11 指令选择

徐辉，xuh@fudan.edu.cn

本章学习目标：

- 了解 AArch64 指令集
- 掌握基础的 AArch64 指令和函数调用规约
- 了解指令选择图和铺树问题

本章学习如何将 IR 代码翻译为 AArch64 指令集的汇编代码，目标指令集版本是 ARM-v8A 版本。我们先介绍单条 IR 指令对应的 AArch64 指令，然后介绍针对 IR 代码块翻译的指令选择问题和解法。我们暂且不考虑具体的寄存器数目，均使用 w0...wn（32 位寄存器）或 x0...xn（64 位寄存器）表示。

11.1 翻译 AArch64 指令

ARM-v8a 是精简指令集，其主要特点是内存的存取与运算由不同的指令分别完成。代码 11.1展示了一段简单的 hello world 汇编代码。其中函数和全局变量均以冒号定义，如 `_main` 和 `l_.str`；立即数则以 `#` 开头，如 `#0`。

```
  .globl  _main                           ; -- Begin function main
  .p2align  2
_main:                                    ; @main
  stp x29, x30, [sp, #-16]!               ; 16-byte Folded Spill
  mov x29, sp
  adrp  x0, l_.str@PAGE
  add x0, x0, l_.str@PAGEOFF
  bl  _printf
  mov w0, #0
  ldp x29, x30, [sp], #16                 ; 16-byte Folded Reload
  ret
l_.str:                                   ; @.str
  .asciz  "Hello, World!"
```

代码 11.1: Helloworld 汇编代码示例

表 11.1介绍了主要 IR 指令对应的 ARM-v8A 汇编指令翻译方法。由于 ARM-v8A 指令是 32 位定长编码的，对于参数是立即数的情况支持比较有限，如 add、mov 等指令只能支持有限的立即数作为参数。如果遇到不支持的立即数则需要通过 mov 操作将其保存到寄存器中，如通过 `mov x8, #3` 和 `movk x8, #1, lsl #16` 两条指令将 x8 的值设置为 65539。其中 movk 指令表示保持寄存器其余位不变。

ARM-v8A 支持以下寻址模式：

```
ldr x2, [x1]
ldr x2, [x1, #10] ; x2 = [x1 + 10]
ldr x2, [x1, x0] ; x2 = [x1 + x0]
ldr x2, [x1, #10]! ; x1 = x1 + 10, x2 = [x1]
ldr x2, [x1], #10 ; x2 = [x1], x1 = x1 + 10
ldr x2, [x1, x0, lsl #3] ; x2 = [x1 + x0 * 8]
```

表 11.1: ARM-v8A 指令及其应用

IR 指令	ARM-v8A 指令	说明
`%a = alloca i32`	`sub sp, sp, #16`	在栈帧上为局部变量分配内存空间；sp 指针要求 16 字节对齐
`store i32 %0, i32* %a`	`str w0, [sp, #12]`	将寄存器值保存到内存地址 sp+12
`store i32 1, i32* %a`	`mov w0 #1` `str w0, [sp, #12]`	将整数保存到栈空间
`%a0 = load i32, i32* %a`	`ldr w0, [sp, #12]`	将局部变量值由内存地址 sp+12 加载到寄存器
`%g0 = load i32, i32* @g`	`adrp x8, g` `ldr w0, [x8, :lo12:g]`	将全局变量值加载到寄存器
`%a0 = add i32 %a1, %a2`	`add w0, w1, w2`	两个寄存器的值相加，结果保存到 w1
`%a0 = add i32 %a1, 4095`	`add w0, w1, #4095`	整数范围：$x \in [0, 2^{12})$ 以及 $x * 2^{12}$
`%a0 = sub i32 %a1, %a2`	`sub w0, w1, w2`	两个寄存器的值相减；亦支持减立即数，方法同加法
`%a0 = mul i32 %a1, %a2`	`mul w0, w1, w2`	不支持立即数
`%a0 = sdiv i32 %a1, %a2`	`sdiv w0, w1, w2`	不支持立即数
`%a0 = icmp sgt i32 %a1, %a2` `br i1 %a1, label %bb1, label %bb2`	`cmp w0, w1` `b.le .LBB2`	比较两个寄存器的值，结果保存到 CPSR 寄存器，然后条件跳转
`%a0 = xor i1 %a1, %a2`	`eor w0, w1, w1`	异或运算，支持一个立即数
`%a0 = xor i1 %a1, %a2`	`bl foo`	函数调用
`ret i32 %a0`	`ldr x30, [sp], #16` `ret`	将返回值存入 x30 寄存器，还原栈顶指针，返回

ARM-v8A 提供了非常多的指令，同学们可以参考 ARM 公司提供的官方手册 [1]。

11.2 指令选择问题

通过上一节的介绍，我们很容易为单条 IR 指令找到对应的汇编代码翻译方式。但是仅考虑单条 IR 指令翻译得到的汇编代码未必是最优的，有些汇编指令可以对应多条 IR 指令，如一条复合算数运算指令可以同时对应 IR 中的两条指令：乘法和加法或减法。本节介绍的指令选择问题主要针对的是该情况。

```
madd x0, x1, x2, x3 ; x0 = x1 * x2 + x3; mul指令本质上是该指令在x3=0时的特例
msub x0, x1, x2, x3 ; x0 = x1 * x2 - x3;
```

下面我们主要探讨针对单个代码块的指令选择问题。由于一个函数由多个代码块组成，分治解决即可。

11.2.1 指令选择图

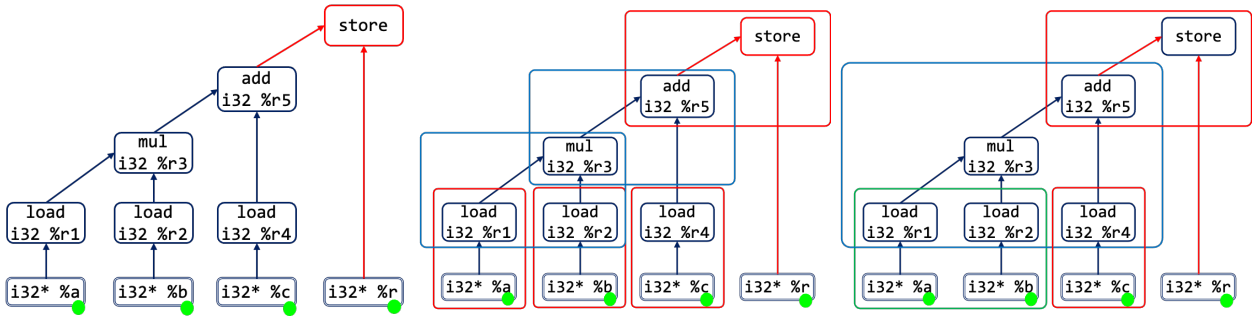
为了解决指令选择问题，我们将该单个代码块内的 IR 表示为指令选择图。

定义 1 (指令选择图)**.** 指令选择图是一个有向无环图，包括两种类型的节点，指令节点和参数节点，且指令的运算结果和该指令本身在图上表示为一个点；其中的边表示指令运行所需的参数。

以下列 LLVM IR 代码（代码 11.2）为例，其指令选择图可表示为图 11.1a。

```
%r1 = load i32 %a;
%r2 = load i32 %b;
%r3 = mul i32 %r1, %r2;
%r4 = load i32 %c;
%r5 = add i32 %r3, %r4;
store i32 %r5, %r;
```

代码 11.2: LLVM IR 代码



(a) 指令选择图　(b) 指令选择方案 1　(c) 指令选择方案 2

图 11.1: 指令选择问题举例

11.2.2 铺树问题

通过指令选择图，我们可以将指令选择问题转化成铺树（图）问题，即如何翻译汇编指令，使其可以覆盖指令选择图上的所有节点，同时使得目标汇编代码指令数最少、运行时间最优。以图 11.1a为例，该图至少包含图 11.1b和图 11.1c中的两种铺树方案。其对应的汇编代码分别为代码 11.3和代码 11.4。我们很容易看出代码 11.4对应的指令数更少。如果 ldr 和 ldp 指令的性能开销以及 mul 和 madd 指令的性能开销都相同，则明显图 11.1c对应的铺树方案更优。

铺树问题是一个 NP-hard 问题，可以通过贪心法或动态规划求解。一种常用的贪心法称为 Maximal Munch，即每步选择覆盖节点最多的方案进行铺树。

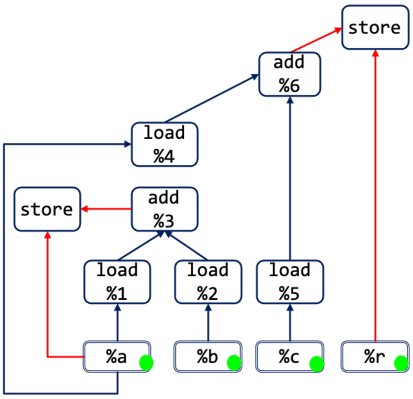
```
ldr %w1, [sp, #a]
ldr %w2, [sp, #b]
ldr %w3, [sp, #c]
mul %w3, %w1, %w2
add %w5, %w3, %w4
store %w5, [sp, #r]
```

代码 11.3: 图 11.1b对应的汇编代码

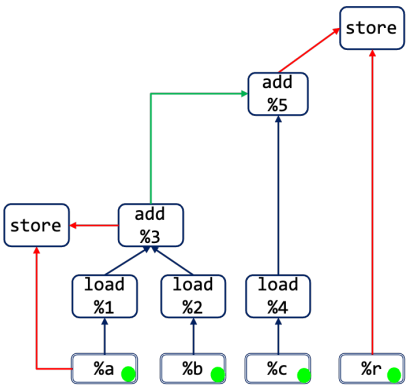
```
ldp %w1, %w2, [sp, #a]
ldr %w3, [sp, #c]
madd %w5, %w1, %w2, %w4
store %w5, [sp, #r]
```

代码 11.4: 图 11.1c对应的汇编代码

指令选择树仅包含了指令间的依赖关系，通过拓扑排序即可生成满足依赖顺序的汇编指令序列。但是由于拓扑排序序列不唯一，可能会出现对针同一变量存在多种 load-store 的顺序情况，并且不同的顺序含义完全不同。以图 11.2a为例，其中包含对变量 a 的两次 load 和一次 store。但如果先将 IR 进行 SSA 优化再进行指令选择（图 11.2b），同一代码块中至多存在针对同一变量的一次 load 和 store，且 load 一定是在 store 之前。因此在翻译汇编代码时先 ldr 后 str 即可。



(a) load-store 顺序问题示例：针对变量 a 有两次 load，一次 store



(b) SSA 优化后的指令选择树：先 load，后 store

图 11.2: 指令选择中的 load-store 顺序问题

Bibliography

[1] Arm® Architecture Reference Manual for A-profile architecture. https://developer.arm.com/documentation/ddi0487/latest/, 2023.