Code Generation

Mini-C Compiler

CS4031 - Compiler Construction | *Fall 2025*

Team Members

Subhan (22K-4316)

Sadiq (22K-4303)

Muhammad Ahmed Haque (22K-4232)

Muhammad Irtiza (22K-4638)

Umer (22K-4160)

Junaid (22K-4369)

December 2025

1. Overview

Code generation is the final phase of the Mini-C compiler. It transforms TACKY IR into x86-64 assembly following the System V AMD64 ABI calling convention.

Location: src/backend/codegen/

2. Target Architecture: x86-64

2.1 Registers

Register	Purpose	Callee-Saved
%rax	Return value, accumulator	No
%rbx	General purpose	Yes
%rcx	4th argument	No
%rdx	3rd argument	No
%rsi	2nd argument	No
%rdi	1st argument	No
%rbp	Base pointer	Yes
%rsp	Stack pointer	Yes
%r8	5th argument	No
%r9	6th argument	No
%r10-%r11	Temporary	No
%r12-%r15	General purpose	Yes

2.2 Stack Frame Layout

```
High Address
+-----------------------------+
| Argument 8 (if needed)      |  +24(%rbp)
+-----------------------------+
| Argument 7 (if needed)      |  +16(%rbp)
+-----------------------------+
| Return Address              |  +8(%rbp)
+-----------------------------+
| Saved %rbp                  |  (%rbp)      <- Frame pointer
+-----------------------------+
| Local Variable 1            |  -8(%rbp)
+-----------------------------+
| Local Variable 2            |  -16(%rbp)
+-----------------------------+
| Saved Callee Registers      |
+-----------------------------+
| Temporary Storage           |
+-----------------------------+
| (Stack must be 16-aligned)  |  (%rsp)      <- Stack pointer
+-----------------------------+
Low Address
```

3. System V AMD64 ABI Calling Convention

3.1 Argument Passing

Integer/Pointer Arguments (first 6):

```
%rdi, %rsi, %rdx, %rcx, %r8, %r9
```

Arguments 7+: Pushed on stack (right to left)

Floating-Point Arguments: %xmm0 - %xmm7

3.2 Return Values

Type	Register
Integer/Pointer	%rax
Floating-point	%xmm0
Struct (small)	%rax, %rdx
Struct (large)	Via pointer

4. Code Generation Pipeline

```
TACKY IR
    |
    v
+-------------------------+
| 1. Instruction Selection|  IR -> Assembly AST (with pseudo-registers)
+-------------------------+
    |
    v
+-------------------------+
| 2. Register Allocation  |  Map pseudo-registers to stack slots
+-------------------------+
    |
    v
+-------------------------+
| 3. Instruction Fixing   |  Handle x86-64 constraints
+-------------------------+
    |
    v
+-------------------------+
| 4. Code Emission        |  Generate AT&T assembly text
+-------------------------+
    |
    v
Assembly File (.s)
```

5. Instruction Selection

5.1 TACKY to x86-64 Mapping

TACKY Instruction	x86-64 Assembly
dst = src (Copy)	movl src, dst
dst = -src (Negate)	movl src, dst; negl dst
dst = ~src (Complement)	movl src, dst; notl dst
dst = a + b	movl a, dst; addl b, dst
dst = a - b	movl a, dst; subl b, dst
dst = a * b	movl a, %eax; imull b; movl %eax, dst
dst = a / b	movl a, %eax; cdq; idivl b; movl %eax, dst
dst = a % b	movl a, %eax; cdq; idivl b; movl %edx, dst

5.2 Comparison and Conditional

TACKY Instruction	x86-64 Assembly
dst = a < b	cmpl b, a; setl %al; movzbl %al, dst
dst = a <= b	cmpl b, a; setle %al; movzbl %al, dst
dst = a == b	cmpl b, a; sete %al; movzbl %al, dst
dst = a != b	cmpl b, a; setne %al; movzbl %al, dst
goto label	jmp label
if x == 0 goto L	cmpl $0, x; je L
return x	movl x, %eax; leave; ret

5.3 Function Calls

```
; call f(a, b, c)
movl a, %edi       ; 1st argument
movl b, %esi       ; 2nd argument
movl c, %edx       ; 3rd argument
call f
movl %eax, result  ; Get return value
```

6. Complete Code Generation Example

Source Code:

```
int factorial(int n) {
    if (n <= 1)
        return 1;
    return n * factorial(n - 1);
}
```

Generated Assembly (AT&T Syntax):

```
.globl factorial
    .text
factorial:
    pushq   %rbp
    movq    %rsp, %rbp
    subq    $16, %rsp           # Allocate stack space

    # Save parameter n
    movl    %edi, -4(%rbp)      # n at -4(%rbp)

    # if (n <= 1)
    cmpl    $1, -4(%rbp)
    jg      .L_else1            # Jump if n > 1

    # return 1
    movl    $1, %eax
    leave
    ret

.L_else1:
    # n - 1
    movl    -4(%rbp), %eax
    subl    $1, %eax

    # factorial(n - 1)
    movl    %eax, %edi          # Pass argument
    call    factorial
    movl    %eax, -8(%rbp)      # Save result

    # n * factorial(n - 1)
    movl    -4(%rbp), %eax
    imull   -8(%rbp), %eax

    # return
    leave
    ret
```

7. Assembly Directives

Directive	Purpose
.globl name	Export symbol
.text	Code section
.data	Initialized data section
.bss	Uninitialized data section
.align n	Align to n-byte boundary
.long value	4-byte integer
.quad value	8-byte integer
.string "..."	Null-terminated string

8. AT&T vs Intel Syntax

AT&T	Intel	Note
movl $5, %eax	mov eax, 5	Source, Dest order
%eax	eax	Register prefix
$5	5	Immediate prefix
(%rax)	[rax]	Memory reference
8(%rbp)	[rbp+8]	Displacement
movl	mov dword	Size suffix