Bab 7

Three-Address Code Generation

Sub-CPMK yang Dicakup dalam Bab Ini:

- **Sub-CPMK 4.1:** Merancang three-address code representation

7.1 Pengenalan Three-Address Code (TAC)

Three-Address Code (TAC) adalah bentuk Intermediate Representation (IR) yang setiap instruksinya memiliki maksimal tiga operandi (biasanya satu hasil dan dua operan).

7.1.1 Format Instruksi TAC

Format umum instruksi TAC adalah: `x = y op z`.

- `x`: Tujuan (biasanya variabel *temporary*).
- `y`, `z`: Operan (variabel, konstanta, atau temporary).
- `op`: Operator (aritmatika, logika, atau relasional).

7.1.2 Temporary Variables

Kompilator secara otomatis membangkitkan variabel sementara (*temporaries*) untuk menyimpan hasil antara dari ekspresi kompleks. Misal, `a + b * c` ditransformasikan menjadi:

```
t1 = b * c
t2 = a + t1
```

7.2 Representasi: Quadruples dan Triples

Ada beberapa cara menyimpan instruksi TAC di dalam memori:

7.2.1 Quadruples

Setiap instruksi disimpan dalam struktur dengan empat *fields*: `(op, arg1, arg2, result)`. Keunggulannya adalah kemudahan dalam optimasi karena setiap instruksi memiliki lokasi eksplisit.

ID	op	arg1	arg2	result
(0)	*	b	c	t1
(1)	+	a	t1	t2

Gambar 7.1: Representasi Quadruples untuk `a + b * c`

7.2.2 Triples

Representasi yang lebih efisien memori dengan mengacu pada ID instruksi sebelumnya sebagai operan: `(op, arg1, arg2)`. Hasil antara direferensikan melalui index array instruksi tersebut.

7.3 Translasi Ekspresi dan Penugasan

7.3.1 Skema Translasi AST ke TAC

Proses ini biasanya dilakukan melalui penelusuran AST (*recursive traversal*). Fungsi translasi untuk ekspresi biner mengembalikan variabel *temporary* yang menampung hasilnya.

```
string translateExpr(ASTNode* node) {
    if (node->isLiteral()) return node->value;
    if (node->isBinary()) {
        string t1 = translateExpr(node->left);
        string t2 = translateExpr(node->right);
        string temp = generateTemp();
        emit(node->op, t1, t2, temp);
        return temp;
    }
}
```

7.3.2 Manajemen Label

Untuk struktur kontrol, kita memerlukan label unik agar parser dapat melakukan *jump* (lompatan) antar blok kode.

7.4 Struktur Kontrol: Label dan Jump

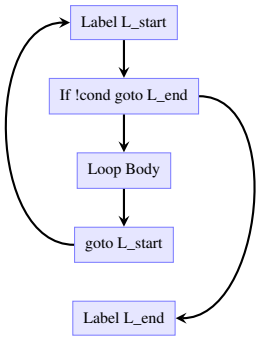
7.4.1 Translasi If-Then-Else

Struktur kondisional diterjemahkan menggunakan instruksi lompatan bersyarat (*conditional jump*) dan tanpa syarat (*unconditional jump*).

```
    if (condition) goto L1
    // Code for False branch
    goto L2
L1: // Code for True branch
L2: // Next statement
```

7.4.2 Translasi While-Loop

Loop memerlukan label di awal untuk pengulangan dan label di akhir untuk terminasi berdasarkan evaluasi kondisi.



Gambar 7.2: Struktur aliran TAC untuk statement While

7.5 Praktikum: TAC Generator dari AST

Mahasiswa akan mengimplementasikan fungsi `genTAC()` pada setiap kelas node AST untuk menghasilkan instruksi IR secara otomatis.

7.5.1 Pengelolaan Temp Table

Penting untuk memastikan nama variabel *temporary* (`t1, t2, ...`) unik di seluruh program dan didaftarkan ke *temporary table* untuk manajemen alokasi memori nantinya.

```
void genTAC(ASTNode* node) {
    if (node->type == ASSIGN) {
        string src = translateExpr(node->right);
        emit("STORE", src, "-", node->left->name);
    }
}
```

Aktivitas Pembelajaran

1. **Expression Translation**: Konversi berbagai ekspresi kompleks ke three-address code.
2. **Control Flow**: Implementasikan translator untuk if-then-else dan while loops.
3. **Function Translation**: Bangun translator untuk function calls dan parameter passing.
4. **Array Handling**: Implementasikan array access dan assignment dalam three-address code.
5. **TAC Generator**: Buat generator lengkap dari AST ke three-address code.

Latihan dan Refleksi

1. Konversi ekspresi `a * (b + c) - d / e` ke three-address code!
2. Terjemahkan statement `for(i=0; i<10; i++) x = x + i;` ke three-address code!
3. Implementasikan temporary variable management dengan optimalisasi!
4. Analisis keuntungan three-address code untuk optimasi!
5. Bandingkan three-address code dengan SSA form!
6. **Refleksi**: Bagaimana three-address code menyederhanakan code generation?

Asesmen (Evaluasi Kinerja)

Instrumen Penilaian untuk Sub-CPMK 4.1

A. Pilihan Ganda

1. Three-address code memiliki maksimal:
   (a) 2 operands
   (b) 3 operands
   (c) 4 operands

(d) Tidak terbatas

2. Temporary variables digunakan untuk:

   (a) Menyimpan hasil sementara

   (b) Optimasi kode

   (c) Error handling

   (d) Debugging

3. Keuntungan utama three-address code adalah:

   (a) Eksekusi lebih cepat

   (b) Ukuran kode lebih kecil

   (c) Memudahkan optimasi

   (d) Debugging lebih mudah

B. Essay

1. Jelaskan proses konversi dari AST ke three-address code dengan contoh kompleks!

2. Desain dan implementasikan three-address code generator untuk bahasa dengan arrays dan function calls!

Rubrik Penilaian: Lihat Lampiran A

Checklist Pencapaian Kompetensi

Centang item berikut setelah Anda yakin telah menguasainya:

☐ Saya dapat merancang three-address code representation

☐ Saya dapat mengkonversi ekspresi aritmatika ke three-address code

☐ Saya dapat menerjemahkan control flow structures

☐ Saya dapat mengimplementasikan temporary variable management

☐ Saya dapat menangani arrays dan pointers

☐ Saya dapat mengimplementasikan function calls

Rangkuman

Bab ini membahas three-address code generation, termasuk format instruksi, translation dari AST, implementasi data structures, dan penanganan konstruksi kompleks. Mahasiswa belajar membangun intermediate code generator.

Poin Kunci:

- Three-address code adalah IR dengan maksimal 3 operands per instruksi
- Translation dari AST ke TAC memerlukan temporary variables
- Control flow structures memerlukan label dan jump instructions
- TAC memudahkan optimasi dan code generation
- Arrays dan pointers memerlukan address arithmetic

Kata Kunci: ***Three-Address Code***, ***Intermediate Representation***, ***TAC***, ***Temporary Variables***, ***AST Translation***, ***Control Flow***