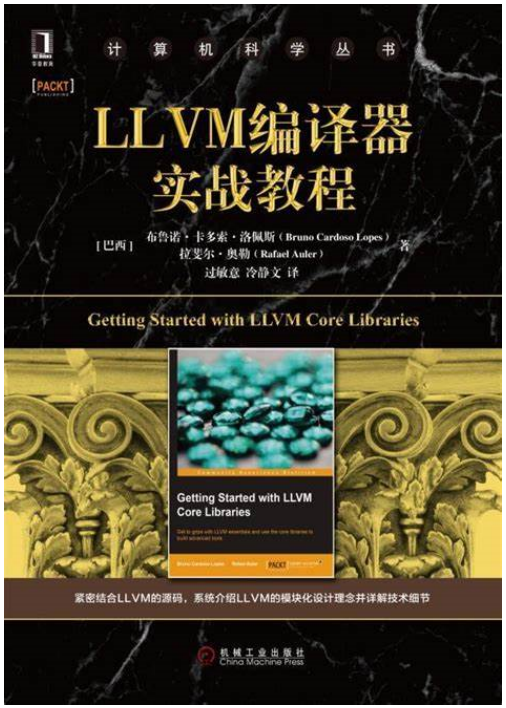
五、目标代码生成
(16. 指令选择)

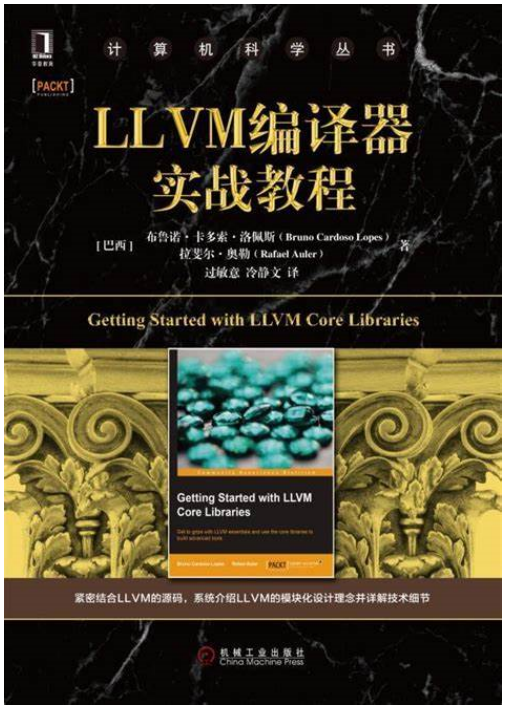
魏恒峰

hfwei@nju.edu.cn

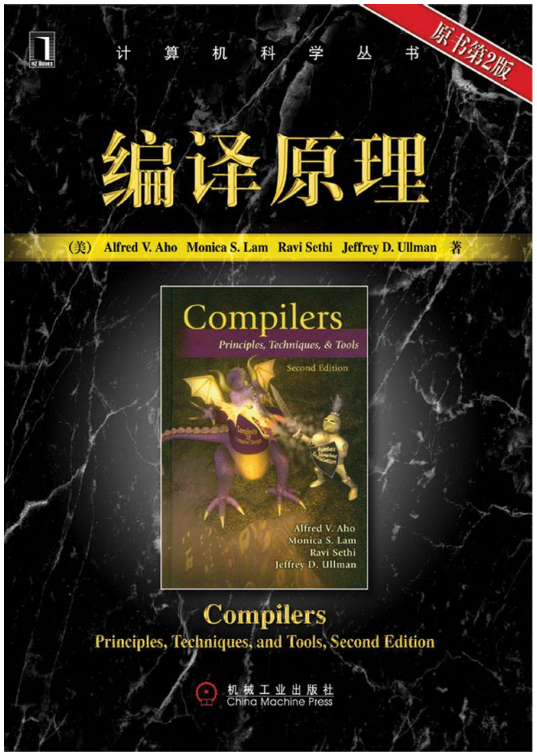
2024 年 06 月 12 日

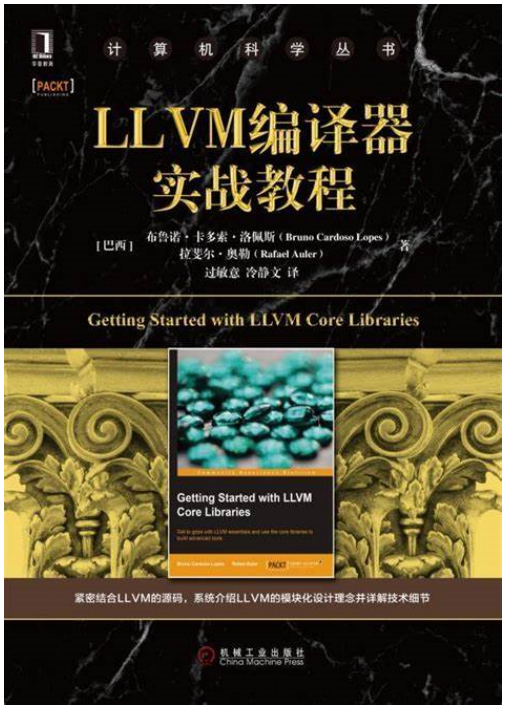


https://llvm.org/docs/CodeGenerator.html

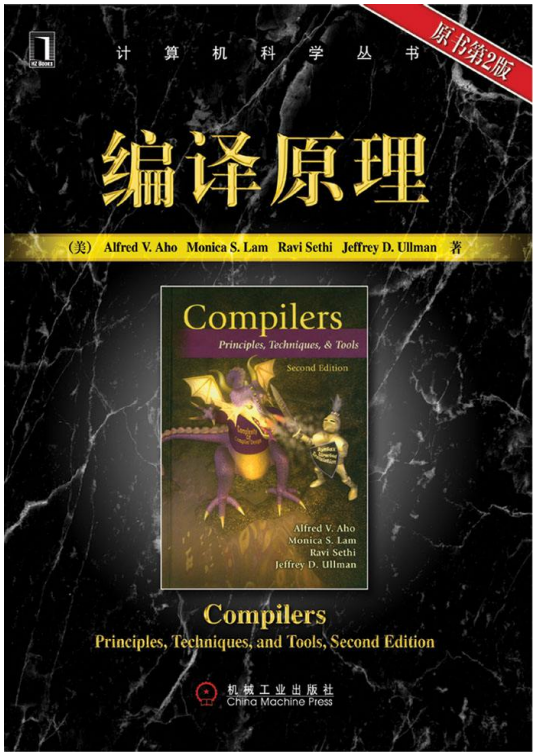


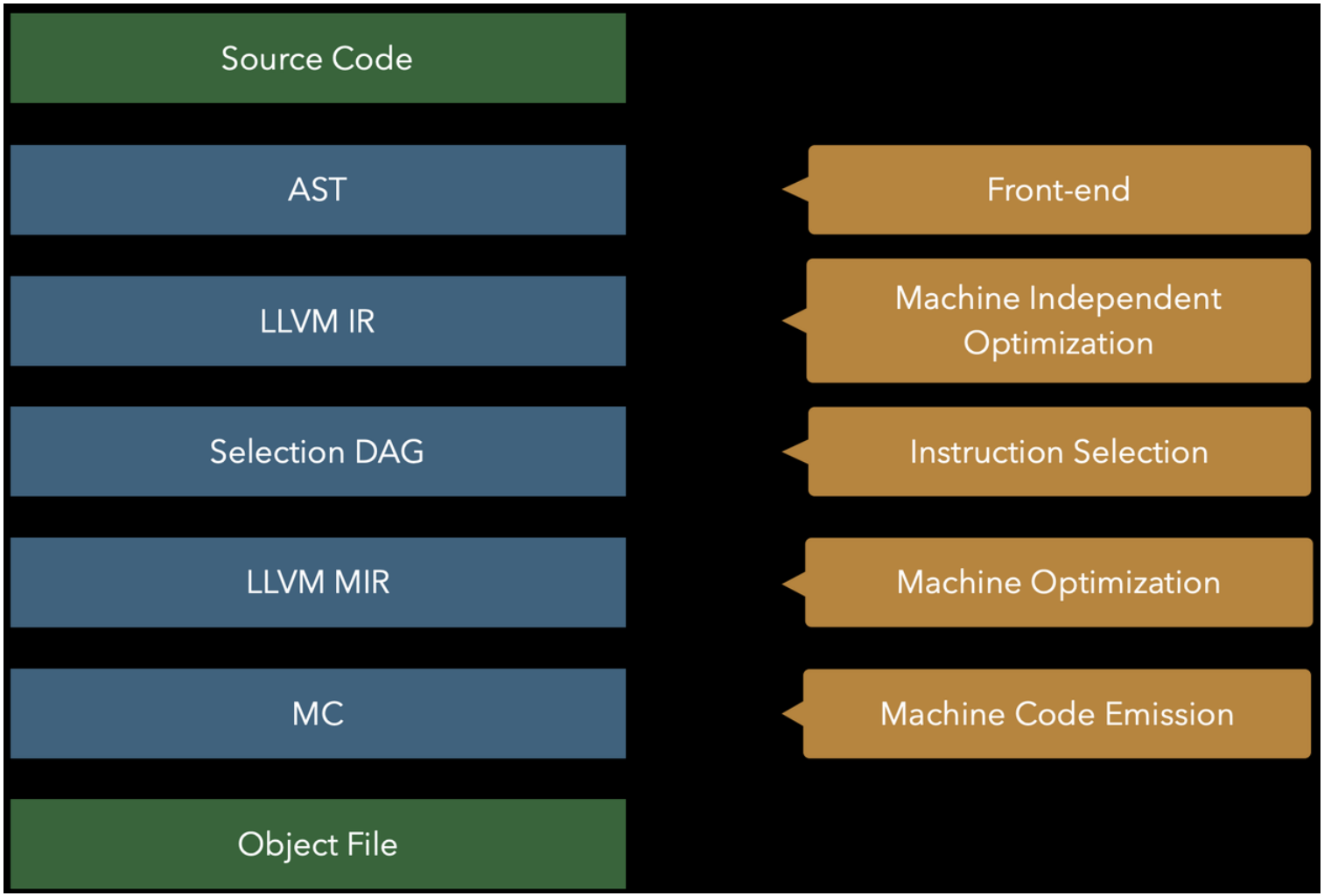
https://llvm.org/docs/CodeGenerator.html



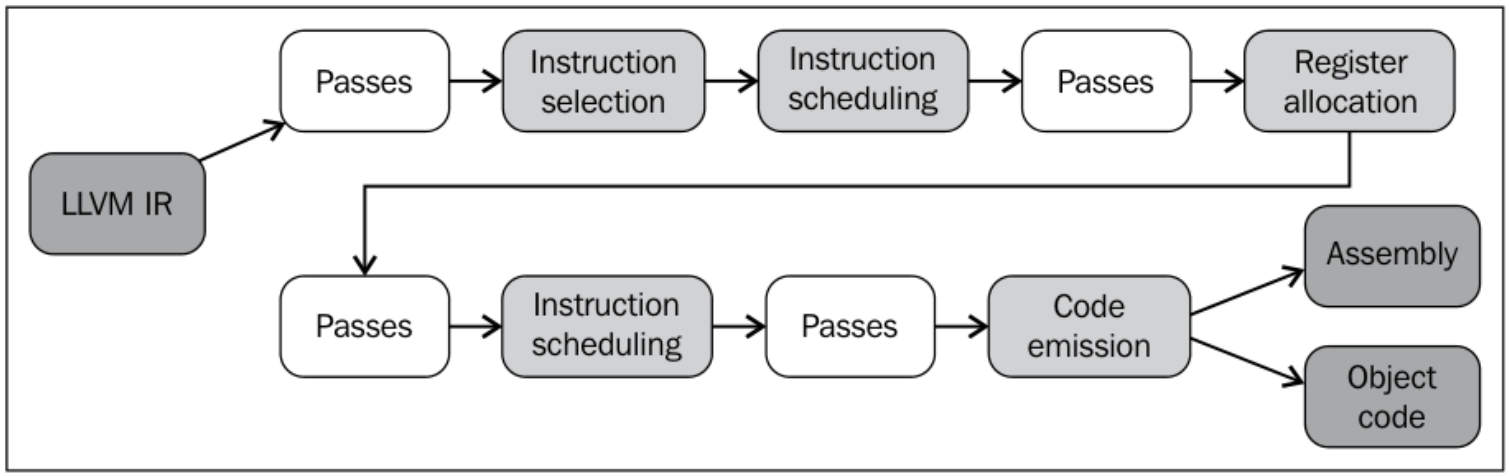


https://llvm.org/docs/CodeGenerator.html

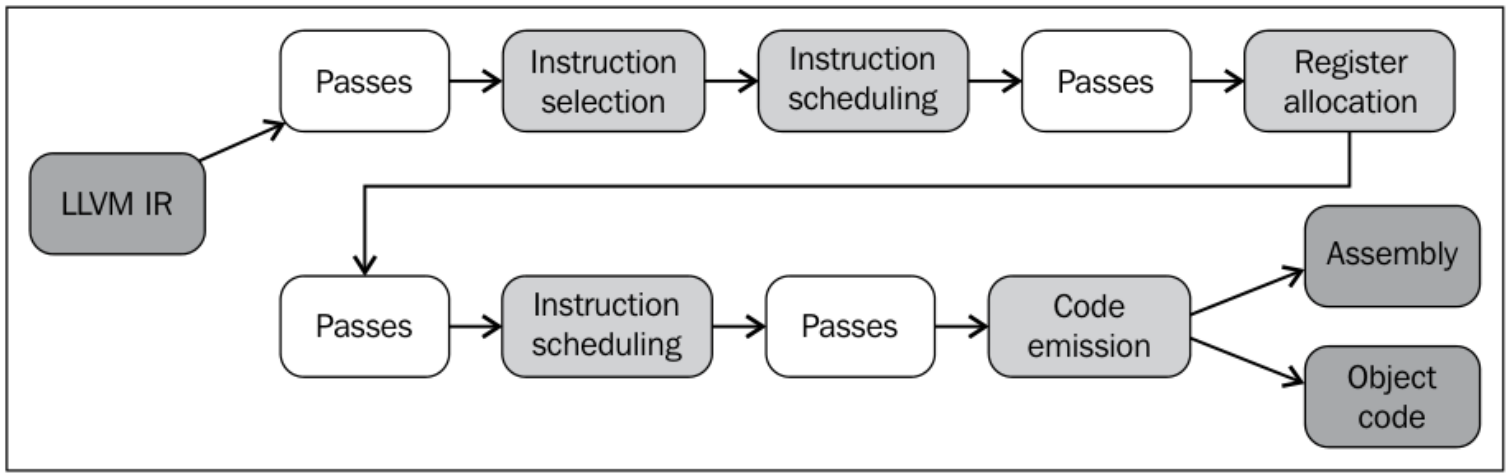



Source Code
AST
LLVM IR
Selection DAG
LLVM MIR
MC
Object File
Front-end
Machine Independent Optimization
Instruction Selection
Machine Optimization
Machine Code Emission

(in-memory) LLVM IR SelectionDAG MachineInstr MCInst

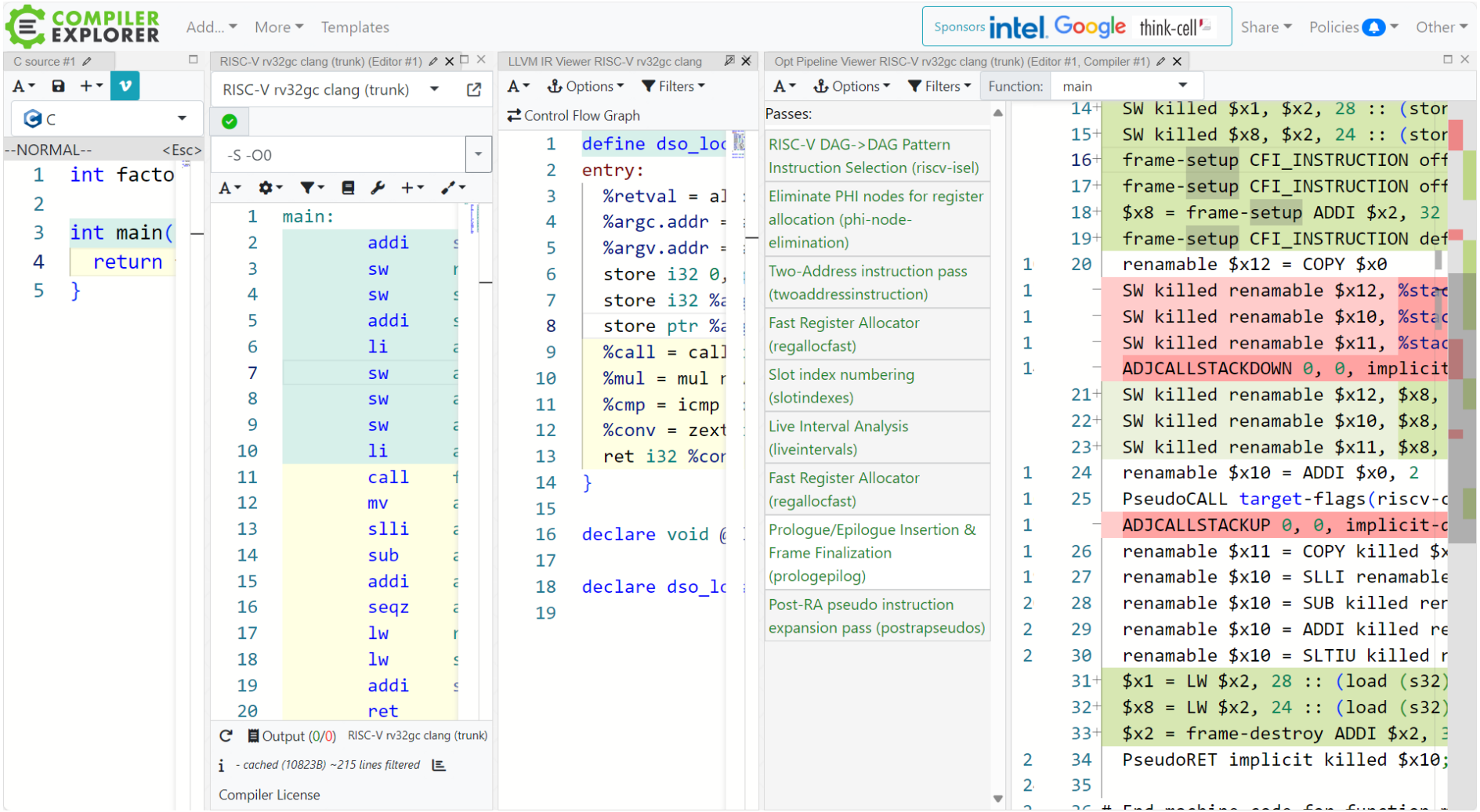


(in-memory) LLVM IR SelectionDAG MachineInstr MCInst

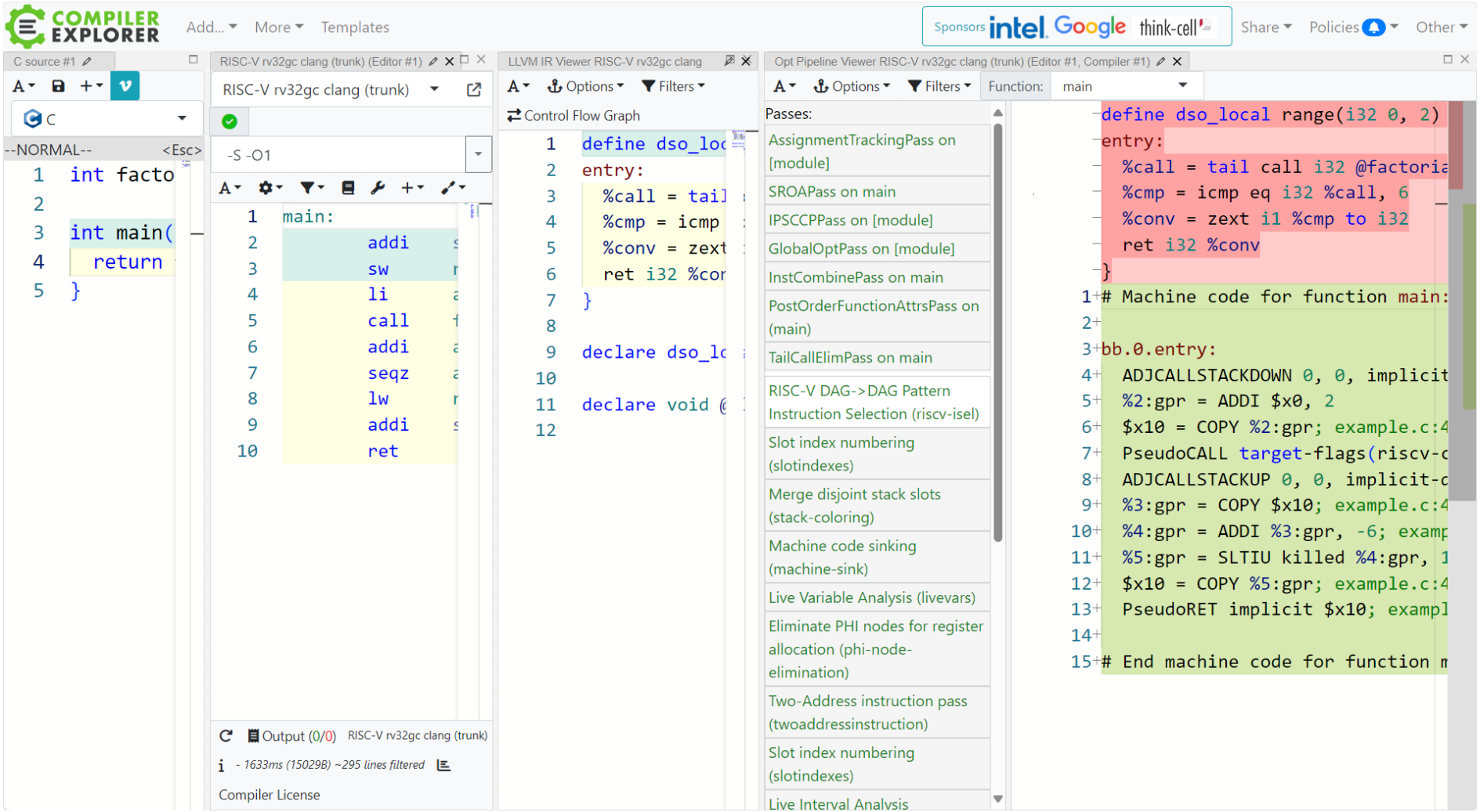


Where is “Prologue/Epilogue Insertion”?

f0-O0 @ Compiler Explorer

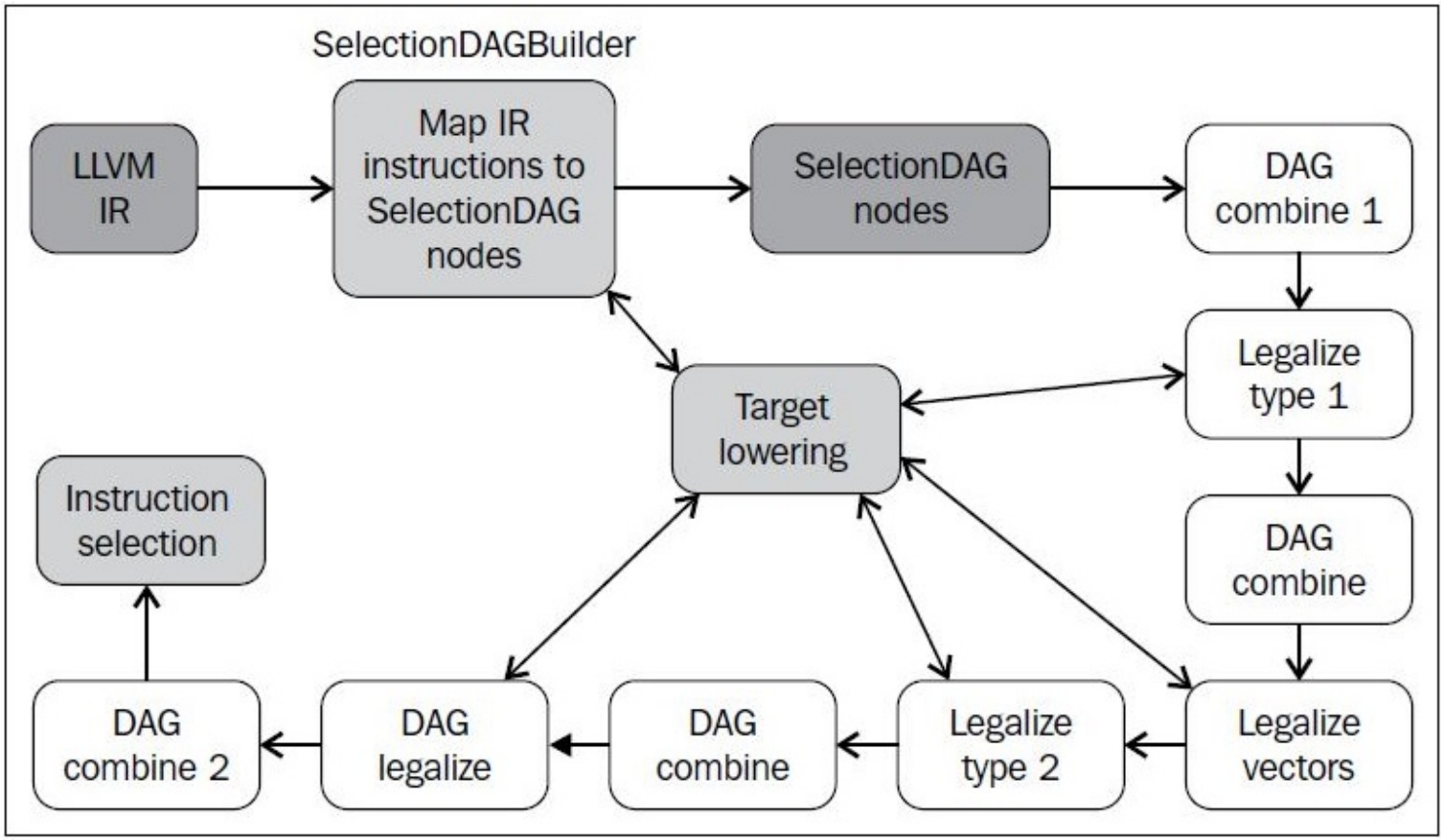


f0-O1 @ Compiler Explorer

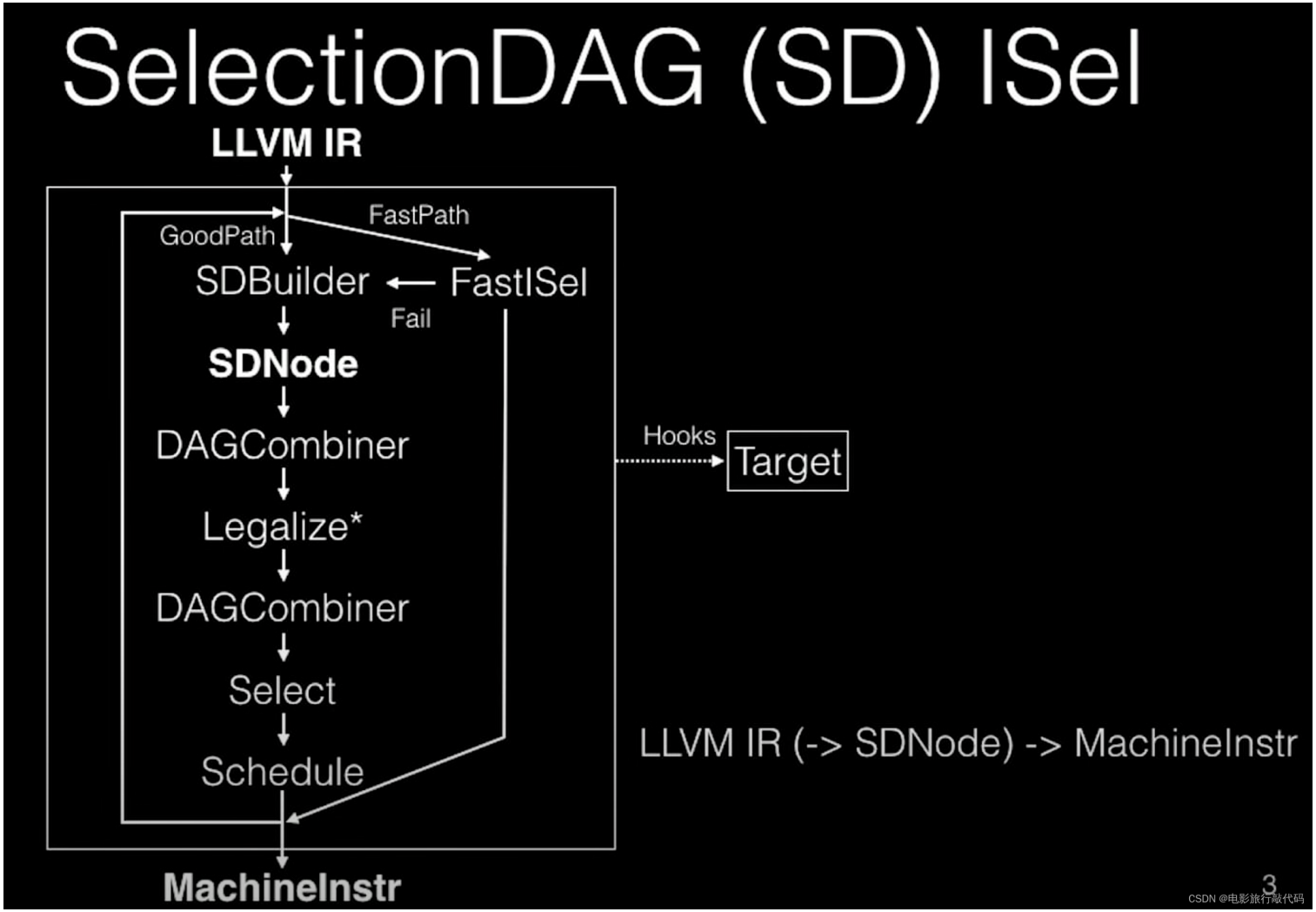


Instruction Selection

Instruction Selection is the process of translating LLVM code presented to the code generator into target-specific machine instructions. There are several well-known ways to do this in the literature. LLVM uses a SelectionDAG based instruction selector.


SelectionDAGBuilder
LLVM IR
Map IR instructions to SelectionDAG nodes
SelectionDAG nodes
DAG combine 1
Legalize type 1
DAG combine
Legalize vectors
Legalize type 2
DAG combine
DAG legalize
DAG combine 2
Instruction selection
Target lowering

SDISel FastISel (per basic block) GlobalISel (per function)



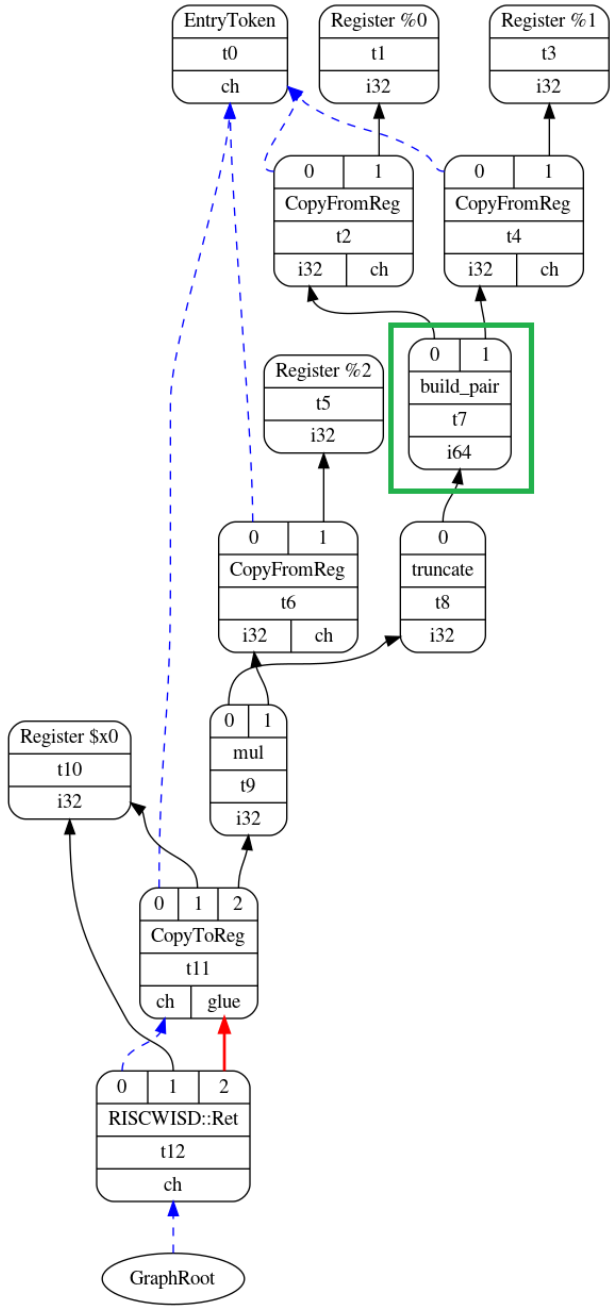
GlobalISel (per function)

```
unsigned int MUL(unsigned long long int x, unsigned int y)
{
    return x * y;
}
```

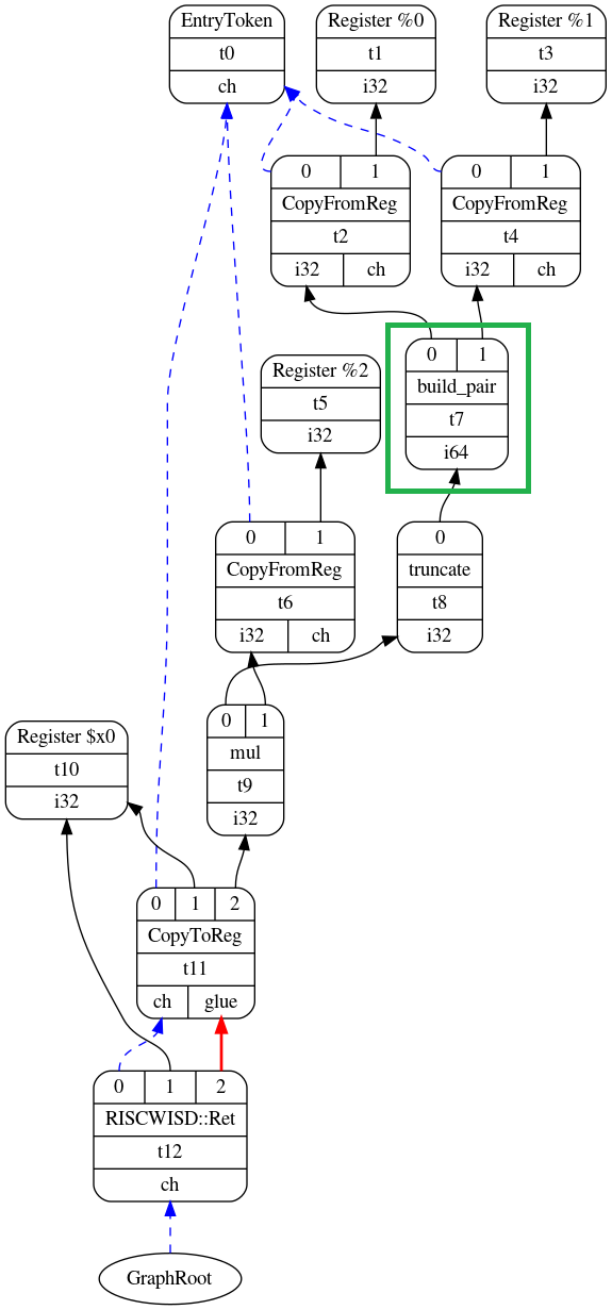
```
unsigned int MUL(unsigned long long int x, unsigned int y)
{
    return x * y;
}
```

```
define dso_local i32 @MUL(i64 %x, i32 %y) local_unnamed_addr #0 {
entry:
  %0 = trunc i64 %x to i32
  %conv1 = mul i32 %0, %y
  ret i32 %conv1
}
```

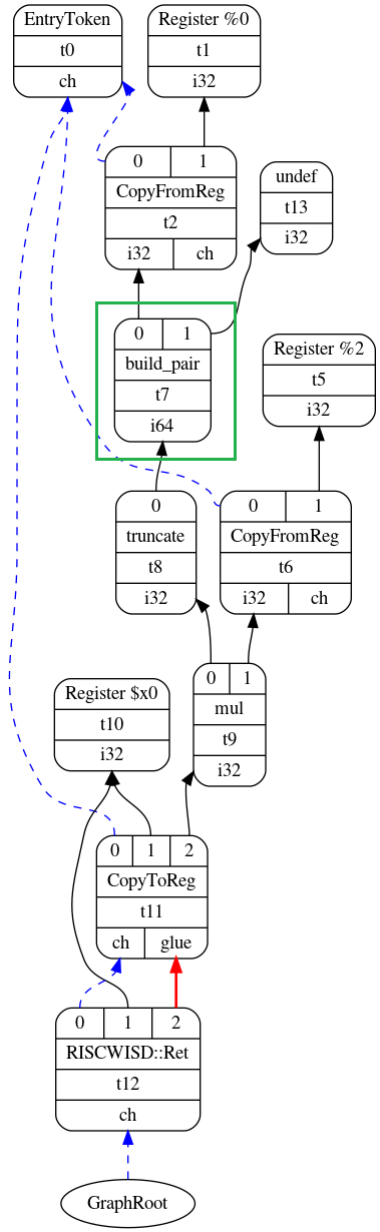
```
define dso_local i32 @MUL(i64 %x, i32 %y) local_unnamed_addr #0 {
entry:
  %0 = trunc i64 %x to i32
  %conv1 = mul i32 %0, %y
  ret i32 %conv1
}
```



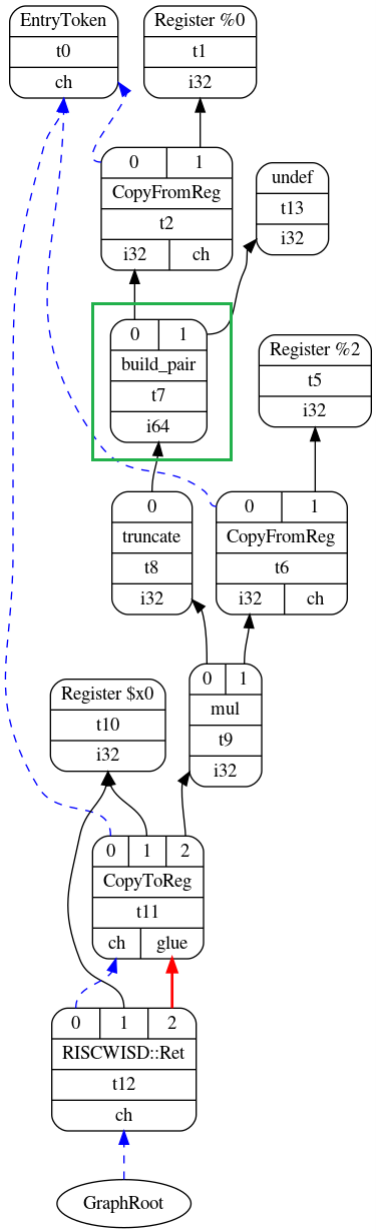
dag-combine1 input for MUL:entry



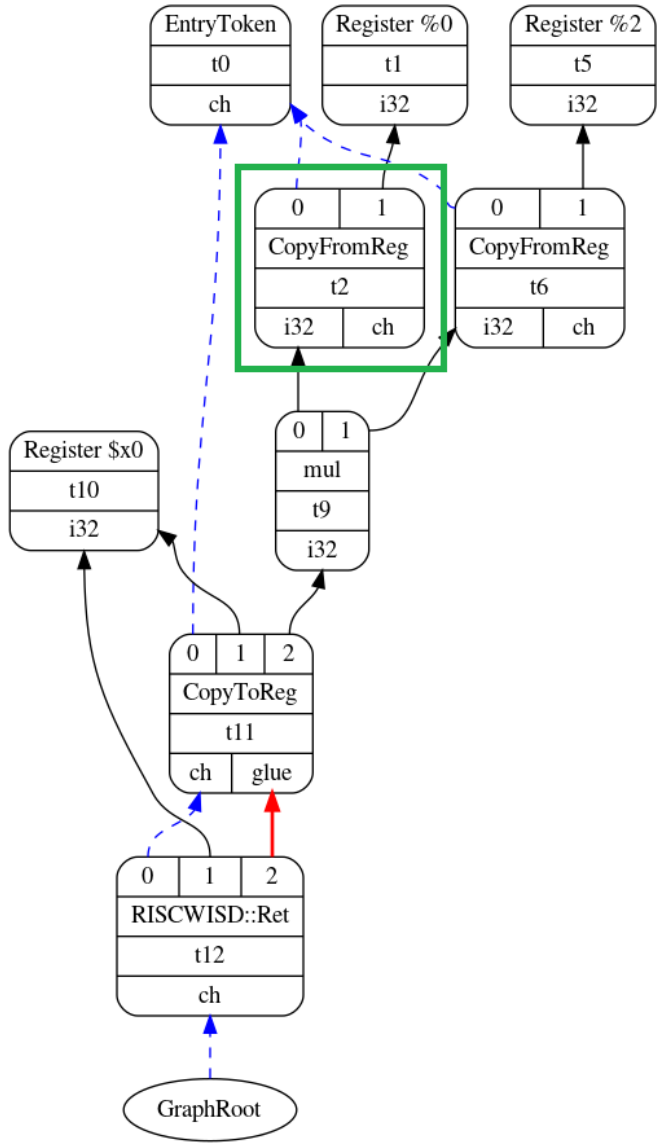
dag-combine1 input for MUL:entry



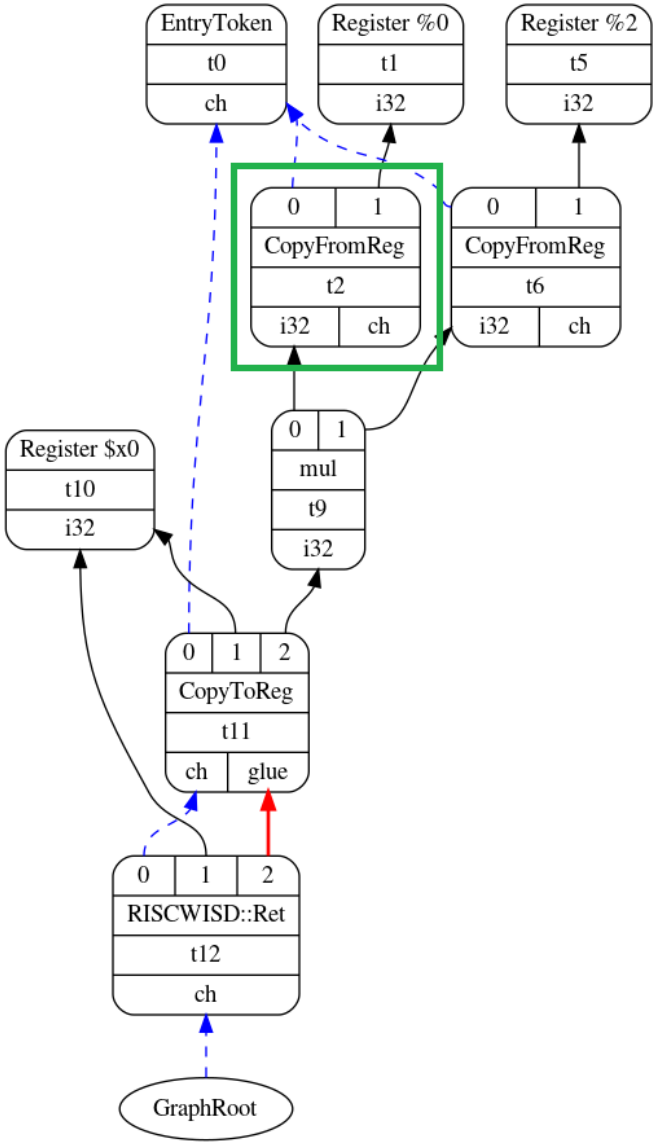
legalize-types input for MUL:entry



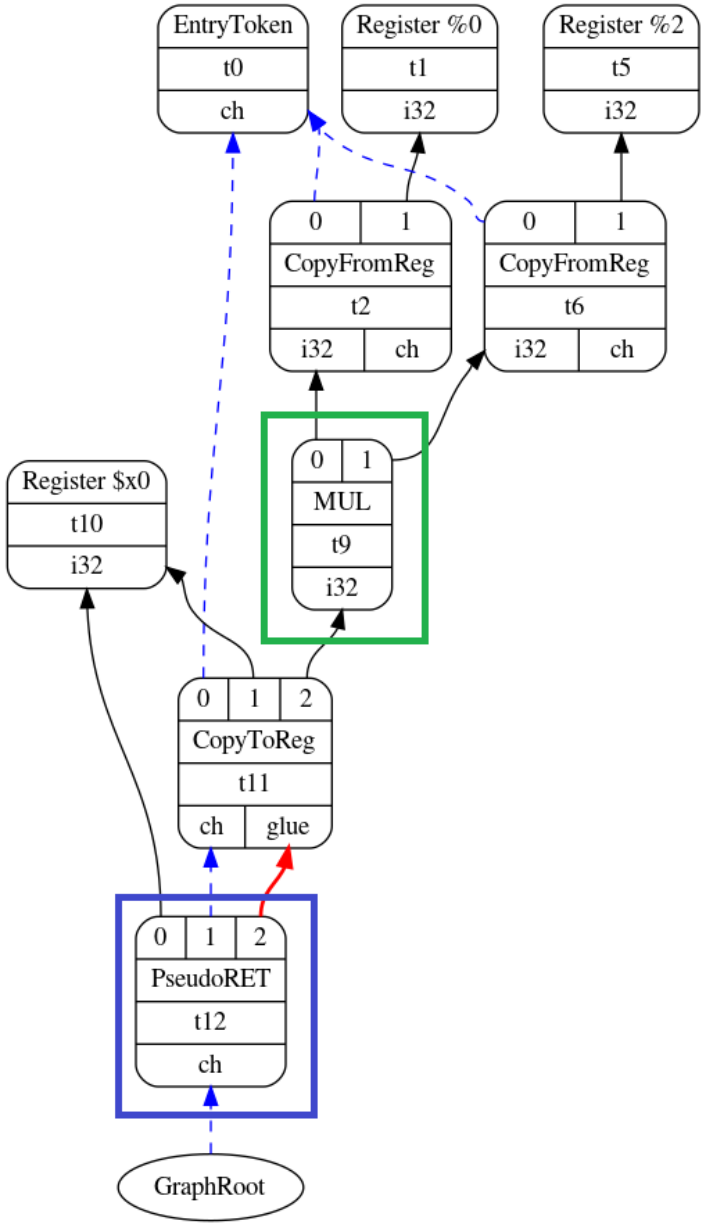
legalize-types input for MUL:entry



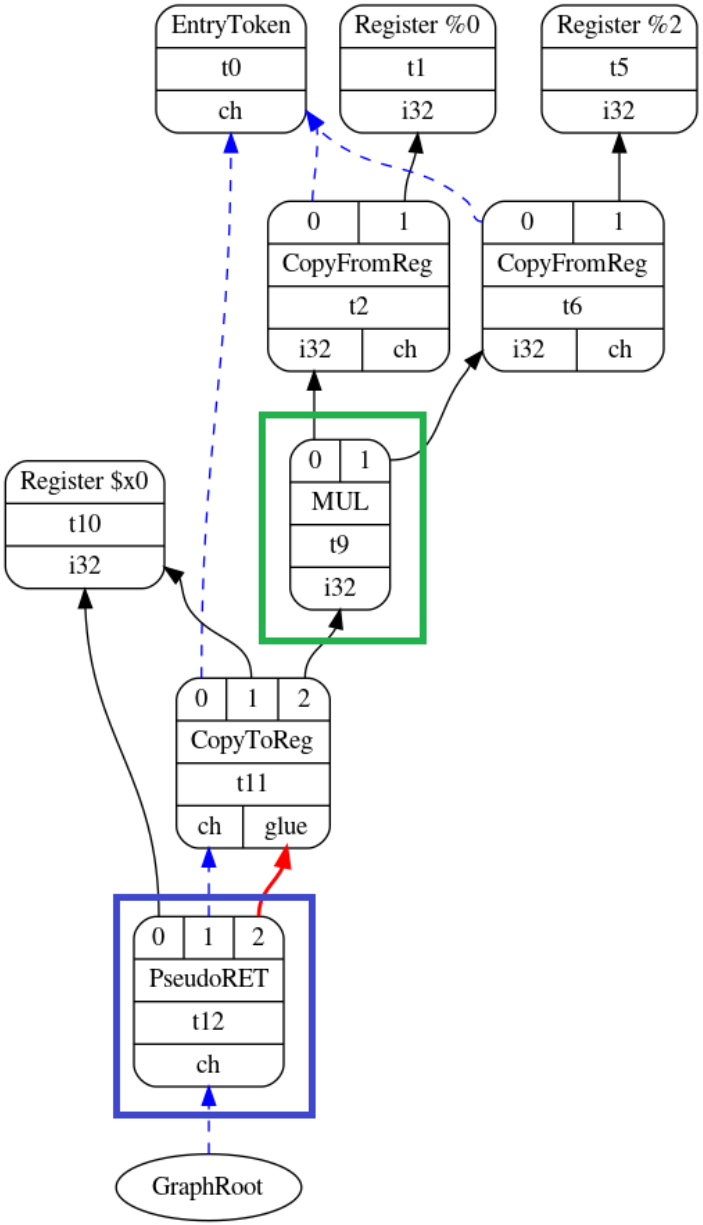
dag-combine2 input for MUL:entry



dag-combine2 input for MUL:entry



scheduler input for MUL:entry

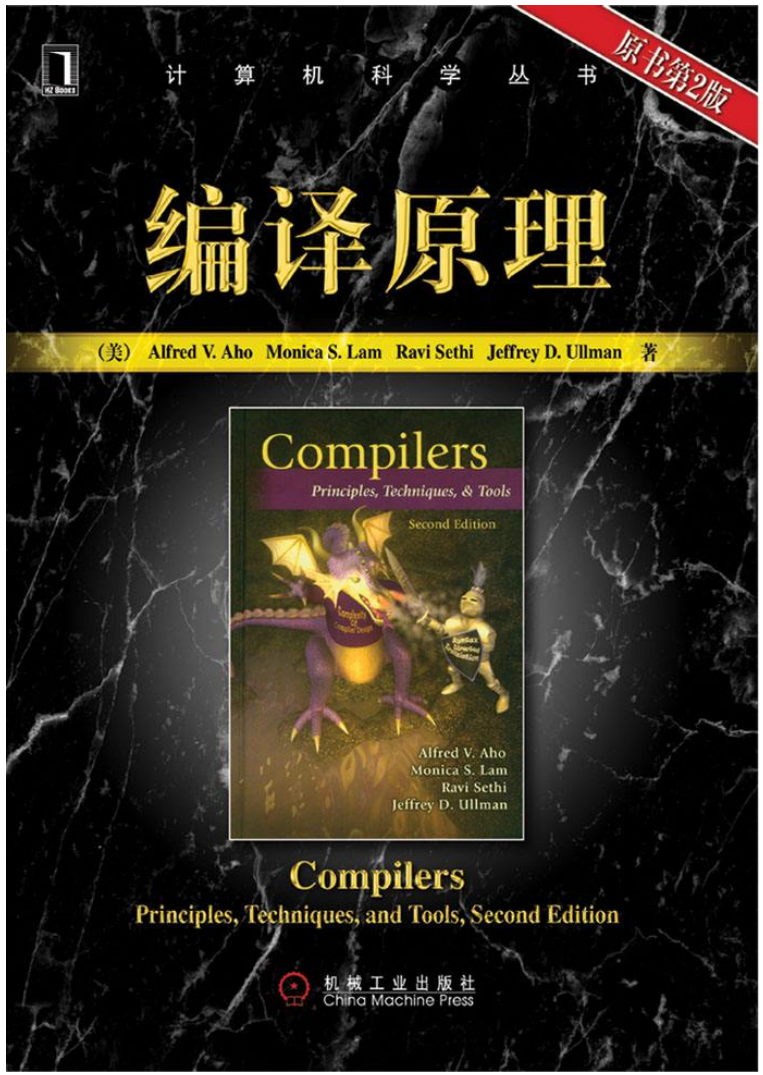


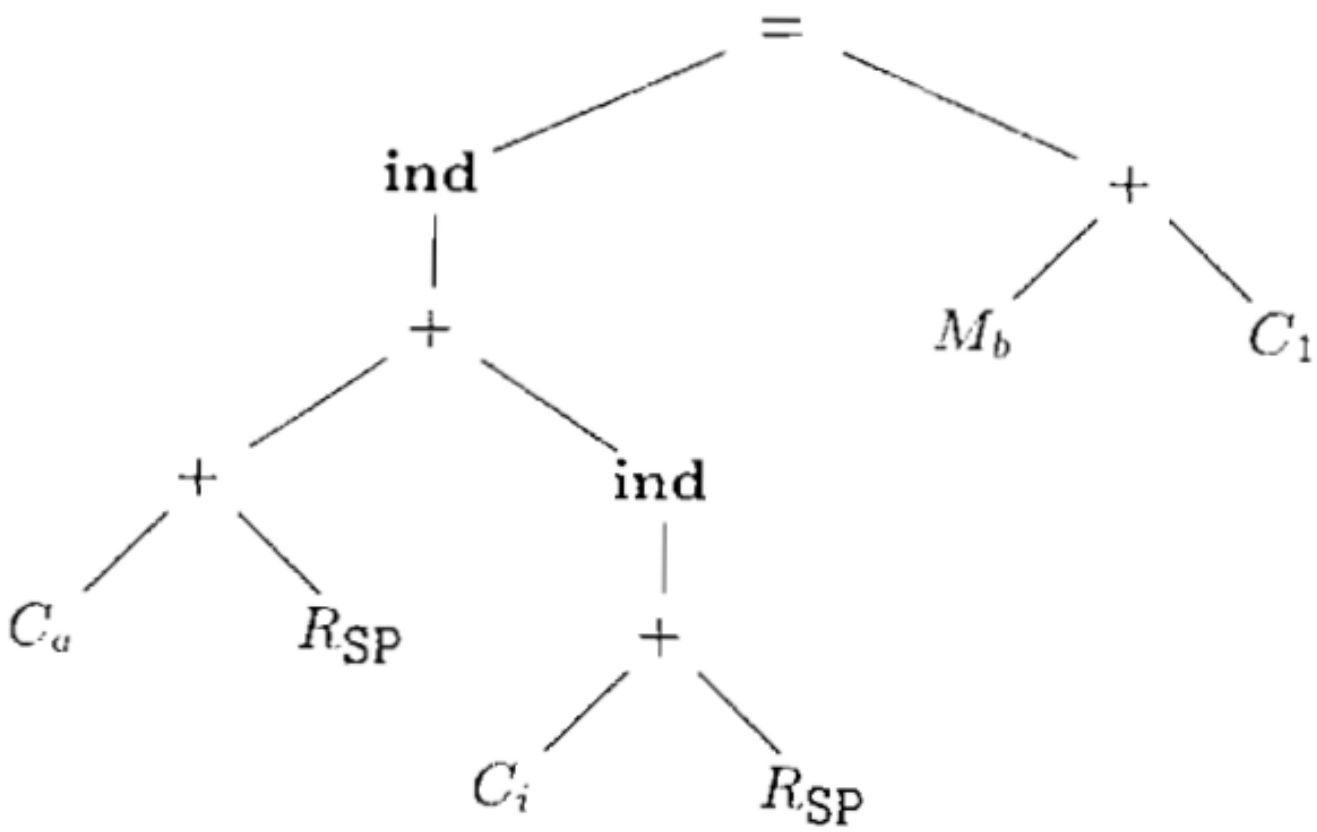
scheduler input for MUL:entry

```
bb.0.entry:
  liveins: $x0, $x2
  %2:gpr = COPY $x2
  %0:gpr = COPY $x0
  %3:gpr = MUL %0:gpr, %2:gpr
  $x0 = COPY %3:gpr
  PseudoRET implicit $x0
```

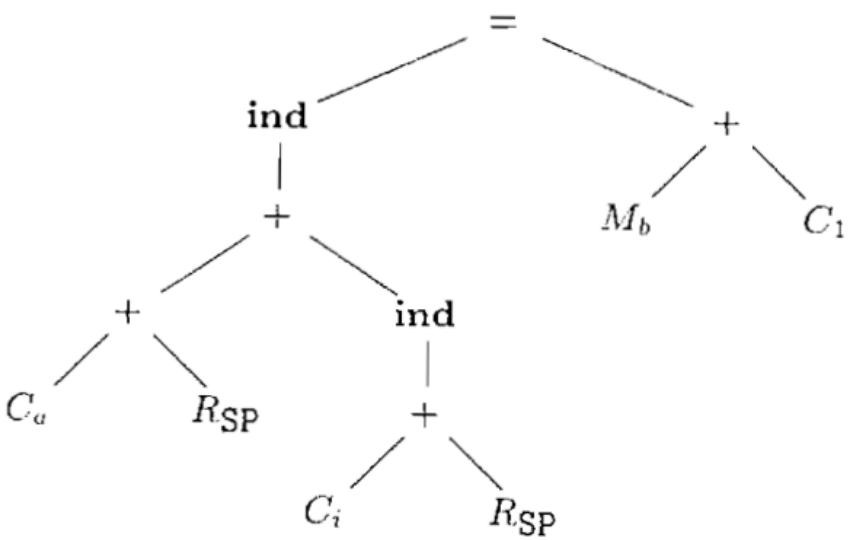
SelectionDAG Select Phase

The Select phase is the bulk of the target-specific code for instruction selection. This phase takes a legal SelectionDAG as input, pattern matches the instructions supported by the target to this DAG, and produces a new DAG of target code. For example, consider the following LLVM fragment:

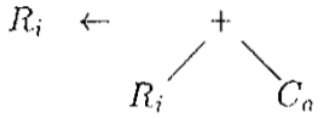

计 算 机 科 学 丛 书
原书第2版
编译原理
(美) Alfred V. Aho Monica S. Lam Ravi Sethi Jeffrey D. Ullman 著
Compilers
Principles, Techniques, & Tools
Second Edition
Alfred V. Aho
Monica S. Lam
Ravi Sethi
Jeffrey D. Ullman
Compilers
Principles, Techniques, and Tools, Second Edition
机械工业出版社
China Machine Press



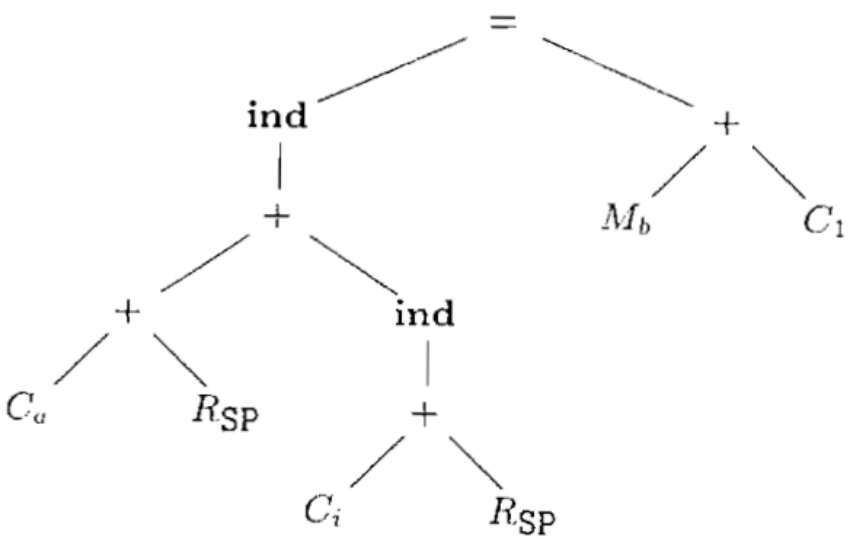
局部变量 a 与 i 的地址: 相对于 SP (栈指针) 的常数偏移量 $C_{\mathtt{a}}$ 和 $C_{\mathtt{i}}$




```
LD   R0, #a
ADD  R0, R0, SP
ADD  R0, R0, i(SP)
LD   R1, b
INC  R1
ST   *R0, R1
```




```
{ if (a = 1)
      INC Ri
  else
      ADD Ri, Ri, #a }
```



1)	R_i	$\rightarrow$ $\mathbf{c}_a$	{ LD Ri, #a }
2)	R_i	$\rightarrow$ M_x	{ LD Ri, x }
3)	M	$\rightarrow$ = M_x R_i	{ ST x, Ri }
4)	M	$\rightarrow$ = **ind** R_i R_j	{ ST *Ri, Rj }
5)	R_i	$\rightarrow$ **ind** + $\mathbf{c}_a$ R_j	{ LD Ri, a(Rj) }
6)	R_i	$\rightarrow$ + R_i **ind** + $\mathbf{c}_a$ R_j	{ ADD Ri, Ri, a(Rj) }
7)	R_i	$\rightarrow$ + R_i R_j	{ ADD Ri, Ri, Rj }
8)	R_i	$\rightarrow$ + R_i $\mathbf{c}_1$	{ INC Ri }
9)	R	$\rightarrow$ **sp**	
10)	M	$\rightarrow$ **m**	

= **ind** + + C_a R_{SP} **ind** + C_i R_{SP} + M_b C_1

= **ind** + + $\mathbf{c}_a$ **sp** **ind** + $\mathbf{c}_i$ **sp** + $\mathbf{m}_b$ $\mathbf{c}_1$

前缀表示

二义性：**偏向于执行较大的归约，而不是较小的归约**

归约/归约冲突　　移入/归约冲突

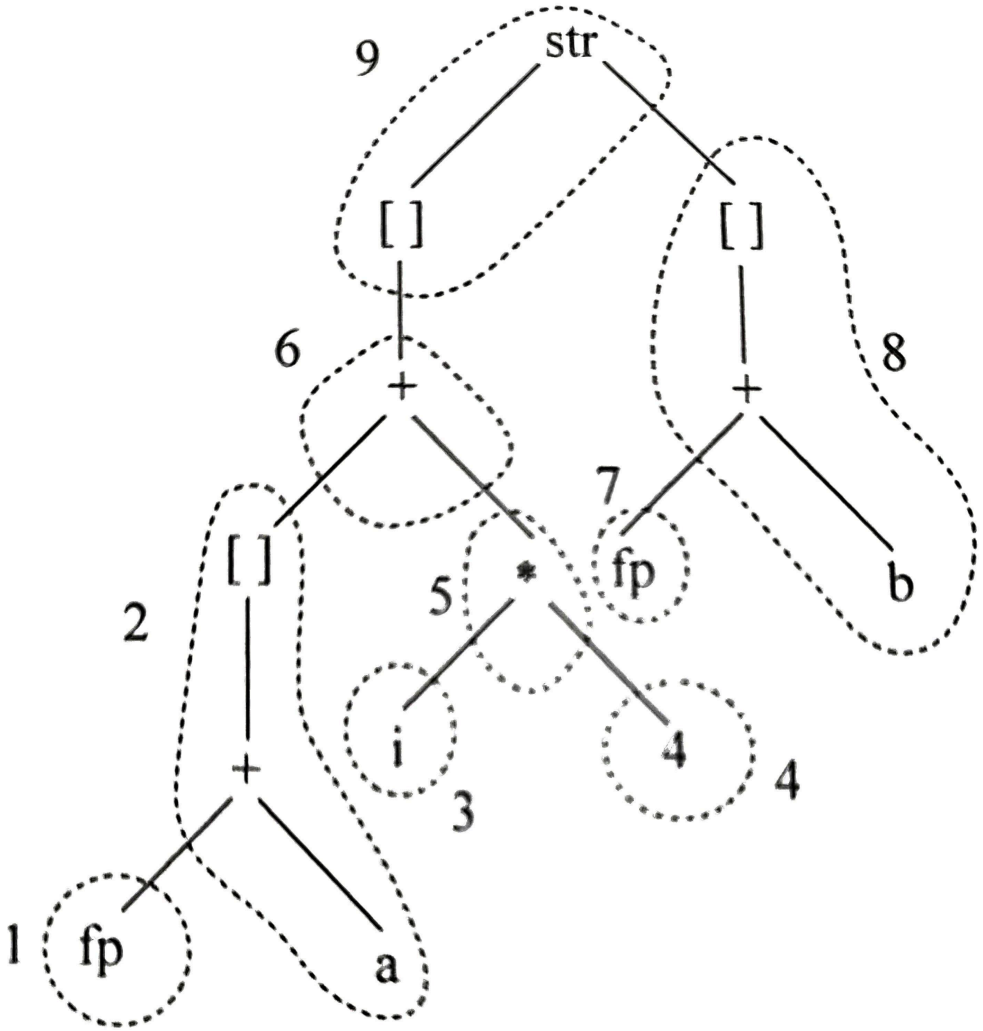
优先选择较长的归约　　优先选择移入动作


HUAWEI
华为智能计算技术丛书
华为ICT产品与解决方案总裁 汪 涛 | 中国工程院院士 郑纬民
联袂作序
Principle and Practice of the BiSheng Compiler
毕昇编译器
原理与实践
华保健 高耀清 ◎ 编著
Hua Baojian Gao Yaoqing
BiSheng
深入解析毕昇编译器原理与设计实践
清華大學出版社

算术与存储指令及树型 (鲲鹏处理器)

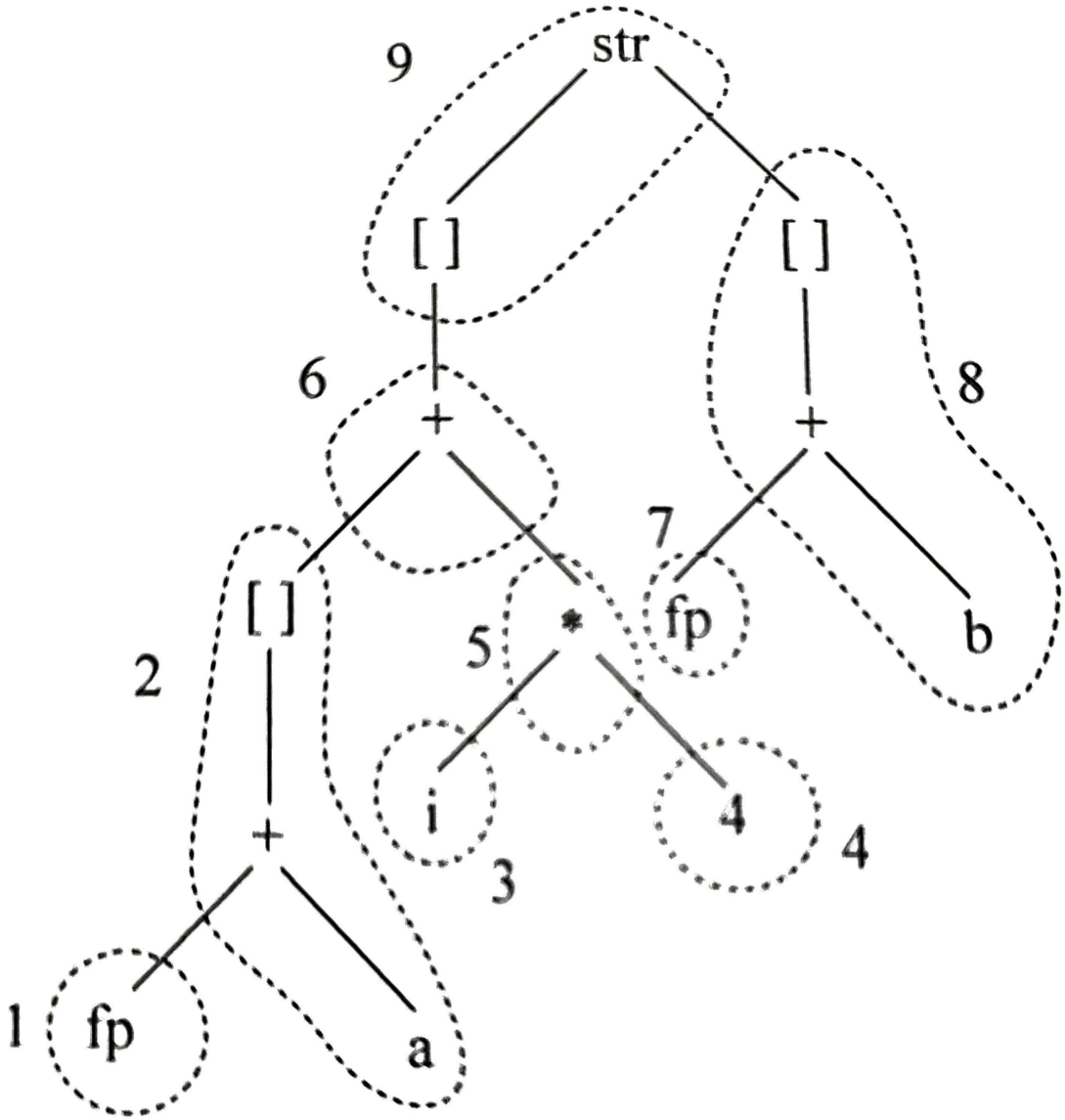
指　令	树　型
r_i	x
add r_i, r_j, r_k	+
mul r_i, r_j, r_k	*
sub r_i, r_j, r_k	−
div r_i, r_j, r_k	/
addi r_i, r_j, c	+ c　+ c c
subi r_i, r_j, c	− c　− c
ldr r_i, [r_j, c]	[] + c　[] + c　[] c　[]
str [r_j, c], r_i	str [] + c　str [] + c　str [] c　str []

瓦片覆盖 (tiling)




```
a[i] = b
```

```
ldr r1, [fp, a]
addi r2, r0, 4
mul r2, ri, r2
add r1, r1, r2
ldr r2, [fp, b]
str  [r1, 0], r2
```

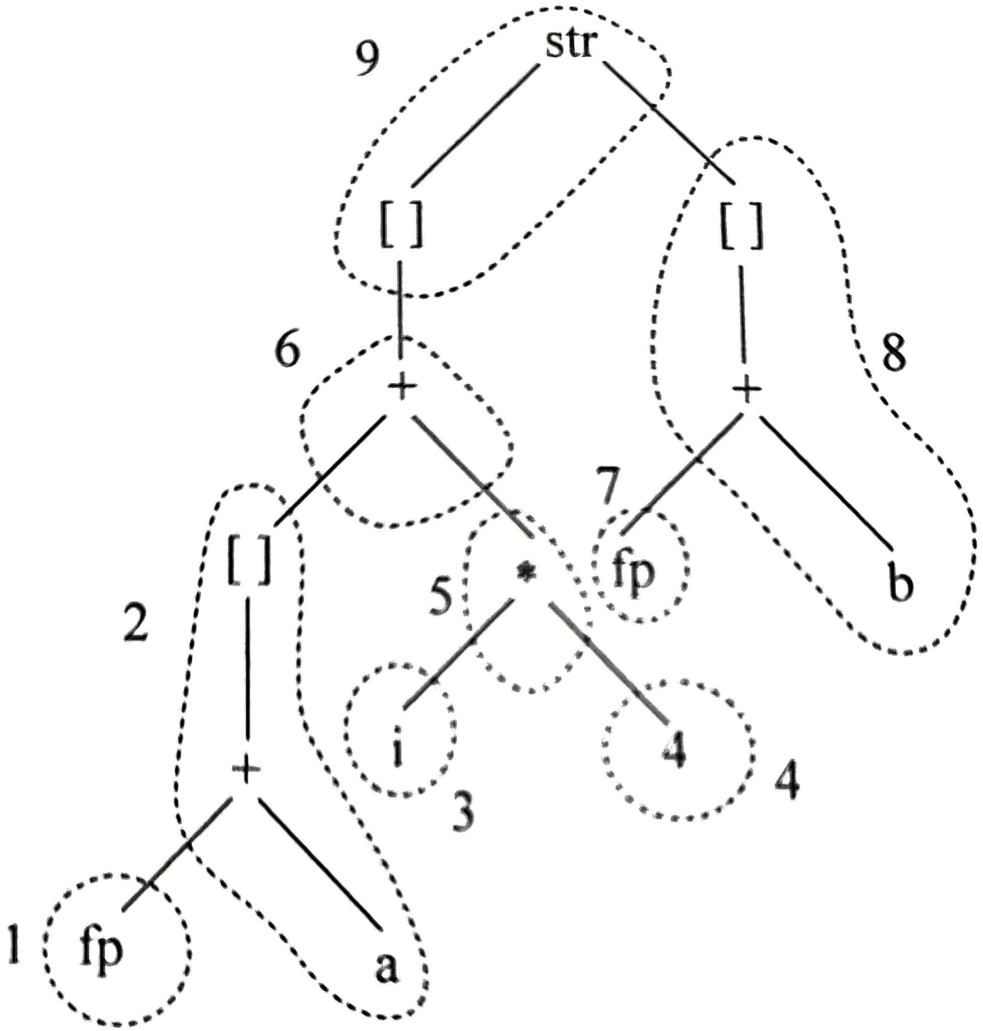



```
addi r1, r0, a
add  r1, fp, r1
ldr  r1, [r1, 0]
addi r2, r0, 4
mul  r2, r1, r2
add  r1, r1, r2
addi r2, r0, b
add  r2, fp, r2
ldr  r2, [r2, 0]
str  [r1, 0], r2
```

小树型覆盖方案

a[i] = b

瓦片覆盖 (tiling)




```
a[i] = b
```

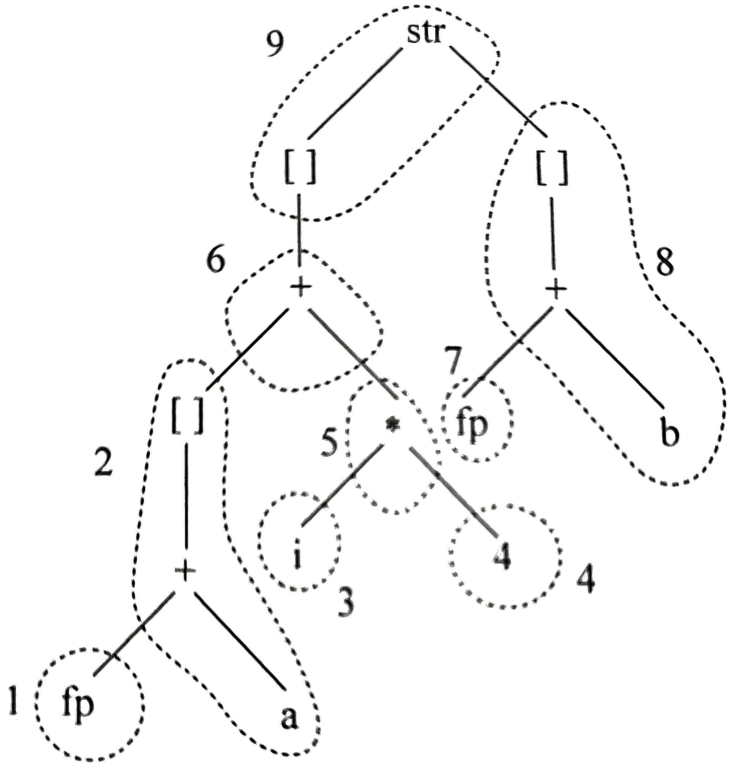
最大吞进算法 (maximal munch)

```
ldr r1, [fp, a]
addi r2, r0, 4
mul r2, ri, r2
add r1, r1, r2
ldr r2, [fp, b]
str  [r1, 0], r2
```

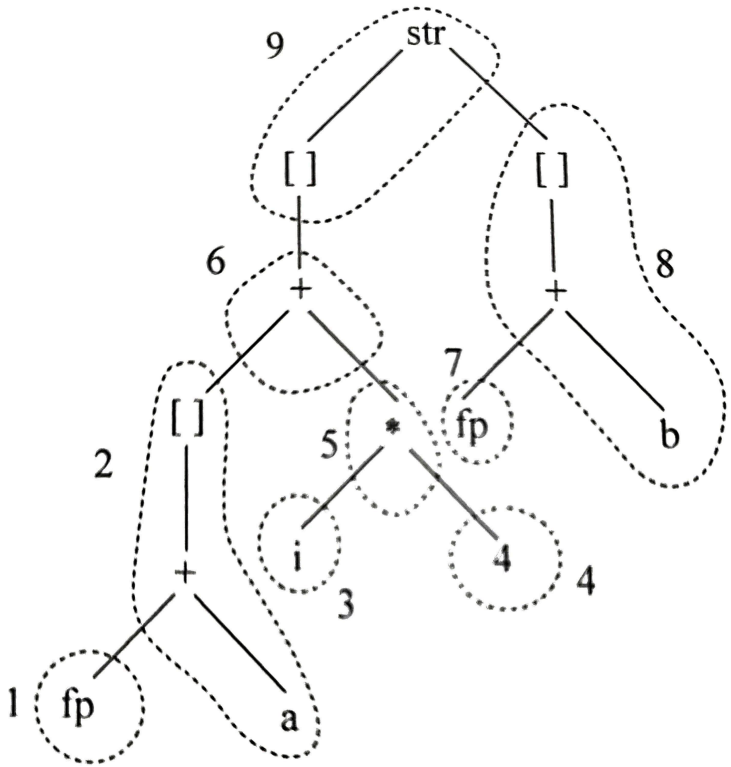
最佳覆盖方案

(不存在可以合并成更小代价树型的相邻树型)

使用**动态规划**思想计算最优覆盖方案

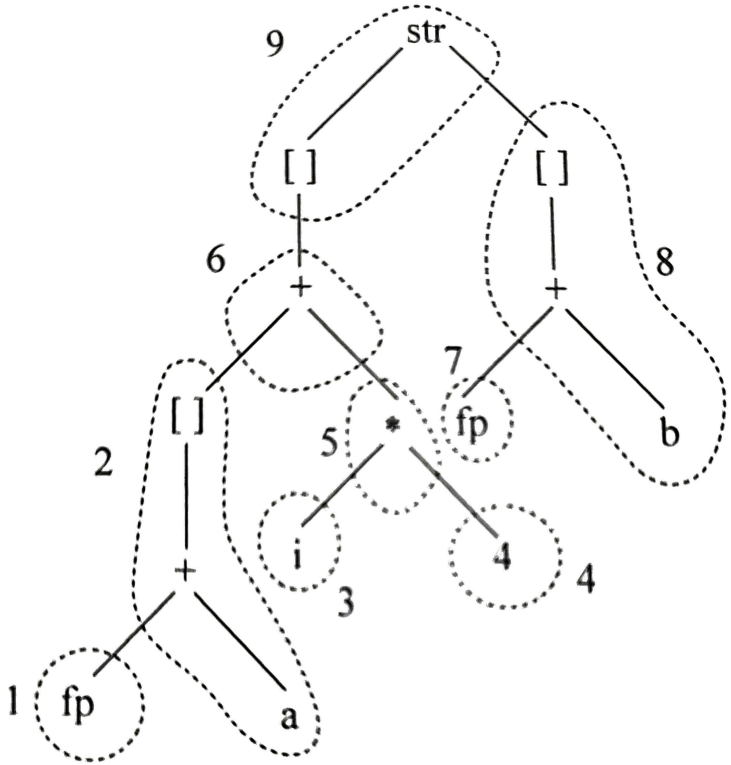


使用**动态规划**思想计算最优覆盖方案



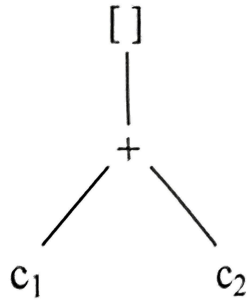
$$\forall n \in Node.\ c(n) = \min_{t \in tiles(n)} \left(c(t) + \sum_{n_i \in descendants(n)} c(n_i) \right)$$

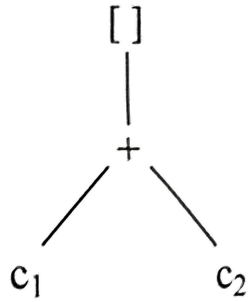
使用**动态规划**思想计算最优覆盖方案

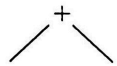
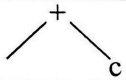
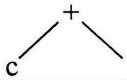


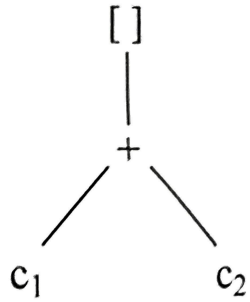
$$\forall n \in Node.\ c(n) = \min_{t \in tiles(n)} \left(c(t) + \sum_{n_i \in descendants(n)} c(n_i) \right)$$

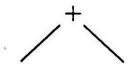
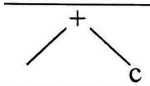
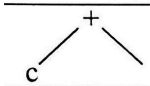
$$\texttt{Opt} = c(r) \quad (r \text{ is the root})$$

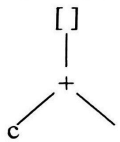
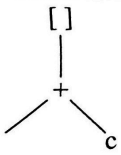

[ ]
+
c_1
c_2

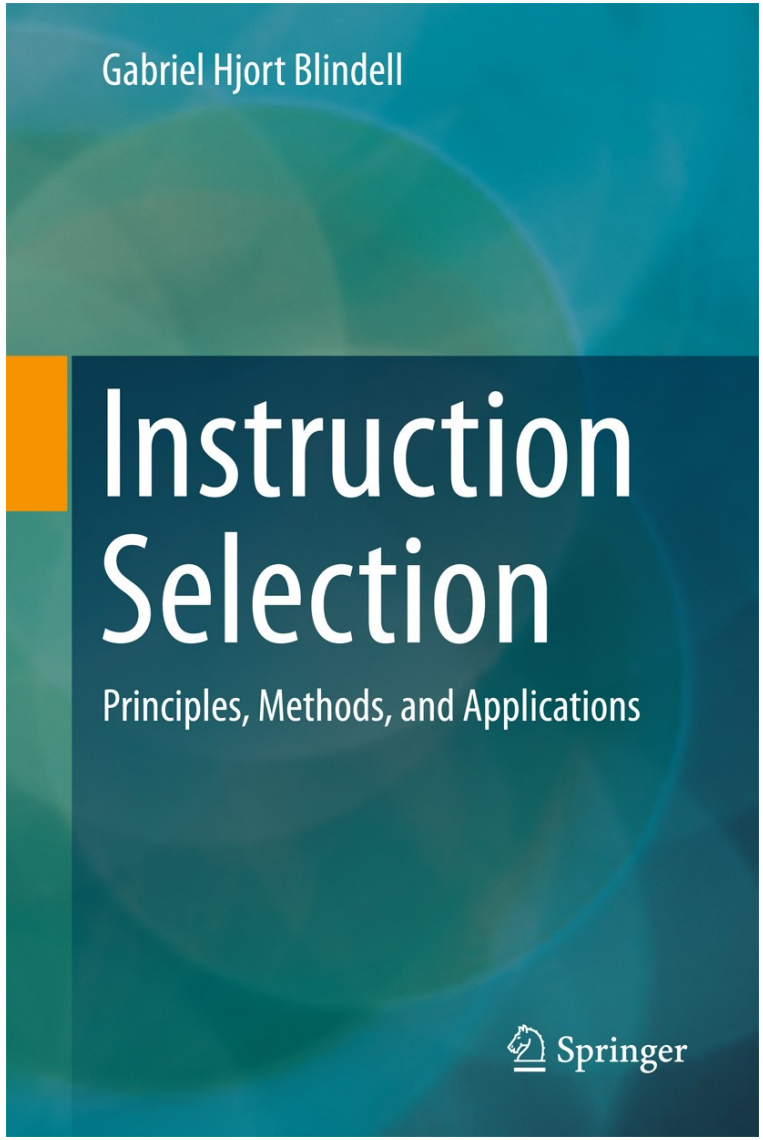


瓦片	指令	瓦片代价	叶子节点代价	总代价
	add	1	1+1	3
	addi	1	1	2
	addi	1	1	2

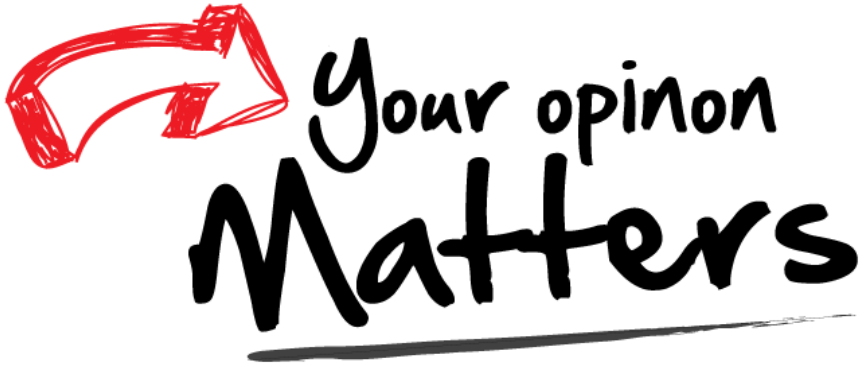


瓦片	指令	瓦片代价	叶子节点代价	总代价
	add	1	1+1	3
	addi	1	1	2
	addi	1	1	2

瓦片	指令	瓦片代价	叶子节点代价	总代价
	ldr	1	2	3
	ldr	1	1	2
	ldr	1	1	2


Gabriel Hjort Blindell
Instruction Selection
Principles, Methods, and Applications
Springer


Thank
You!



Office 926

hfwei@nju.edu.cn