, 15.000000, 0.000000, 7.000000, 7.000000, 7.00
0000, 7.000000, 22862829672416478815248711168.000000, 22579498013620981379775856640.000000, 872019680455529080601
2870656.000000, 0.000000, 0.000000, 0.000000, 0.000000,
//sgemm,S-------------------//

//------------------------//
```

```
// #include<cstdio.h>

typedef unsigned long size_t;

int printf(const char *restrict format,...);

extern char* strcpy(char *d, const char *s);
extern char* strncpy(char *d, const char *s, size_t n);
extern void vvaddint32(size_t n, const int *x, const int *y, int *z){
        for (size_t i=0; i<n; i++)
        { z[i]=x[i]+y[i];  }
}

extern void* memcpy(void* dest, const void* src, size_t n);
extern size_t strlen(const char* str);

extern void saxpy(size_t n, const float a, const float* x, float* y){
        size_t i;
        for (i=0; i<n; i++)
                y[i] = a*x[i] + y[i];
}


void sgemm_nn(size_t i, size_t j, size_t k, const float* aa, size_t lda, const float* bb, size_t ldb, float* cc, size_t ldc);
```

```
int main(void){

        printf("\033[1m\033[45;33m rvv-benchmark \033[0m\n\n");
//-------------------------//

        const char srcstr[] = "ABCDEFGH";
              char dststr[] = "abcdefgh";

        printf("//strcpy.s ---------------------// \n\n");
        strcpy(dststr, srcstr);
        printf(" srcstr[] = ABCDEFGH, dststr[] = abcdefgh, after calling strcpy.s     \n ");
        printf("dststr[] = ");
        printf("%s \n", dststr);

//--------------------------//

        printf("//strncpy.s-------------------// \n\n");
        const char srcnstr[] = "ABCDEFGH";
              char dstnstr[] = "abcdefgh";
        strncpy(dstnstr, srcnstr, 3);
        printf(" srcnstr[] = ABCDEFGH, dstnstr[] = abcdefgh, n=3,  after calling strcpy.s     \n ");
        printf("dstnstr[] = ");
        printf("%s \n", dstnstr);
```

```
//-------------------------//
        printf("//vvaddint32.s--------------------// \n\n");
        size_t n=5;
        const int x[5]= { 1, 2, 3, 4, 5};
        const int y[5]= { 1, 1, 1, 1, 1};
              int z[5];
        vvaddint32(n, x, y, z);
        for(int i=0; i<n; i++)
        printf("%d, ", z[i]);
        printf("\n");

//-------------------------//
        printf("//memcpy.s and strlen.s---------// \n\n");
        char* src="hellowilliam";
        char dest[20];
//      size_t m=5;
         size_t m= strlen(src)+1;
        memcpy(dest, src, m);
        printf("%s \n", dest);

//-------------------------//
        printf("//saxpy.s--------------------// \n\n");
        const float xx[7] = {7, 7, 7, 7, 7, 7, 7};
              float yy[7] = {1, 1, 1, 1, 1, 1, 1};
        const float a=2;
        size_t p=19;
        saxpy(p, a, xx, yy);
        for(int i=0; i<p; i++)
        printf("%f, ", yy[i]);
        printf("\n");
```

```
//-----------------------------//
        printf("//sgemm,S---------------------// \n \n");
  /*      size_t i=1;
        size_t j=1;
        size_t k=1;
        const float aa[1] = {1};
        size_t lda;
        const float bb[1] = {2};
        size_t ldb;
              float cc[1] = {3};
        size_t ldc;

        sgemm_nn(i, j, k, aa, lda, bb, ldb, cc, ldc);

        for(int i=0; i<4; i++)
        printf("%f ", cc[i]);
        printf("\n");
*/


//-----------------------------//
//
        printf("//---------------------------// \n");

        return 0;
}
```

```
# RV64IDV system
#
# void
# sgemm_nn(size_t n,
#          size_t m,
#          size_t k,
#          const float*a,   // m * k matrix
#          size_t lda,
#          const float*b,   // k * n matrix
#          size_t ldb,
#          float*c,         // m * n matrix
#          size_t ldc)
#
#  c += a*b (alpha=1, no transpose on input matrices)
#  matrices stored in C row-major order
```

```
u@u-virtual-machine:~/tools/test/c-assemble/riscv-v-spec/zz$ clang -o h main.c
/tmp/main-2fd8d0.o: In function `main':
main.c:(.text+0x5ae): undefined reference to `sgemm_nn'
clang-10: error: linker command failed with exit code 1 (use -v to see invocation)
```

```
u@u-virtual-machine:~/tools/test/c-assemble/riscv-v-spec/zz$ clang-10 --target=riscv32 -o try-test main.c
main.c:8:14: warning: incompatible redeclaration of library function 'strncpy' [-Wincompatible-library-redeclaration]
extern char* strncpy(char *d, const char *s, size_t n);
             ^
main.c:8:14: note: 'strncpy' is a builtin with type 'char *(char *, const char *, unsigned int)'
main.c:14:14: warning: incompatible redeclaration of library function 'memcpy' [-Wincompatible-library-redeclaration]
extern void* memcpy(void* dest, const void* src, size_t n);
             ^
main.c:14:14: note: 'memcpy' is a builtin with type 'void *(void *, const void *, unsigned int)'
main.c:15:15: warning: incompatible redeclaration of library function 'strlen' [-Wincompatible-library-redeclaration]
extern size_t strlen(const char* str);
              ^
main.c:15:15: note: 'strlen' is a builtin with type 'unsigned int (const char *)'
3 warnings generated.
/usr/bin/ld: cannot find crt0.o: No such file or directory
/usr/bin/ld: cannot find crtbegin.o: No such file or directory
/usr/bin/ld: /tmp/main-6ef8dd.o: Relocations in generic ELF (EM: 243)
/tmp/main-6ef8dd.o: error adding symbols: File in wrong format
clang-10: error: ld command failed with exit code 1 (use -v to see invocation)
```

```
  --rbb-port=<port>        Listen on <port> for remote bitbang connection
  --dump-dts               Print device tree string and exit
  --disable-dtb            Don't write the device tree blob into memory
  --progsize=<words>       Progsize for the debug module [default 2]
  --debug-sba=<bits>       Debug bus master supports up to <bits> wide accesses [de
fault 0]
  --debug-auth             Debug module requires debugger to authenticate
  --dmi-rti=<n>            Number of Run-Test/Idle cycles required for a DMI access
 [default 0]
  --abstract-rti=<n>       Number of Run-Test/Idle cycles required for an abstract
command to execute [default 0]
u@u-virtual-machine:~/tools/rvv-benchmark/strcpy$ spike pk rvv-test
bbl loader
z  0000000000000000 ra 0000000000000000 sp 000000007f7e9b50 gp 0000000000000000
tp 0000000000000000 t0 0000000000000000 t1 0000000000000000 t2 0000000000000000
s0 0000000000000000 s1 0000000000000000 a0 0000000000000000 a1 0000000000000000
a2 0000000000000000 a3 0000000000000000 a4 0000000000000000 a5 0000000000000000
a6 0000000000000000 a7 0000000000000000 s2 0000000000000000 s3 0000000000000000
s4 0000000000000000 s5 0000000000000000 s6 0000000000000000 s7 0000000000000000
s8 0000000000000000 s9 0000000000000000 sA 0000000000000000 sB 0000000000000000
t3 0000000000000000 t4 fffffffffffffff4 t5 0000000000000000 t6 0000000000000000
pc 0000000000400436 va 0000000000008948 insn       00008948 sr 8000000200046020
An illegal instruction was executed!
u@u-virtual-machine:~/tools/rvv-benchmark/strcpy$
```

```
u@u-virtual-machine:~/tools/test/rvv-benchmark/strcpy-test$ clang  --target=riscv64 -c strcpy.s -o strcpy.o
 strcpy.s:9:5: error: instruction use requires an option to be enabled
    vsetvli x0, t0, e8,m8   # Max length vectors of bytes
    ^
strcpy.s:12:5: error: instruction use requires an option to be enabled
    vmseq.vi v0, v1, 0      # Flag zero bytes
    ^
strcpy.s:13:5: error: instruction use requires an option to be enabled
    vfirst.m a3, v0         # Zero found?
    ^
strcpy.s:15:5: error: instruction use requires an option to be enabled
    vmsif.m v0, v0          # Set mask up to and including zero byte.
    ^
```

EPI的clang也不支持将rvv-benchmark的那几个汇编测试用例

clang --target=riscv64 -c strcpy.s -o strcpy.o

clang --target=riscv32 -c strcpy.s -o strcpy.o 我们用这个选项

```
//===----------------------------------------------------------------------===//
// Instructions
//===----------------------------------------------------------------------===//

let Predicates = [HasStdExtV] in {

let hasSideEffects = 0, mayLoad = 0, mayStore = 0 in {
def VSETVLI : RVInstSetVLi<(outs GPR:$rd), (ins GPR:$rs1, VTypeIOp:$vtypei),
                           "vsetvli", "$rd, $rs1, $vtypei">;
def VSETVL : RVInstSetVL<(outs GPR:$rd), (ins GPR:$rs1, GPR:$rs2),
                         "vsetvl", "$rd, $rs1, $rs2">;
}
```

EPI已经定义了vsetvli指令，但是在strcpy.s汇编文件中，用clang编译的时候，编译报错。EPI对vsetvli指令的定义只有RISCVInstrInfoV.t这一处

```
u@u-virtual-machine:~/tools/test/rvv-benchmark/strcpy-test$ clang  --target=riscv64 -mattr=+v  -c strcpy.s -o strcpy.o
clang-10: error: unknown argument: '-mattr=+v'
```

```
u@u-virtual-machine:~/tools/test/rvv-benchmark/strcpy-test$ clang  --target=riscv64 -target-feature=+v  -c strcpy.s -o strcpy.o
clang-10: error: unknown argument: '-target-feature=+v'
```

clang --target=riscv32 -mattr=+v -c strcpy.s -o strcpy.o 的时候，clang就报错了，说没有这个option。

我clang --help，开启find功能，和vector字眼相关的，只有

-mprefer-vector-width=<value>，-flax-vector-conversions=<value>，-fslp-vectorize， -fvectorize，-fzvector，-mhvx-length=<value> 这几个选项

```
u@u-virtual-machine:~/tools/test/rvv-benchmark/strcpy-test$ clang  -march=rv32ifv  -c strcpy.s -o strcpy.o
'rv32ifv' is not a recognized processor for this target (ignoring processor)
'rv32ifv' is not a recognized processor for this target (ignoring processor)
strcpy.s:6:7: error: invalid instruction mnemonic 'mv'
      mv a2, a0              # Copy dst
      ^~
strcpy.s:7:7: error: invalid instruction mnemonic 'li'
      li t0, -1              # Infinite AVL
      ^~
```

调试追踪下这段代码：
clang/lib/Driver/ToolChains/Arch/RISCV.cpp

riscv::getRISCVTargetFeatures

这个函数是clang用来根据march选项获得具体的feature，里面注释里有提到v：

StringRef StdExts = "mafdqlcbjtpvn";

```
u@u-virtual-machine:~/tools/test/rvv-benchmark/strcpy-test$ llvm-mc -triple=riscv64 -mattr=+v -o strcpy.o strcpy.s
u@u-virtual-machine:~/tools/test/rvv-benchmark/strcpy-test$ ls
main.c  main.i  main.s  memcpy.s  saxpy.s  sgemm.S  strcpy.o  strcpy.s  strlen.s  strncpy.s  vvaddint32.s
```

llvm-mc -triple=riscv64 -mattr=+v -o strcpy.o strcpy.s

使用上述命令就可以执行，将strcpy.s编译为strcpy.o

```
u@u-virtual-machine:~/tools/test/rvv-benchmark/strcpy-test$ clang -v --target=riscv64 -c strcpy.s -o strcpy.o
clang version 10.0.0
Target: riscv64
Thread model: posix
InstalledDir: /usr/local/bin
 "/usr/local/bin/clang-10" -cc1as -triple riscv64 -filetype obj -main-file-name strcpy.s -target-feature +relax -fdebug-compilation-dir /home/u/tools/tes
t/rvv-benchmark/strcpy-test -dwarf-debug-producer "clang version 10.0.0 " -dwarf-version=4 -mrelocation-model static -target-abi lp64 -o strcpy.o strcpy.
s
strcpy.s:9:5: error: instruction use requires an option to be enabled
    vsetvli x0, t0, e8,m8    # Max length vectors of bytes
    ^
strcpy.s:12:5: error: instruction use requires an option to be enabled
    vmseq.vi v0, v1, 0       # Flag zero bytes
    ^
strcpy.s:13:5: error: instruction use requires an option to be enabled
    vfirst.m a3, v0          # Zero found?
    ^
strcpy.s:15:5: error: instruction use requires an option to be enabled
    vmsif.m v0, v0           # Set mask up to and including zero byte.
    ^
```

出错的地方是， 把vector 属性传给 cc1as 了

邢老师调试了下clang，知道了clang目前不支持 v 特性选项

```
   312         switch (c) {
   313         default:
   314           // Currently LLVM supports only "mafdc".
-> 315            D.Diag(diag::err_drv_invalid_riscv_ext_arch_name)
   316             << MArch << "unsupported standard user-level
extension"
   317             << std::string(1, c);
```

另一种方法是采用riscv64-unknown-elf-gcc来编译，用 ld -nostdlib 将目标文件链接成可执行文件，spike pk main来执行，用spike -d pk main来查看执行的每一条汇编指令。

• 02 SelectionDAG 介绍

在LLVM 3.8之前，LLVM的指令选择有两个，一个基于DAG覆盖的SelectionDAG，另外一个是FastISel。前者在-O方式启用，后则在-O0的时候启用。从3.8开始，后端增加了一个GlobalISel的指令选择器(基于自动macro expanding，其实和FastISel类似，只不过FastISel是手写指令匹配模式)。由于GlobalISel尚未完全成熟(LLVM 6.0仅完整支持ARM后端 ，所以仅基于SelectionDAG讨论后端流程。

LLVM后端大致的工作流程如下所述

1. LLVM IR -> SelectionDAG[与机器无关的ISD操作符].
2. 中间经过若干个DAGCombiner和DAGTypeLegalizer，DAGLegalizer(这些类的作用是将目标机器不支持的类型和DAG操作转换为目标机器支持的操作，同时删除部分冗余计算).
3. 通过调用instructionSelect函数，变换过的SelectionDAG输入到指令选择器中，将机器无关的操作转换为机器支持的操作.
4. 将第3步生成的SelectionDAG[仅使用目标机器支持的操作]输入到指令调度器中，指令调度器通过满足data, output, anti-, control约束依照SelectionDAG构造一个ScheduleDAG.

5. . 指令发射(instruction emit), 对ScheduleDAG进行拓扑排序，生成一个包含SUnit的Sequence列表，里面保存了最终的指令顺序流。然后遍历Sequence列表，依据每个SUnit中的SDNode的操作码得到TargetInstrInfo，然后调用buildMI生成一条MachineInstr，之后就是按照tii描述的操作数信息往mi中插入不同种类的操作数，比如：registerOperand将会创建一个虚拟寄存器，然后调用mi.addRegister()。然后把建立好的mi插入到对应的机器基本块MBB中。

示例

以如下C语言代码片段作为叙述例子，目标机器选择i386-linux-gnu，基于LLVM2.6叙述(其他更新版本的LLVM原理一样，当然也可以选择更新版本的LLVM)。

前置条件

1. 在编译LLVM的时候，需要使用cmake -DLLVM_ENABLE_ASSERTIONS=ON才会启用view-dags等选项，这些选项下面会被用于查看输出的DAG图。
2. 安装xdot等dot查看工具，Ubuntu可以使用sudo apt-get install xdot命令安装。
3. 如果使用较新版本的LLVM，下面的clang-cc 命令名替换为clang -cc1。

```
u@u-virtual-machine:~/tools/test/selectionDAG$ cat add.ll
; ModuleID = 'add.c'
source_filename = "add.c"
target datalayout = "e-m:e-p:32:32-i64:64-n32-S128"
target triple = "riscv32"

; Function Attrs: noinline nounwind optnone
define i32 @add(i32 %a, i32 %b) #0 {
entry:
  %a.addr = alloca i32, align 4
  %b.addr = alloca i32, align 4
  store i32 %a, i32* %a.addr, align 4
  store i32 %b, i32* %b.addr, align 4
  %0 = load i32, i32* %a.addr, align 4
  %1 = load i32, i32* %b.addr, align 4
  %add = add nsw i32 %0, %1
  ret i32 %add
}
```

```
u@u-virtual-machine:~/tools/test/selectionDAG$ cat add.c
int add(int a, int b)
{
        return (a+b);
}
```

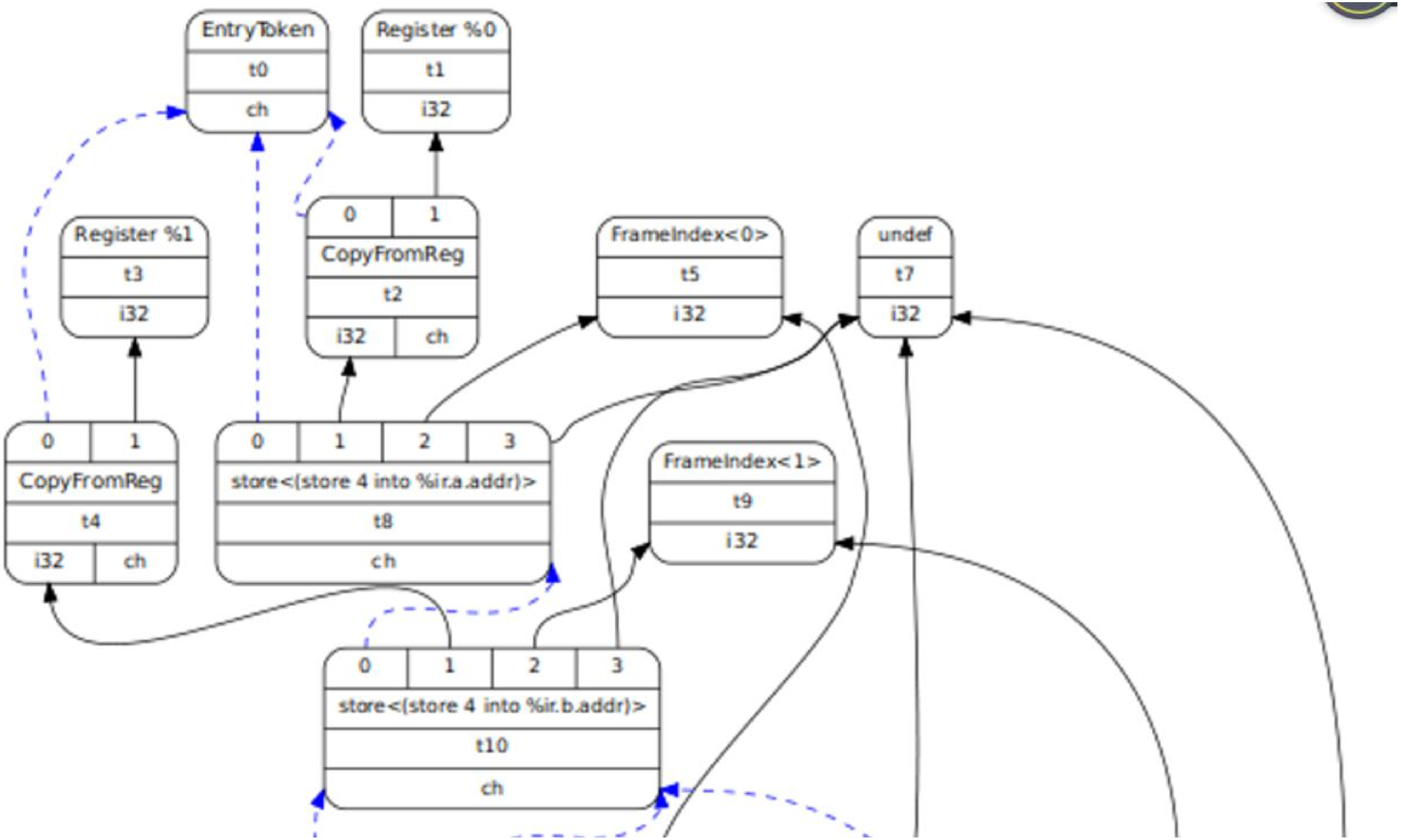
LLVM IR输入

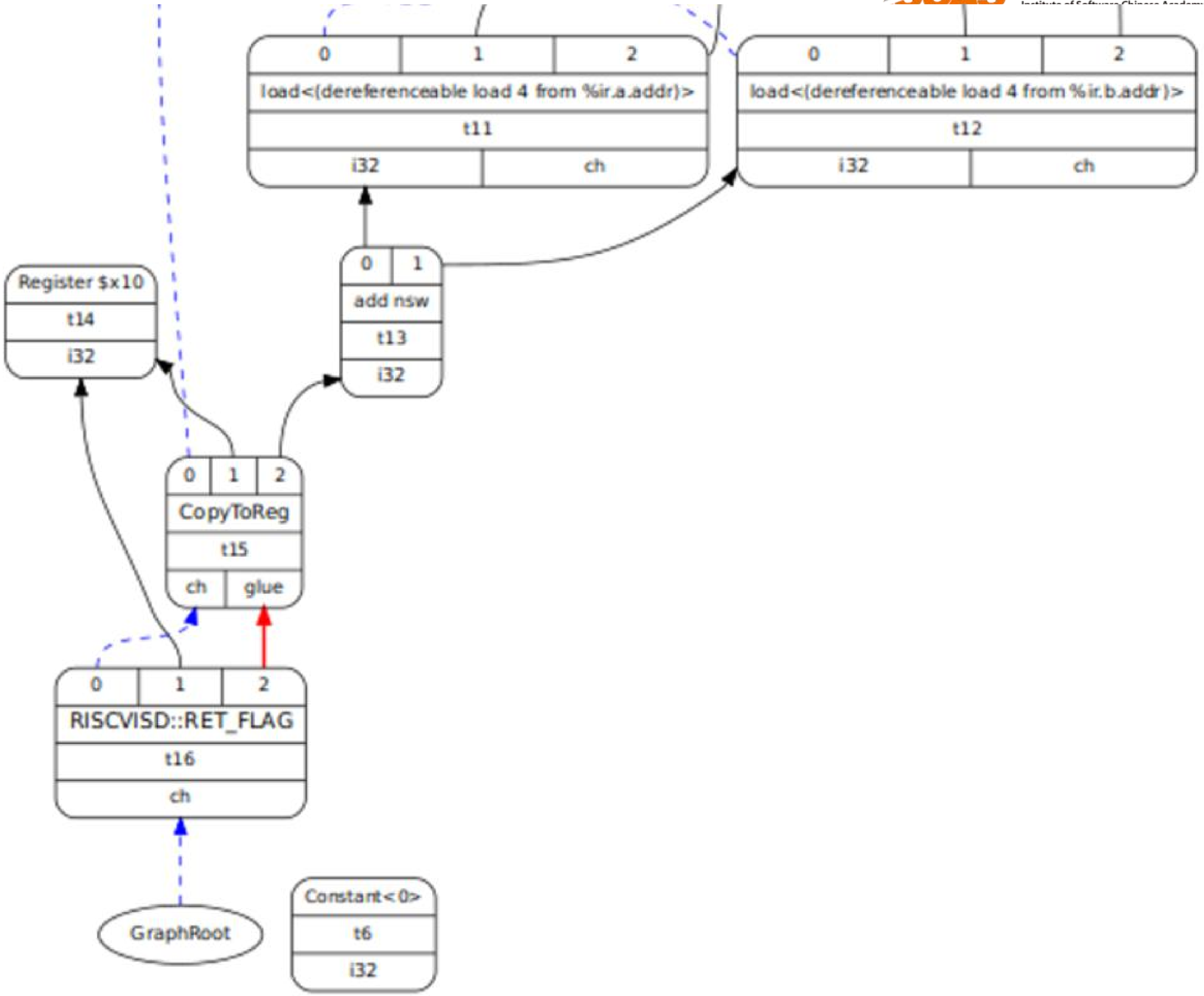
使用命令clang -cc1 -emit-llvm -triple=i386-linux-gnu add.c，生成的LLVM IR如下：

然后调用SelectionDAGBuild类以基本块为单位构造一个SelectionDAG图，所有的alloca会被分配一个函数栈索引FrameIndex，然后构造一个对应的FrameIndexSDNode结点。store和load指令分别转换为ISD::store和ISD::load，add操作转换为ISD::add，ret会转换为X86ISD::RET_FLAG

SelectionDAG变换

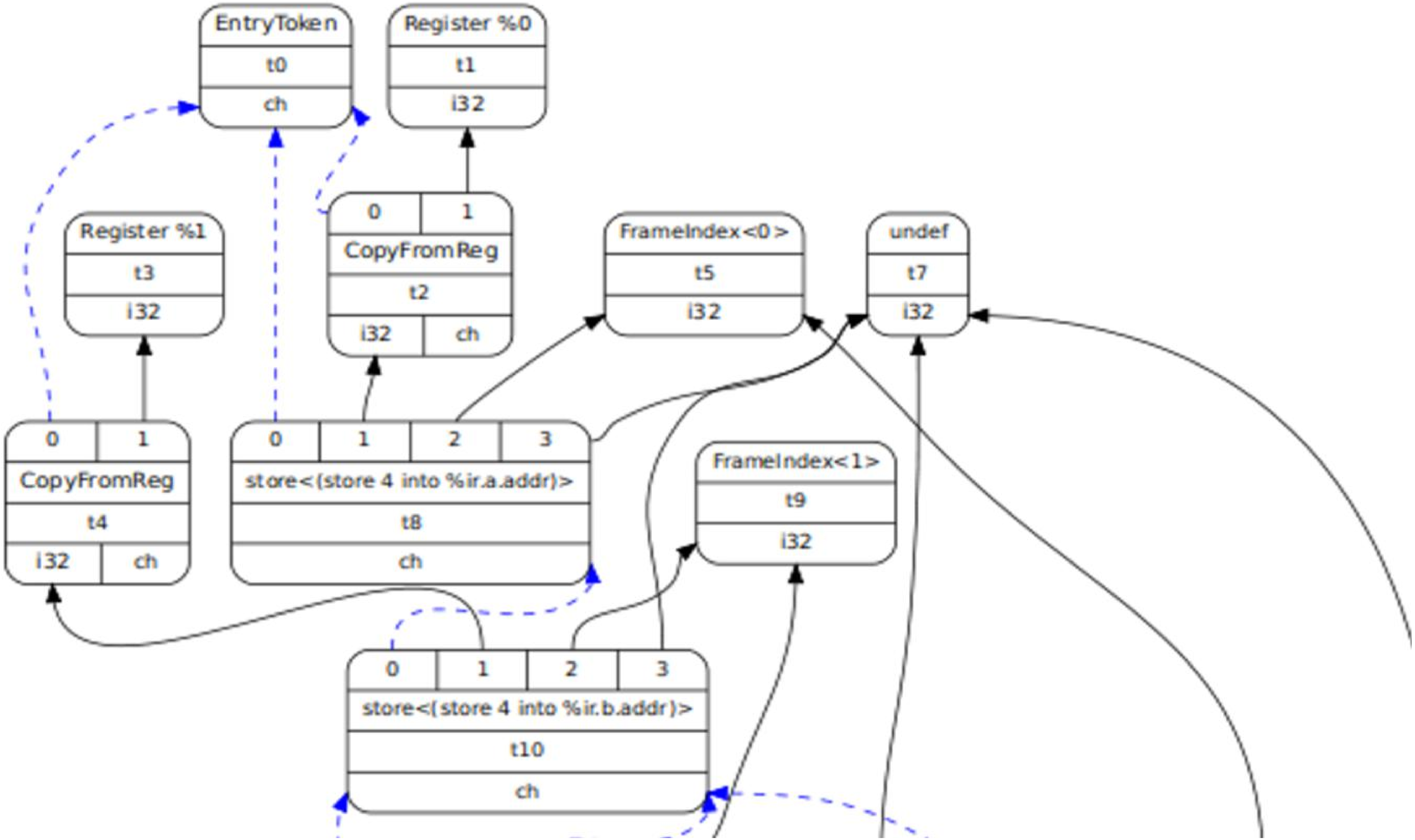
使用clang -cc1 -S -triple=riscv32 add.c -view-dag-combine1-dags 查看输出的DAG如下图所示：

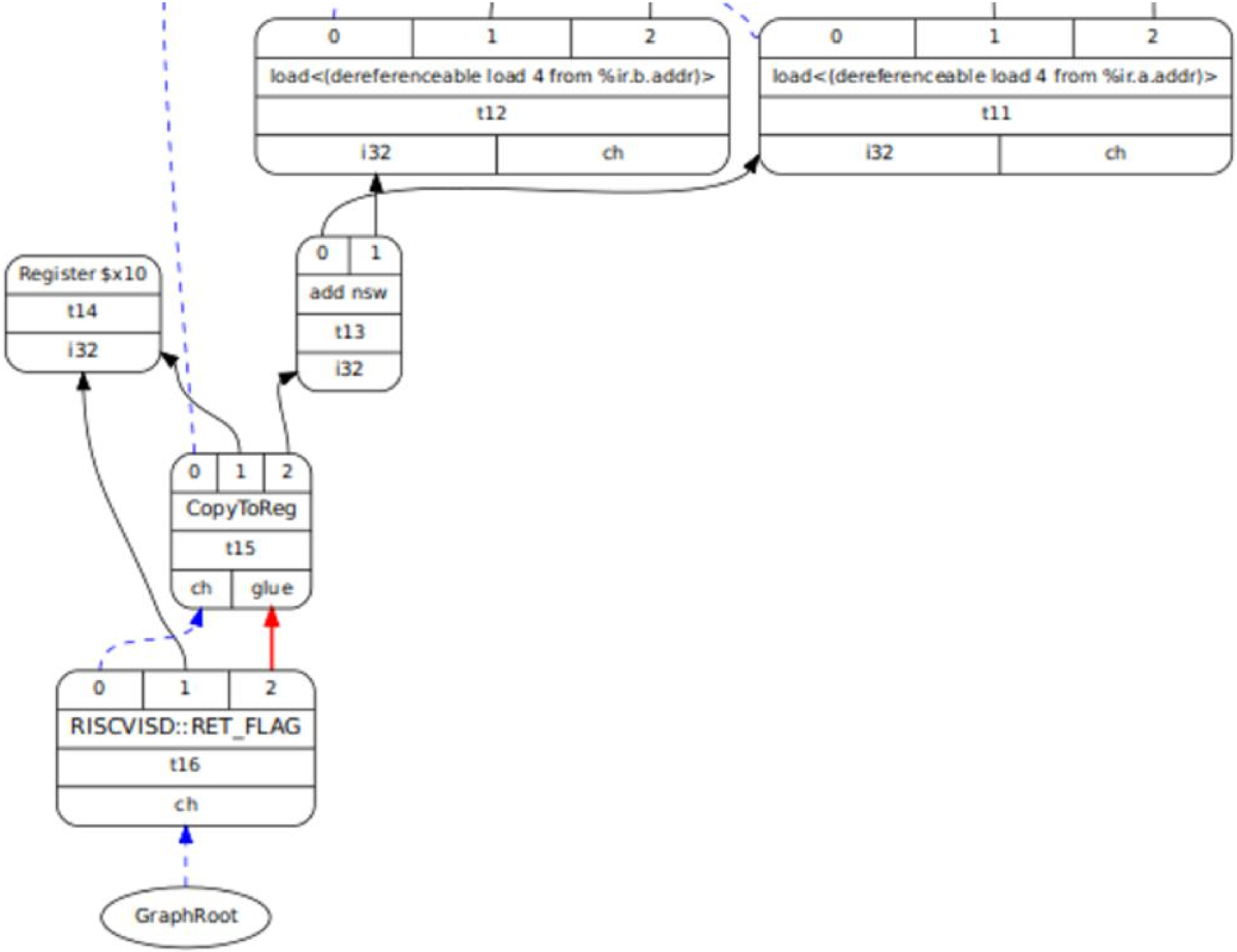

EntryToken
t0
ch
Register %0
t1
i32
Register %1
t3
i32
0
1
CopyFromReg
t2
i32
ch
FrameIndex<0>
t5
i32
undef
t7
i32
0
1
CopyFromReg
t4
i32
ch
0
1
2
3
store<(store 4 into %ir.a.addr)>
t8
ch
FrameIndex<1>
t9
i32
0
1
2
3
store<(store 4 into %ir.b.addr)>
t10
ch



dag-combine1 input for add:entry

经过若干次combine, legalization等变换后，查看命令clang -cc1 -S -triple=riscv32 add.c -view-isel-dags. 生成的DAG图如下：


EntryToken
t0
ch
Register %0
t1
i32
Register %1
t3
i32
0 1
CopyFromReg
t2
i32 ch
FrameIndex<0>
t5
i32
undef
t7
i32
0 1
CopyFromReg
t4
i32 ch
0 1 2 3
store<(store 4 into %ir.a.addr)>
t8
ch
FrameIndex<1>
t9
i32
0 1 2 3
store<(store 4 into %ir.b.addr)>
t10
ch

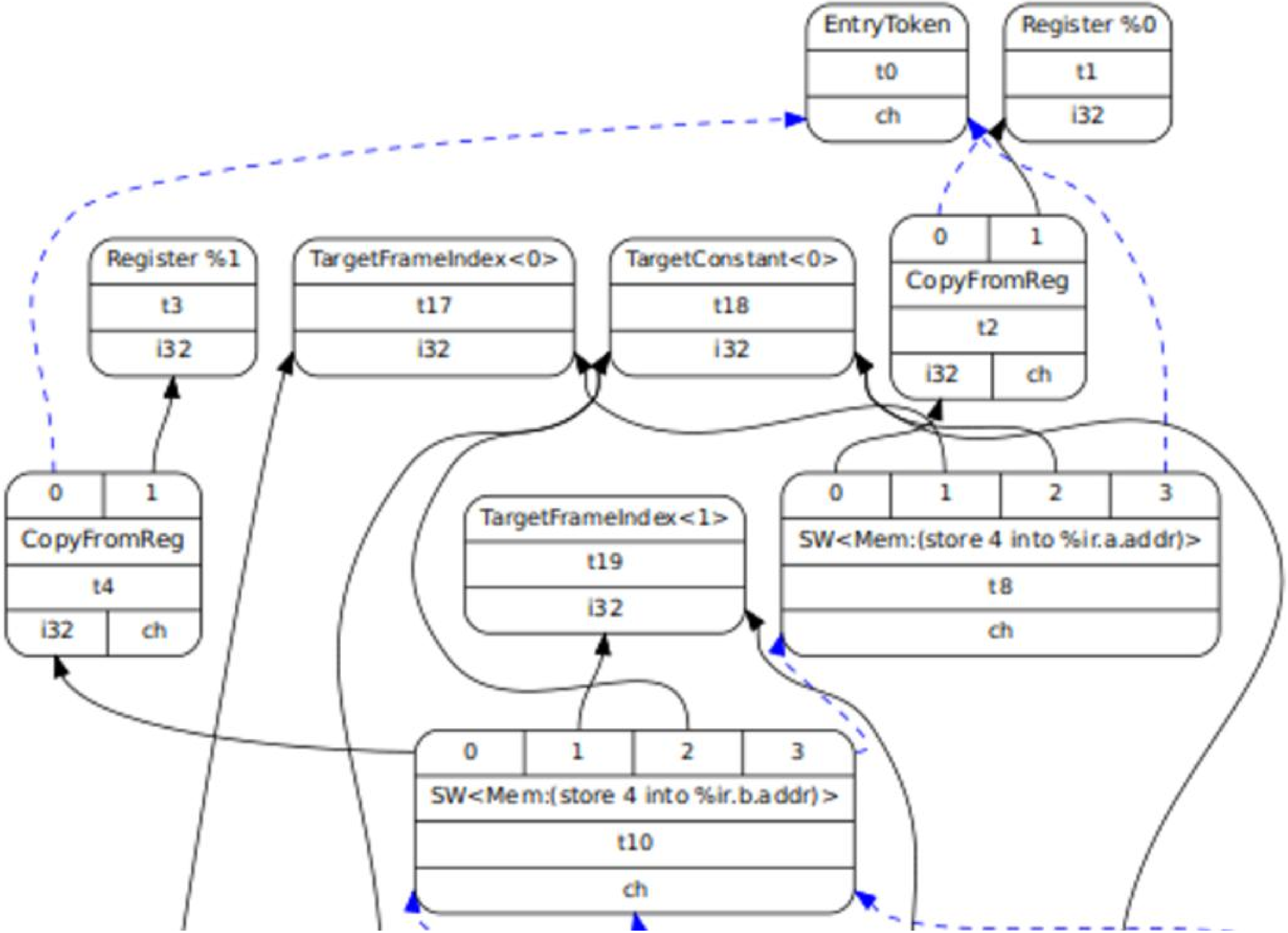


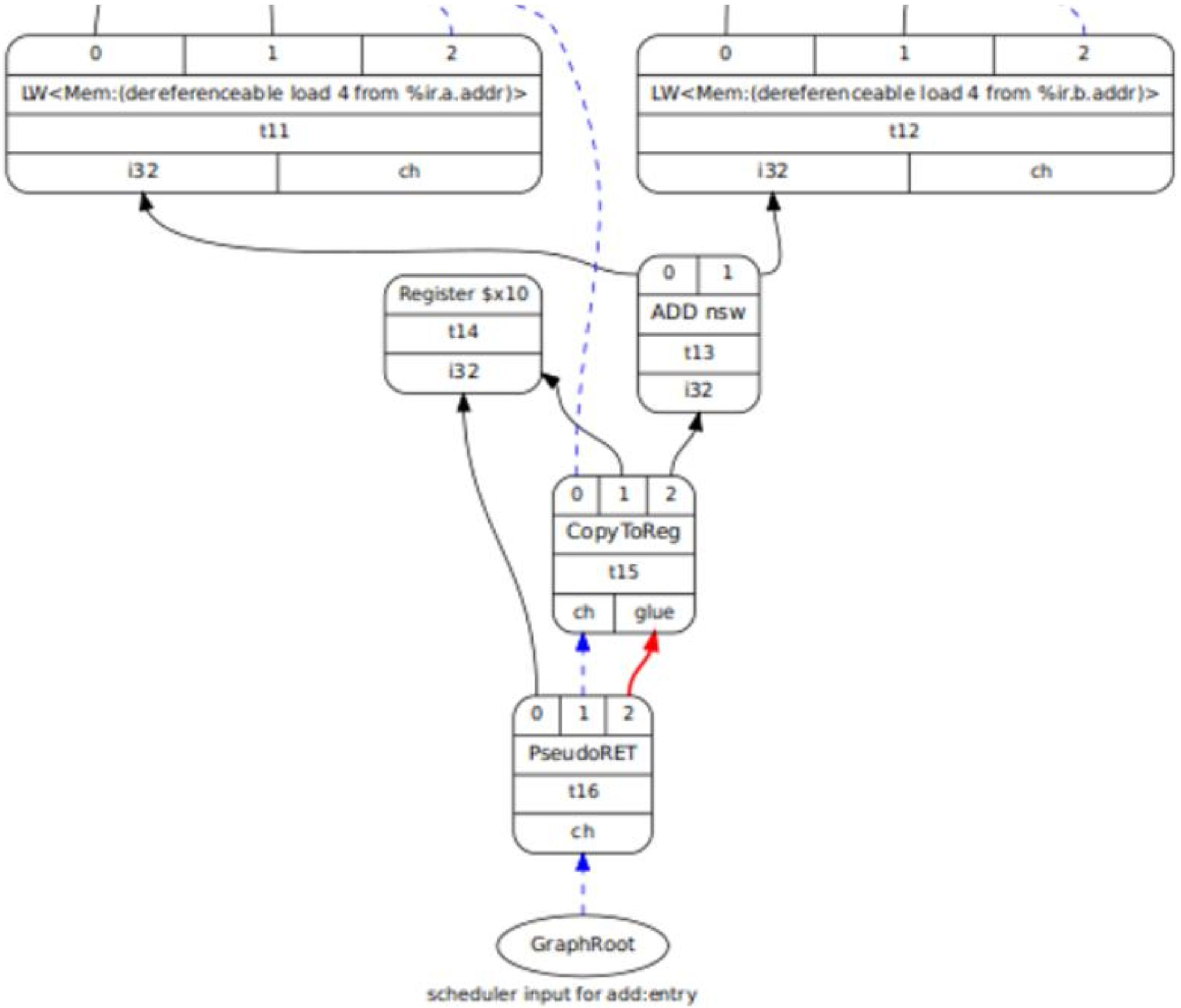
isel input for add:entry

指令选择

指令选择会调用InstructionSelect()函数，从root结点开始，从后往前遍历SDNode，进入X86GenDAGISel.inc:SelectCode函数，该函数会为SelectionDAG上的每个SDNode根据操作码调用相关的函数，然后将该子树替换成机器指令模式(此处会略过BasicBlock, Register, TargetConstant, TokenFactor, EntryToken, CopyFromReg,CopyToReg这些操作码)。如ISD::ADD会调用Select_ISD_ADD_i32()函数将该子树替换X86ISD::ADD32rr指令模版，ISD::LOAD会被替换为X86ISD::MOV32rm，ISD::Store指令会被替换为X86ISD::MOV32mr。当然由于此处未开启优化，所以add没有匹配X86ISD::ADD32rm模版。

clang -cc1 -S -triple=riscv32 add.c -view-sched-dags，查看指令选择之后如图所示：


EntryToken
t0
ch
Register %0
t1
i32
0 1
CopyFromReg
t2
i32 ch
Register %1
t3
i32
TargetFrameIndex<0>
t17
i32
TargetConstant<0>
t18
i32
0 1
CopyFromReg
t4
i32 ch
TargetFrameIndex<1>
t19
i32
0 1 2 3
SW<Mem:(store 4 into %ir.a.addr)>
t8
ch
0 1 2 3
SW<Mem:(store 4 into %ir.b.addr)>
t10
ch


0
1
2
LW<Mem:(dereferenceable load 4 from %ir.a.addr)>
t11
i32
ch
0
1
2
LW<Mem:(dereferenceable load 4 from %ir.b.addr)>
t12
i32
ch
Register $x10
t14
i32
0
1
ADD nsw
t13
i32
0
1
2
CopyToReg
t15
ch
glue
0
1
2
PseudoRET
t16
ch
GraphRoot
scheduler input for add:entry

指令调度

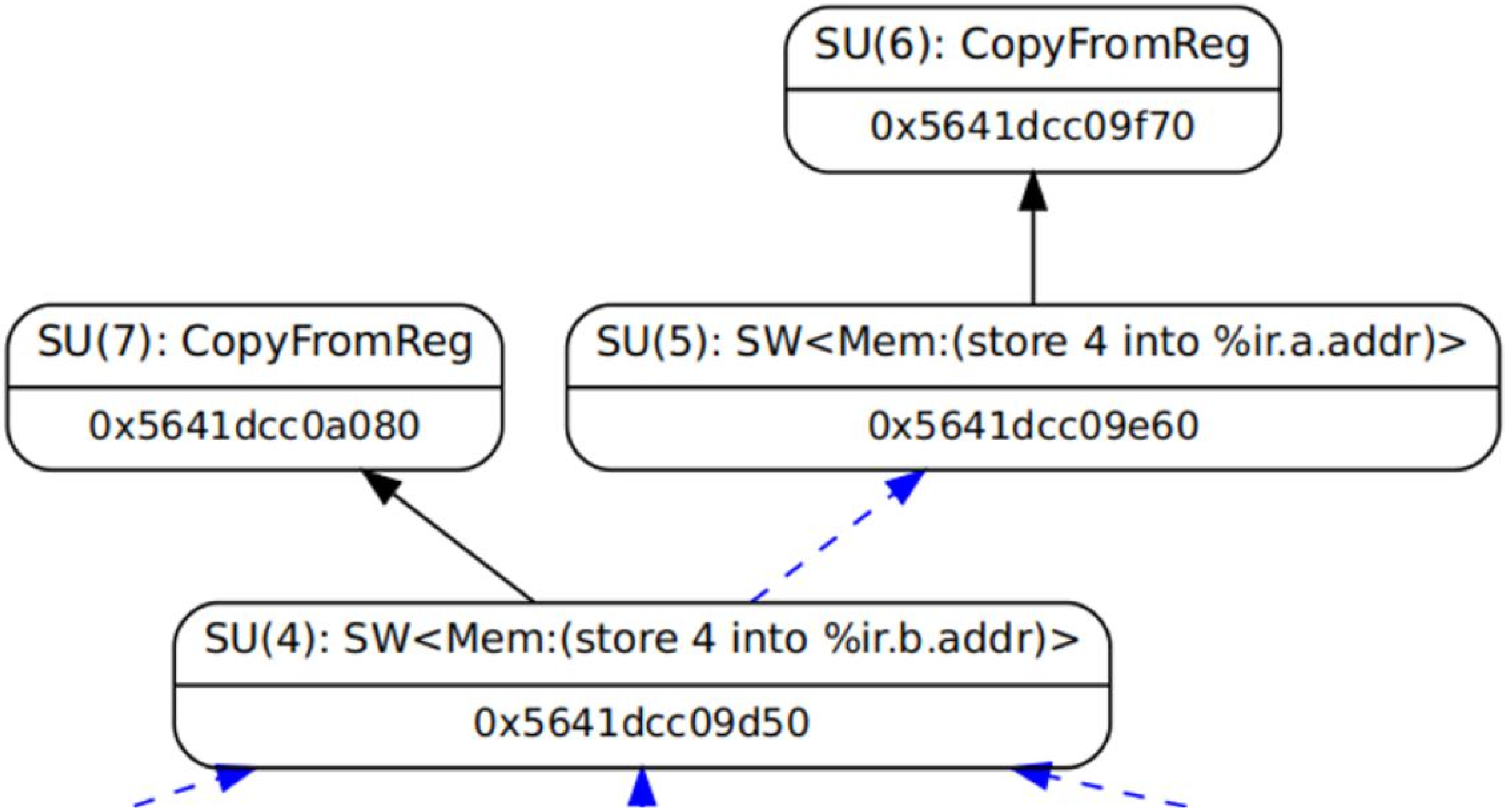
使用命令clang-cc -S -triple=riscv32 add.c -view-sunit-dags (X)

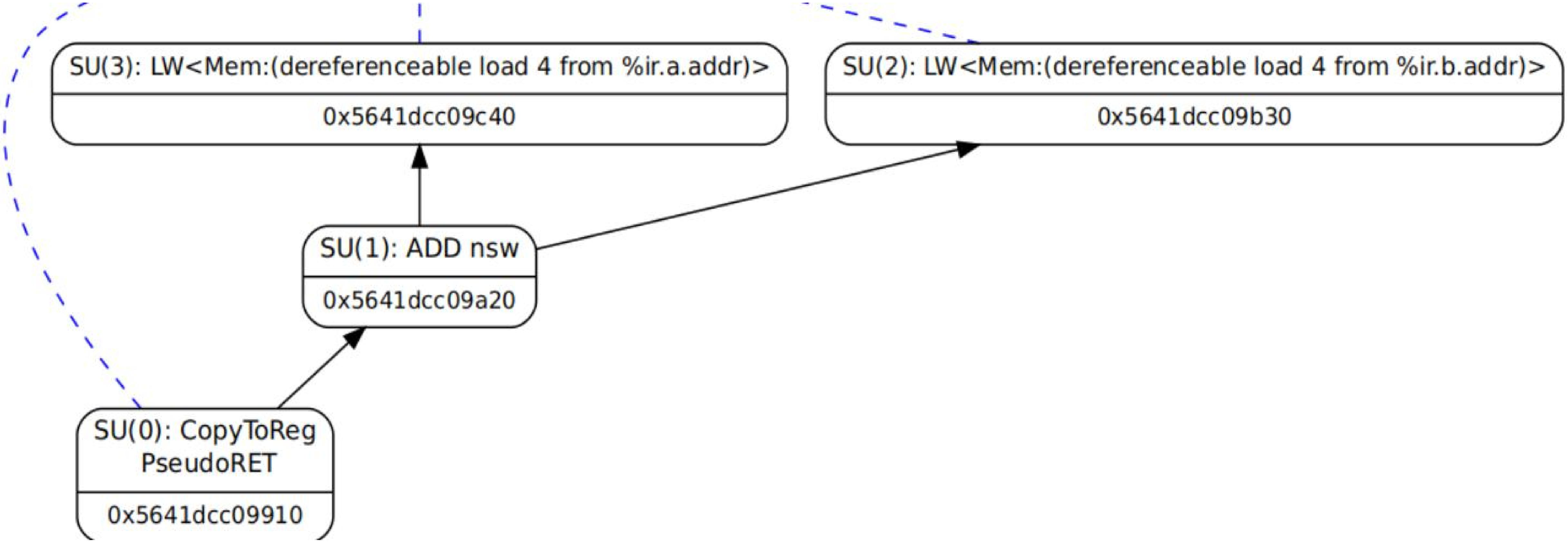
llc -view-sunit-dags add.ll (√)

查看最终经过ScheduleDAGSDFast指令调度器生成的ScheduleDAG如图所示：

代码发射

经过CodeEmit，最终生成的机器相关的machine basic block（待做）


SU(6): CopyFromReg
0x5641dcc09f70
SU(7): CopyFromReg
0x5641dcc0a080
SU(5): SW<Mem:(store 4 into %ir.a.addr)>
0x5641dcc09e60
SU(4): SW<Mem:(store 4 into %ir.b.addr)>
0x5641dcc09d50


SU(3): LW<Mem:(dereferenceable load 4 from %ir.a.addr)>
0x5641dcc09c40
SU(2): LW<Mem:(dereferenceable load 4 from %ir.b.addr)>
0x5641dcc09b30
SU(1): ADD nsw
0x5641dcc09a20
SU(0): CopyToReg PseudoRET
0x5641dcc09910

```
# Machine code for add():
  <fi#-2>: size is 4 bytes, alignment is 1 byte, fixed at location [SP+8]
  <fi#-1>: size is 4 bytes, alignment is 1 byte, fixed at location [SP+4]
  <fi#0>: size is 4 bytes, alignment is 4 bytes, at location [SP+4]
  <fi#1>: size is 4 bytes, alignment is 4 bytes, at location [SP+4]
  <fi#2>: size is 4 bytes, alignment is 4 bytes, at location [SP+4]
Live Outs: EAX
```

```
entry: 0x371b008, LLVM BB @0x36d5780, ID#0:
    %reg1024<def> = MOV32rm <fi#-2>, 1, %reg0, 0, %reg0, Mem:LD(4,4) [FixedStack-2 + 0]
    %reg1025<def> = MOV32rm <fi#-1>, 1, %reg0, 0, %reg0, Mem:LD(4,4) [FixedStack-1 + 0]
    MOV32mr <fi#1>, 1, %reg0, 0, %reg0, %reg1025, Mem:ST(4,4) [a.addr + 0]
    MOV32mr <fi#2>, 1, %reg0, 0, %reg0, %reg1024, Mem:ST(4,4) [b.addr + 0]
    %reg1026<def> = MOV32rm <fi#1>, 1, %reg0, 0, %reg0, Mem:LD(4,4) [a.addr + 0]
    %reg1027<def> = ADD32rr %reg1026, %reg1024, %EFLAGS<imp-def>
    MOV32mr <fi#0>, 1, %reg0, 0, %reg0, %reg1027, Mem:ST(4,4) [retval + 0]
    %EAX<def> = MOV32rr %reg1027
    RET

# End machine code for add().
```

上述的指令未经任何优化，所以生成的代码不是最优的。存在冗余的指令，并且没有利用X86的指令集优势(add操作可以使用内存操作数)，但是用于叙述原理是完全足够的。

• 03 参考资料

LLVM中，使用LLC生成可视化SelectionDAG

https://blog.csdn.net/qq_27885505/article/details/80366525

基于DAG覆盖的SelectionDAG

https://www.zhihu.com/question/64887261/answer/424114917

LLVM基础(IR简介&CFG图生成&可视化

https://www.jianshu.com/p/707c4f8af150

谢谢

欢迎交流合作

2019/02/25