COMPILER

3.1. Aspek Kompilasi

Bertahun tahun pemrograman bahasa tingkat tinggi (high level languange-HLL) telah dikenal luar di lingkungan pemrograman komputer karena kemudahan dalam pembuatannya. Beberapa tahun terakhir ada dua faktor yang menjadi perhatian dalam pengembangan program, yaitu : protabilitas dan pemeliharaan program. Seperti diketahui komputer tidak 'mengenal' HLL, sehingga diperlukan suatu konversi ke dalam bahasa mesin.

Compiler adalah suatu program yang melakukan proses translasi dari HLL ke dalam bahasa mesin di komputer. Disamping program translasi, compiler juga mempunyai beberapa fungsi penting, seperti diagnostik, contohnya kemampuan pendeteksian error/kesalahan. Pelannggaran spesifikasi HLL akan terdeteksi dan dilaporkan kepada programmer oleh compiler agar seger diperbaiki hingga mempermudah pembentukan machine language equivalent.

Dari penjelasan di atas terlihat bahwa tugas dari compiler adalah :

(i) melakukan translasi dari HLL program sebagai input (source program) ke dalam equivalent machine language program.

(ii) Menghasilkan pesan hasil diagnostik kepada programmer bila terjadi penyimpangan spesifikasi HLL(source language).

3.1.1. Assembler dan Compiler

Assembler adalah translator untuk bahasa tingkat rendah (low level language –LLL), sedangkan compiler adalah translator untuk HLL. Suatu bahasa assembly memiliki ciri khas dari komputer yang digunakannya, berbeda dengan HLL yang tidak tergantung dari jenis mesin (machine independent). Perbedaan lain, bahasa assembly menurunkan sifat arsitektur komputer yang digunakannya, sedangkan HLL membuat feature arsitekturnya sendiri. Feature arsitektur dari HLL dapat dilihat dari beberapa komponen yang dibutuhkan saat pemrograman dilakukan, yaitu :

- Tipe Data

  Pada kebanyakan bahasa assembly, programmer diijinkan untuk menggunakan dan memanipulasi data yang memiliki tipe data berbeda dalam program. Programmer harus memilih dan menggunakan insruksi mesin secara tepat untuk memanipulasi data dengan perbedaan tipe data tadi. Hal ini menyulitkan programmer, karena tidak ada pengecekan dari assembler untuk memverifikasi apakah instruksi yang digunakan tersebut compatibled engan tipe data.

  Pada HLL terdapat pelindung terhadap integritas tipe data. Kompatibilitas operator dan operand akan dicek oleh compiler dan bila perlu dilakukan pemaksaan operasi. Karenanya bisa saja nanti ditemukan pesan 'no invalid operator' pada data. Untuk itu compiler harusmemiliki kemampuan untuk memilih urutan yang tepat dari instruksi mesin dalamhal penggunaan tipe data yang berbeda.

  Contoh :
  ```
  x, y : real;
  I, j : integer ;
  y := 10 ;
  x := y + i
  ```
  Dari contoh di atas terlihat bahwa ketika melakukan eksekusi statement pertama, compiler harus menyadari bahwa y adalah real, karenanya perlu dilakukan konversi '10' dari representasi integer menjadi floating point. Pada eksekusi statement kedua dilakukanoperasi penambahan yang sebenarnya tidak dapat dilakukan karena y adalah variabel real dan i variable integer. Operasi tersebut bisa dilakukan apabila i di konversi ke dalam real.

- Struktur Data

  HLL memperbolehkan programmer untuk menyatakan dan menggunakan struktur data seperti array, stack, table, record, list, dan sebagainya. Programmer dapat mengakses bagian dari struktur data tersebut, sebagai contoh kita membutuhkan elemen ke 10 dari array satu dimensi A, atau elemen yang ada ada baris ke-2 dan kolom ke-6 dari array dua dimensi B. Record dan tbale dapat berisi data yang heterogen dan membutuhkan storage mapping yang cukup kompleks. User akan mendefinisikan tipe permasalahannya dalam storage allocation untuk kemudian dilakukan pengaksesan

  Contoh :

```
Type
   Employee = record;
            Name : array [1..10] of character;
            Sex : character;
            Id : integer;
   End;
   Weekday = (Monday, Tuesday,Wednesday, Thursday,Friday);
Var
   Info : array [1..500] of employee;
   Today : weekday;
   i, j : integer;
Begin
   Today := Monday;
   Info[i].id :=j;
   If  today = Monday then ….
End.
```

  Pada contoh di ats, info adalah suatu array record. Referensi info[i].id menunjukkan dua jenis mapping. Pertama adalah mapping yang omogen didasarkan pada suatu array. Mapping yang kedua, compiler menyediakan akses fiels id dari sekumpulan field yang heterogen. Weekdaya adalah tipe data yang didefinisikan oleh programmer. Secara keseluruhan, terdapat perbedaan mapping dimana compiler harus memperhitungkan bagaiman merepresentasikan perbedaan nilai yang didefinisikan oleh programmer dan membuat suatu mapping yang benar.

- Lingkup aturan (scope rules)

  DalamHLL, lingkup aturan penamaan data berkaitan dengan blok struktur dalam bahasa, seperti FORTRAN, lingkup aturan data adalah program/subprogram yang berisi variabel yang telah di-declare. Dalam Pascal, linkupnya dibatasi pada program block dari data yang di-declare. Beberapa HLL yang lain mengijinkan programmer untuk menggunakan dynamic control dalam lingkup data itemnya.

  Interaksi antara berbagai space nama yang diijinkan, disediakan oleh compiler. Ada dua aspek yang diperhatikan (i) kesesuaian dalamm enetapkan waktu kompilasi (ii) kesesuaian untuk menetapkan eksekusi, contohnya procedure call. Dalam procedure call, target bahasa programyang digenerate oleh compiler menjadi syarata untuk mengimplementasikan aturan HLL, seperti value, reference/name dan sebagainya.

  Contoh :

```
Procedur abc(m,n);
Var
z, w : real;
   Procedure P (g : real);
   Var
   z, s : real;
   Begin
            Q(z,g);
   End;
   Procedure Q(a, b : real);
   Var
   c, d : real;
   Begin
            C:=w;
            D := a;
   End;
P(w);
End.
```

Akses ke variable non lokal ditentukan oleh lingkup aturan. Pada statement c:= z, procedure Q, variable z tidak dapat dideklarasikan seperti z pada procedure P. Hal ini dikarenakan visibilias variable dibatasi pada deklarasi di P, hanya untuk P saja. Karena itu z disini adalah z yang dideklarasikan di prosedur Utama. Variabel z di P diakses Q ketika procedur call Q(z,g) dieksekusi oleh P. Alias diantara actual parameter z pada prosedur P dan formal parameter dari procedur Q ditetapkan ketika call dieksekusi. Pada akhirnya gabungan itu berada di akhir, selama call Q(z,g)

- Struktur Kendali
  Struktur kendali dalam suatu bahasa adalah kumpulan fasilitas bahasa untuk engurutkan suatu kendali dlamsebuah progarm. Urutan feature tersebut seperti statement transfer kendali : if….. then…. else, iterasi : do dan do while, serta procedure call. Kontrol kendali sangat penting dalam HLL dalam memecahkan masalah.

Pada bab ini, pembahasan difokuskan pada prinsip dasar dalam perancangan compiler. Seperti diketahui perancangan sistem program sangat rentan akan pengaruh lingkungan, seperti konfigurasi mesin yang mengoperasikannya, pengembangan progarm, serta strategi kompilasi. Diharapkan dengan memahami perancangan compiler ini, kita dapatmengetahui bagaimana kegiatan suatu kompilasi, pembangkitan kode yang efisien dan dukungan diagnostik yang baik.

3.2. Pengenalan Proses Kompilasi

Sama halnya dengan bahasa assembly, proses kompilasi mengalami 2 fase, yaitu :

{Analysis of Source Text} + {Synthesis of Target Text} = {Translation from Source Text to Target}

Source text analysis didasarkan pada tata bahasa (grammar) dalam source languange. Komponen dari fase analysis adalah (i) syntax analysis yang menentukan struktur sintaks dari source statement (ii) semantic analysis yang menentukan arti statement secara tata bahasa.
Contoh :

```
Integer i ;
Real a, b;
a := b + i ;
```

Syntax analysis dari statement pada baris ketiga adalah variabel a ditempatkan di sisi kiri dan variable b dan i ditempatkan di sisi kanan yang dihubungkan denga operator ‘+’. Aturan secara semantik mengatakan bahwa b dan i dijumlahkan dan hasilnya disimpan di a. Di atas terlihat, bahwa b adalah variable real dan i adalah integer. Semantic analysis akan mengatakan bahwa keduanya tidak dapat dijumlahkan secara langsung. Dalam aturan semantic ada 3 tahap prosedur yang harus dilakukan:

(i) mengkonversi nilai i ke dalam mode representasi ‘real’
(ii) menambahkannya dengan nilai b
(iii) menyimpan hasilnya pada variabel yang ada di sisi kiri, yaitu a.

Selama proses sintesis target text, compiler akan menentukan instruksi danmode pengalamatan yang digunakan dalam target program berdasarkan ketiga tahapan di atas. Aspek kompilasi disini adalah keterhubugan dengan target machine yang dikenal dengan pragmatic compilation.

3.2.1. Fase Analysis

Faser analysis dalam compiler terbagai menjadi beberapa tahap, yaitu :

(i) lexical analysis
(ii) syntax analysis
(iii) semantic analysis

Seperti telah dijelaskan sebelumnya, syntax analysis menentukant ata bahasa atau struktur sintaks dari input statement dan merepresentasikannya pada intermediate forma yang ditunjukkan oleh semantic analysis. Selama syntax analysis, individual lexical unit dikenal sebagai identifier/konstanta dan sebagainya. Hal yang penting dalam sintaks adalah pemahaman kategori sintaks yang dibentuk dari lexical unit. Contoh a:= b + i , dalam syntax analysis cukup ditulis <id> = <id>+<id>, dimana <id> merupakan identifier (dalam hal ini variable). Lexical analysis adalah tahap pengenalan sebelum masuk ke syntax analysis untuk menyaring informasi dari syntax analyzer.

- Lexical Analysis
  Lexical analysis menjalankan micro level examination dari input text untuk mengidentifikasikan lexical unit yang ada di dalamnya dan menentukan kategori sintaks. Lexical unit merupakan kalimat yang diisolasi untuk mencari suatu set spesial symbol yang dikenal dengan delimiters. Prosesnya disebut SCANNING. Pada kasus natural language, simbol tanda baca dan blank dipisahkan dari kata dalam kalimat sebgai suatu delimiters. Dalam bahasa pemrograman, delimiters umumnya adalah operator dan kata kunci dari bahasa. Setealh pengisolasian unsur pokok lexical, dilakukan pengklasifikasian kategori sintaks dan pengecekan terhadap validitas dari aturan kategori tersebut. hal ini dilakukan untuk mendteksi dan mengindikasikan lexical error, seperti invalid variable name, invalid constant dan sebagainya. Suatu desriptor (disebut juga token) dibangun dalam lexical unit. Desriptor berisi kategori sintaks lexical unit dan output yang berbeda di antara kategori sintaks. Sebagai contoh a := b + i. Syntax analysis ahnya membutuhkan 3 identifier yang dipisahkan oleh operator. Dari contoh di atas operator yang digunakan adalah ‘=’ dan ‘+’. Semantic analysis nantinya akan menyimpulkan variabel di sisi kiri adalah hasil ekspresi di sisi kanan. Ketika berada pada tahap target code generation, diperlukan identifier yang digunakan pada statemetn tersebut. Contoh I yang ditambahkan dengan b hasilnya disimpan di a. Descriptor memerlukan informasi tentang actual identifier (a, b dan i) yang akan dimasukkan ke dalam tabel (symbol tabel). Desriptor untuk identifier terdiri dari ordinal number. Untuk contoh di atas, string descriptor digunakan dalam statement tersebut :

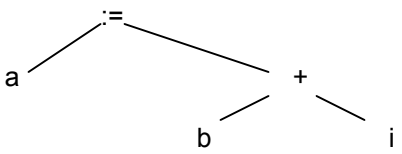
Id# 5	‘=’	Id# 2	‘+’	Id# 11

Symbol tabelnya adalah :

	Id	Other info
# 2	b	
# 5	a	
# 11	i	

Ketika berada dalam semantic analysis dan atau fase sintesis, kita mendapatkan variabel name a, b, i dan bebera informasi seperti tipe data dan atribut lain yang dimasukkan ke dalam table. Aspek mendasar dari lexical analysis adalh membangun table yang benar seperti tabel identifier, table contant, dan sebagainya. Pembuatan table dan desriptor adalah tugas syntax analysis karena terlepas dari nature dan panjang lexicalunit dalam source text. Syntax analysis hany melihat keseragaman deskripsi yang terdiri dari (i) kategori sintaks lexical unit (ii) pointer ke dalam tabel. Bebrapa tabel dibangun oleh lexical analyzer yang digunakan untuk menurunkan inforamsi tahap demi tahap dalam fase analysis, sebgai contoh symbol tabel berisi informasi seperti tipe, panjang, dimensi dan sebaginya yang penting utnuk mengalokasikan storage dalam progarm variable. Beberapa tabel disebut allocation table.

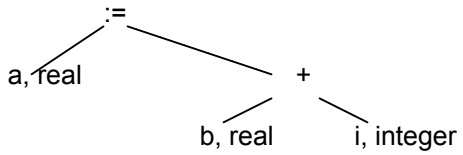
- Syntax Analysis
  Proses syntax analysis dilakukan terhadap desriptor dari lexical analysis untuk menentukan struktur sintaks dari input statement. Proses tersebut dikenal dengan PARSING. Output dari parsing adalah representasi dari struktur sintaks suatu statement. Representasinya digambarkan melalui sintaks tree. Contoh statement a:=b + i , direpresentasikan dalam sintaks tree :



Gambar 3.1(a)

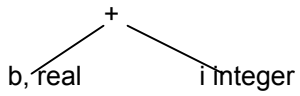
Syntax error seperti hilangnya operator/operand bisa ditemui dalam syntax analysis.

- Semantic Analysis
  Proses semantic analysis dapat diklasifikasikan ke dalam (i) proses declarative statement (ii) proses eksekusi seperti imperative statement. Selama proses semantic, declarative statement dan informasi tambahan dimasukkan ke dalam lexical table. Tipe, panjang dan dimensi variable dan prosedur parameter dimasukkan ke dalam tabel juga. Untuk contoh di atas, statement real a, b, root real diindikasikan sebagai tipe (diasumsikan panjangnya adalah default), yang dimasukkan ke dalam dimbol tabel dengan id a dan b. Pada saat proses eksekusi statement, informasi dari lexical table digunakan untuk menentukan validitas semantic, Tree-nya menjadi :



Gambar 3.1 (b)

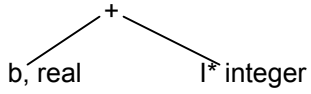
Semantic analysis akan menganalisa sintaks tree untuk mengidentifikasikan elemental evaluation dan menerapkan aturan validitas elemental evaluation.
Perhatikan statement a : b + i untuk melihat detail semantic analysis. Pertamakita perlu menggabungkan tipe dan panjang ke dalam sintaks tree seperti pada gambar 3.1 (b). Ekspresi yang ada di sisi kanan akan dievaluasi sebelum diberikan ke sisi kiri, seperti yang ditunjukkan oleh gambar 3.1(c)



Gambar 3.1(c)

Dari analisis tersebut terlihat bahwa tipe operand tidak compatible. Karenanya diperlukan operasi pengkoreksian :

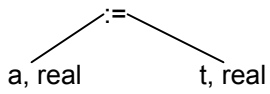
(i) mengkonversi I ke dalam bentuk real I*, seperti terlihat pada gambar 3.1(d)



Gambar 3.1(d)

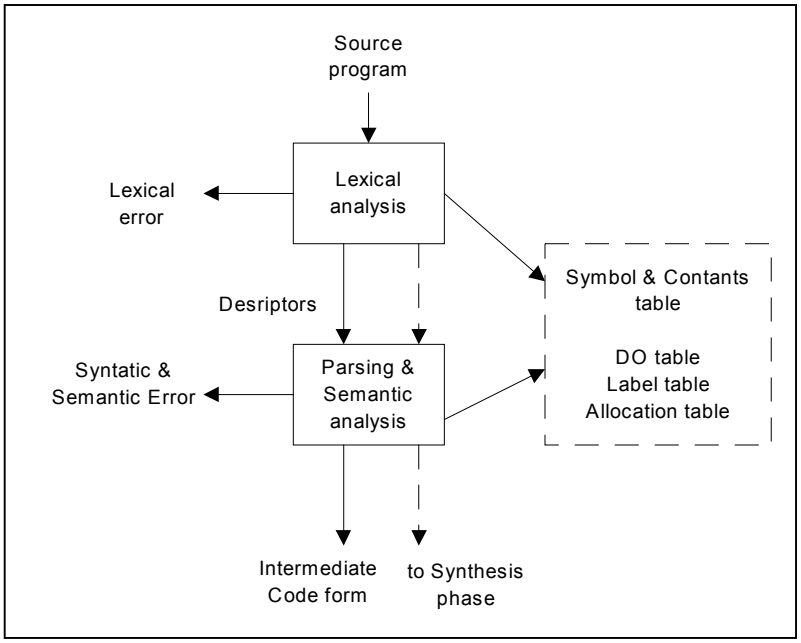
(ii) menambahkan I* dengan b dan diberikan simbol t, dimana t bersifat temporer.

(iii) menyimpan t dalam a



Gambar 3.1(e)

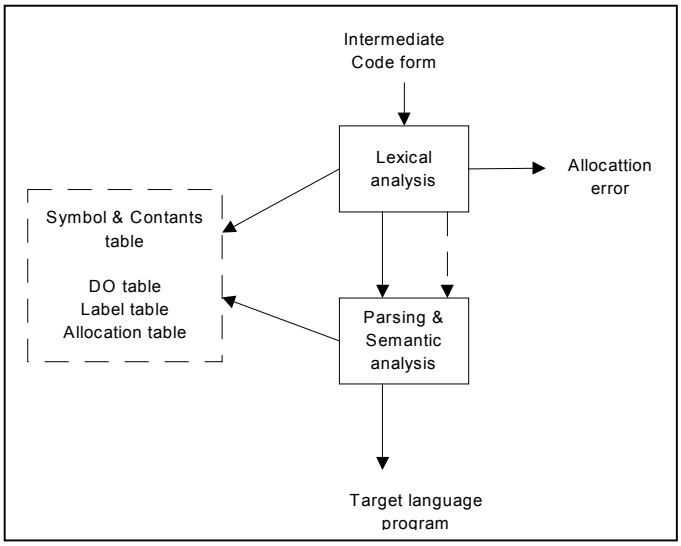
Tahap evaluasi direpresentasikan dalam intermediate code form. Semantic analysis juga akan melahirkan aspek lain daristruktur program yang penting untuk sintesa target text. Inforamsi mengenai aspek ini diletakan dalam structure tabel yang akan digunakan kemudian. Gambar 3.2. berikut mengilustrasikan skema fase analysis.



Gambar 3.2. Fase Analysis Compiler

3.2.2. Fase Synthesis

Syntesa target program terdiri dari tahap (i) storage allocation dan (ii) code generation, seperti terlihat pada gambar 3.3.



Gambar 3.3. Fase Synthesis Compiler

- Storage Allocation
  Storage allocation adalah tahapan yang ada dalam allocation table. Kebutuhan storage dihitung dari beragam informasi seperti panjang dan dimensi peyimpanan dalam tabel dan unsur yang dialokasikan dalam machine address. Alamt pengalokasian dimasukkan ke dalam field dalam table yang mengacu pada unsur utama dalam intermediate code ayng dapat dikonversi ke dalam storage allocation yang berkesesuaian dengan target code. Setelah storage aalocation, symbol table tampak seperti berikut :

	Id	Type	Address
# 2	b	Real	2000
# 5	a	Real	2001
# 11	i	integer	2002

- Code generation
  Code generation ditentukanoleh aspek pragmaticdari compiler. Pengetahuan mengenai insruction set, addressing mode dan sebagainya adalah pengetahuan tentang arsitektur komputer yang dignakan untuk men-generate target program instruction. Tahapannya adalah:
  (i) menkonversi i ke dalam real dengan tanda i*
  (ii) menambahkan i dengan b dan memberi tanda t
  (iii) menyimpan t dalam a
  Keputusan yang dibuat :
  (a) urutan instruksi yang digunakan untuk tipe operasi konversi
  (b) mode pengalamatan yang digunakan untuk mengakses I, b dan a
  (c) mengambil nilai I* dan t di storage allocation atau machine register

  Beberapa hal terkait dengan arsitektur mesin. Dalam hal ini dibutuhkan aspek code optimisation yang akan menganalisis program dan mengumpulkan informasi yang berkaitan dengan progaram struktur dan pendefinisian pengggunaan data dalam program.