Intermediate 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

Intermediate Code Generation is the fourth phase of the Mini-C compiler. It transforms the AST into TACKY IR (Three-Address Code), a low-level representation that is machine-independent but close to assembly.

Location: src/backend/ir/

1.1 Design Principles

- Three-Address Code: Each instruction has at most 3 operands
- Flat Structure: No nested expressions; complex expressions are broken down
- Explicit Control Flow: Jumps and labels instead of structured statements
- Typed Operations: Each operation knows its operand types
- SSA-like Temporaries: New temporaries for intermediate results

2. TACKY IR Instructions

2.1 Value Operations

Instruction	Format	Description
Copy	dst = src	Copy value
Unary	dst = op src	Unary operation
Binary	dst = left op right	Binary operation
Load	dst = *src	Load from pointer
Store	*dst = src	Store to pointer
GetAddress	dst = &src	Get address

2.2 Control Flow Instructions

Instruction	Format	Description
Jump	goto label	Unconditional jump
JumpIfZero	if src == 0 goto label	Conditional jump
JumpIfNotZero	if src != 0 goto label	Conditional jump
Label	label:	Jump target
Return	return src	Return from function
FunCall	dst = call func(args)	Function call

3. Translation Rules

3.1 Expression Translation

Binary Expression: a + b * c

```
tmp.1 = b * c
tmp.2 = a + tmp.1
result = tmp.2
```

Short-Circuit AND: a && b

```
tmp.1 = a
    if tmp.1 == 0 goto false_label
    tmp.2 = b
    if tmp.2 == 0 goto false_label
    tmp.3 = 1
    goto end_label
false_label:
    tmp.3 = 0
end_label:
    result = tmp.3
```

Ternary: a ? b : c

```
tmp.1 = a
    if tmp.1 == 0 goto else_label
    tmp.2 = b
    goto end_label
else_label:
    tmp.2 = c
end_label:
    result = tmp.2
```

3.2 Statement Translation

If-Else Statement:

```
tmp.1 = condition
    if tmp.1 == 0 goto else_label
    ; then_body instructions
    goto end_label
else_label:
    ; else_body instructions
end_label:
```

While Loop:

```
continue_label:
    tmp.1 = condition
    if tmp.1 == 0 goto break_label
    ; body instructions
```

```
    goto continue_label
break_label:
```

For Loop:

```
; init instructions
start_label:
    tmp.1 = condition
    if tmp.1 == 0 goto break_label
    ; body instructions
continue_label:
    ; post instructions
    goto start_label
break_label:
```

4. Complete Translation Example

Source Code:

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

TACKY IR Output:

```
function factorial(n):
    ; if (n <= 1)
    tmp.1 = n <= 1
    if tmp.1 == 0 goto else.1

    ; return 1
    return 1

else.1:
    ; n - 1
    tmp.2 = n - 1

    ; factorial(n - 1)
    tmp.3 = call factorial(tmp.2)

    ; n * factorial(n - 1)
    tmp.4 = n * tmp.3

    ; return result
    return tmp.4
```

Array Sum Example:

```
int sum_array(int *arr, int size) {
    int total = 0;
    for (int i = 0; i < size; i = i + 1) {
        total = total + arr[i];
    }
    return total;
}
```

TACKY IR Output:

```
function sum_array(arr, size):
    ; int total = 0
    total = 0
    ; int i = 0
    i = 0

for.start.1:
    ; i < size
    tmp.1 = i < size
    if tmp.1 == 0 goto for.break.1

    ; arr[i] - compute address
    tmp.2 = i * 4           ; sizeof(int) = 4
    tmp.3 = arr + tmp.2     ; pointer arithmetic
    tmp.4 = *tmp.3          ; load arr[i]

    ; total = total + arr[i]
    tmp.5 = total + tmp.4
    total = tmp.5

for.continue.1:
    ; i = i + 1
    tmp.6 = i + 1
    i = tmp.6
    goto for.start.1

for.break.1:
    ; return total
    return total
```

5. IR Properties

Property	Description
Machine Independence	No register references, no specific instruction set
Type Preservation	Operations maintain type information
Explicit Addressing	All memory accesses made explicit

Linear Representation	Flat sequence of instructions
SSA-like	Temporaries typically assigned once