TEKNIK KOMPILASI

Ernastuti & Sulistyo

TUJUAN

- Mengetahui Penerapan konsep ilmu komputer pada perilaku komputer yaitu algoritma, arsitektur komputer, stuktur data maupun penerapan teori bahasa dan automata
- Compiler adalah merupakan konstruksi inti dari ilmu komputer

MATERI

- Pendahuluan: arti dari Kompilasi
- Translator: Compiler dan interpreter
- Bahasa Pemrograman
- Pembuatan Compiler
- Konsep bahasa dan Notasi
- Hirarki Comsky
- Aturan Produksi
- Diagram state
- Notasi BNF
- Diagram Syntax
- Kualitas Compiler

- Beberapa translator
- Struktur Compiler
- Lexical Analysis + contoh
- Analysis Syntax + contoh
- Analysis Semantics + contoh
- Error Handling
- Optimation
- Tabel informasi

ARTI KATA TEKNIK KOMPILASI

Teknik

adalah suatu Metode atau Cara

Kompilasi

suatu Proses mengabungkan serta menterjermahkan sesuatu (source program) menjadi bentuk lain

Compile :

To translate a program written in a high-level programming language into machine language.

TRANSLATOR: COMPILER & INTERPRETER

Translator :

adalah suatu program dimana mengambil input sebuah program yang ditulis pada satu bahasa program (*source language*) ke bahasa lain (*The object on target language)*

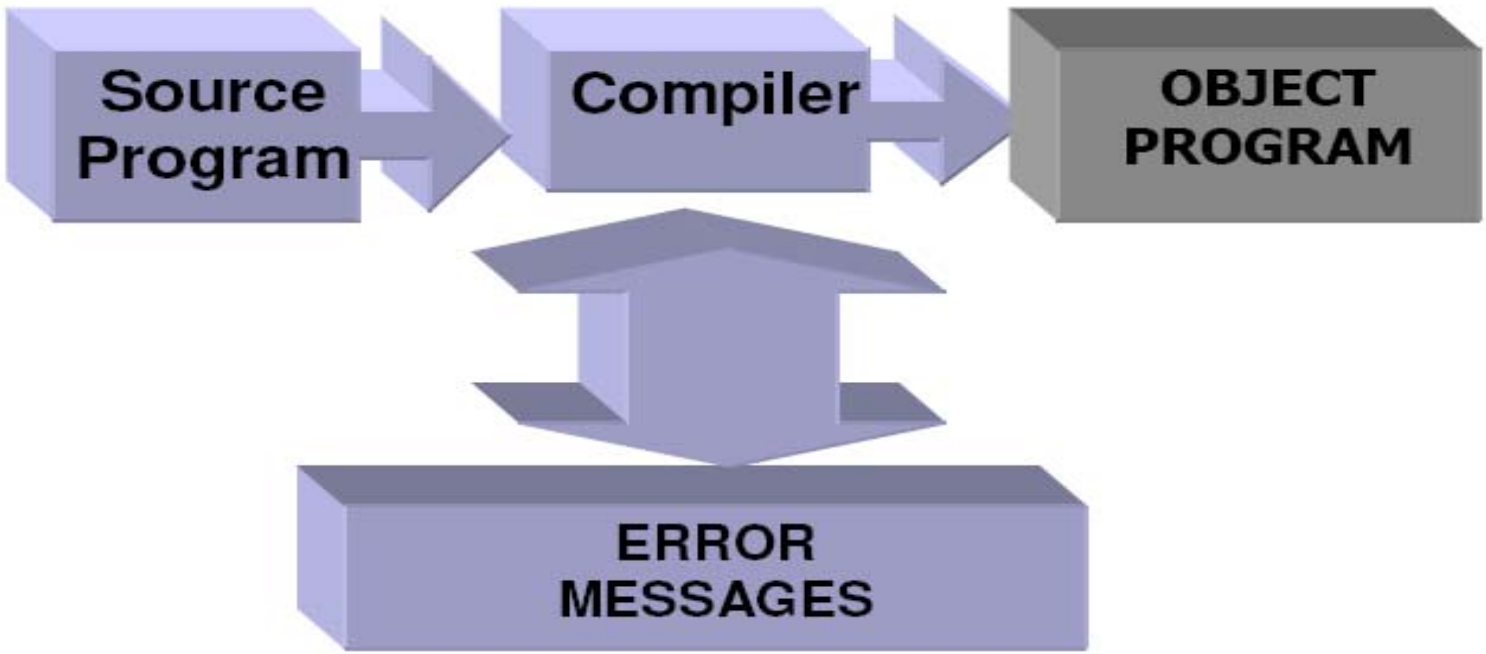
COMPILER & INTERPRETER

Jika source language adalah high level language, seperti cobol, pascal, fortran dan object language adalah low-level language atau mesin language. Maka translator seperti ini disebut COMPILER

Proses perubahan dari source program menjadi object program melalui suatu translator yaitu compiler atau interpreter. meskipun beda pada proses menterjemahkan tetapi fungsi dari interpreter dan compiler adalah sama

COMPILER

Dibawah ini ilustrasi sebuah peterjemah *compiler* menterjemahkan source code mejadi *object file*



Gambar 1: proses penterjemahkan

KENAPA PERLU TRANSLATOR ?

Bagi user yang hanya pengguna mungkin kata-kata translator adalah membingungkan, Kenapa perlu Translator?

- Pertanyaan ini akan membingungkan bagi programmer yang membuat program dengan bahasa mesin.

LATAR BELAKANG

- Bahasa Mesin adalah bentuk bahasa terendah pada komputer, kita dapat berhubungan/komunikasi langsung dengan bagian-bagian yang ada didalam komputer seperti *bits*, register & sangat primitive

- Bahasa mesin adalah tidak lebih dari urutan bit-bit 0 dan 1

- Instruksi dalam bahasa mesin bisa saja dibentuk menjadi *micro-code*, semacam prosedur dalam bahasa mesin

Bagaimana dengan orang yang tidak mengerti bahasa mesin?

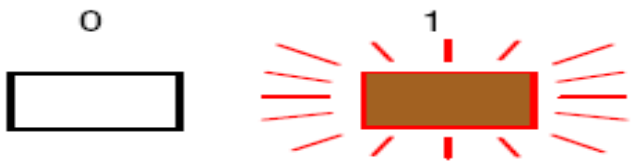
Bagi user yang tidak mengerti bahasa mesin akan mengalami masalah, Hal ini disebabkan karena mereka harus belajar duhulu bahasa mesin, dan akan bergantung pada jenis mesin komputer yang digunakan. Jika jenis komputer mengalami perubahan maka dapat dipastikan bahwa *user* harus memperlajari lagi bahasa mesin dengan jenis komputer yang baru

Oleh karena itu manusia berusaha dan menciptakan suatu bahasa yang dapat dimengerti baik oleh manusia maupun oleh komputer, Bahasa yang demikian ini sering disebut dengan bahasa tingkat tinggi.

Untuk era sekarang *user* tidak lagi banyak dipusingkan mengenai ***penterjemah*** karena kemudahan-kemudahan yang diberikan oleh bahasa tingkat tinggi sekarang sangatlah memudahkan dan lebih fleksibel dalam bekerja pada mesin-mesin komputer yang berbeda.

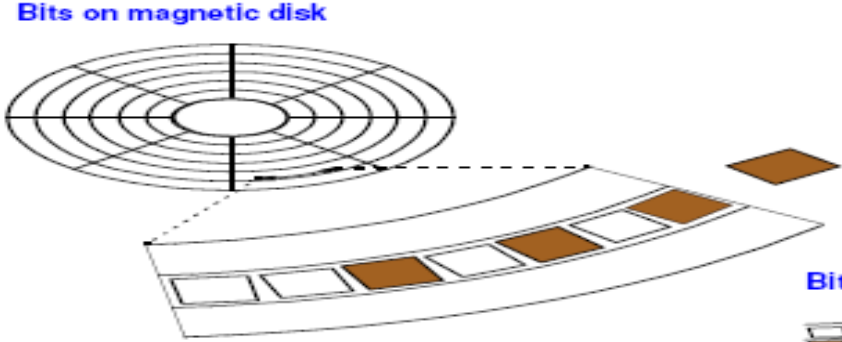
Dibawah ini terdapat ilustrasi mengenai bit-bit yang dikenal oleh komputer dalam mengerjakan sesuatu

The Bit



The bit is the smallest element of computer storage. It is a positive or negative magnetic spot on disk and tape and charged cells in memory.

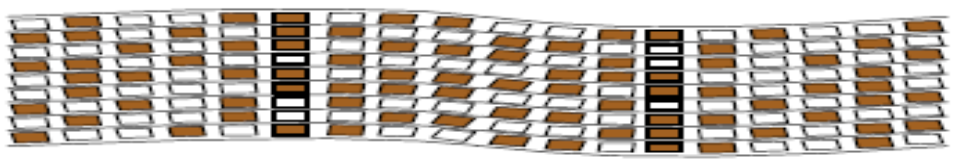
Bits on magnetic disk



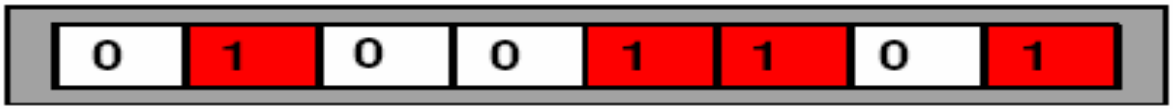
1955
2,000
bits per sq. in.

1995
250,000,000
bits per sq. in.

Bits on magnetic tape



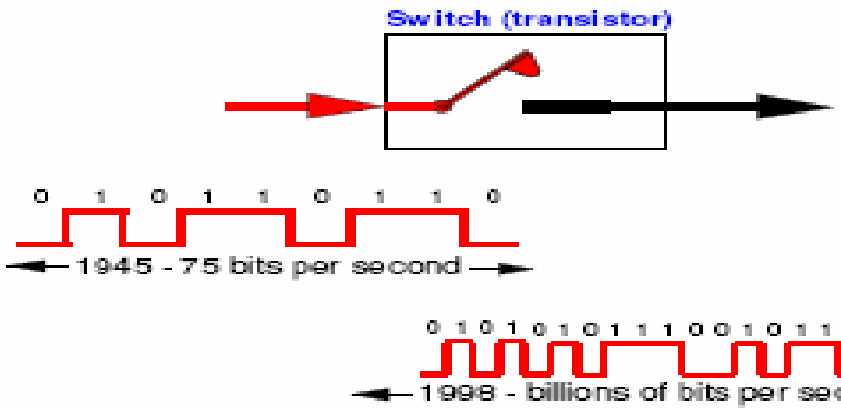
The Byte



A byte is 8 binary digits, or cells.

Bytes in memory

In a 16 megabyte memory, there are 16 million of these 8-bit structures.



BAHASA TINGKAT TINGGI

Pemrograman bisa menggunakan *Bahasa Tingkat Tinggi.*

Bahasa Tingkat Tinggi adalah:

- Bahasa yang lebih dikenal oleh manusia, maksudnya adalah *statement* yang digunakan menggunakan bahasa yang dipakai oleh manusia (inggris),
- Memberikan fasilitas yang lebih banyak, seperti struktur kontrol program yang terstruktur, memiliki blok-blok, serta prosedur dan fungsi-fungsi
  - **Kontrol struktur seperti :**
    - ✓ kondisi (if .. Then.. Else),
    - ✓ perulangan (For, while),
    - ✓ Struktur blok (begin.. End { .. })
- Program mudah untuk di koreksi dan diperbaiki (debug)
- Tidak tergantung pada salah satu jenis mesin komputer
- Bahasa tingkat tinggi biasanya masih membutuhkan translator

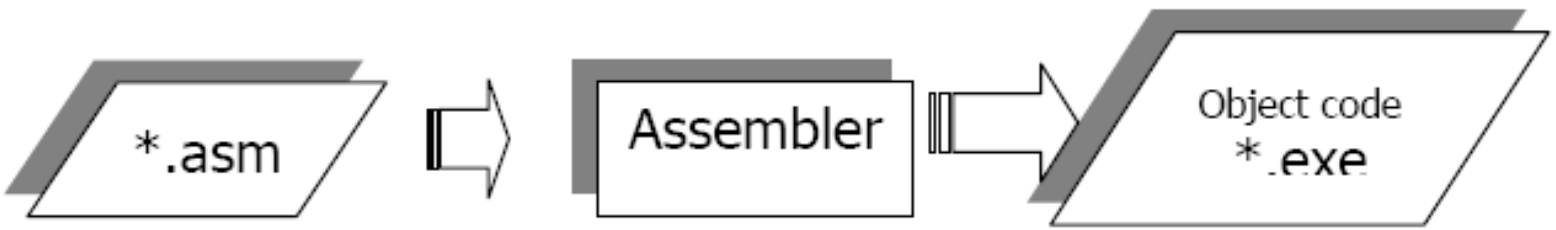
Oleh karena itu dari bahasa tingkat tinggi kedalam bahasa mesin dibutuhkan sesuatu untuk menterjemahkan agar mesin (komputer) mengerti apa yang inginkan oleh manusia. Menerjemahkan statement bahasa tingkat tinggi ke dalam bahasa tingkat rendah dapat dibedakan menjadi dua; melalui *interpreter* atau *compiler* yang fungsinya adalah sama yaitu menterjemahkan.

Jenis Translator: ASSEMBLER

Ada Beberapa jenis Translator untuk menterjemahkan agar dikenali oleh mesin, diantaranya

1. Assembler

Source code adalah bahasa *assembly*, *Object code* adalah bahasa mesin

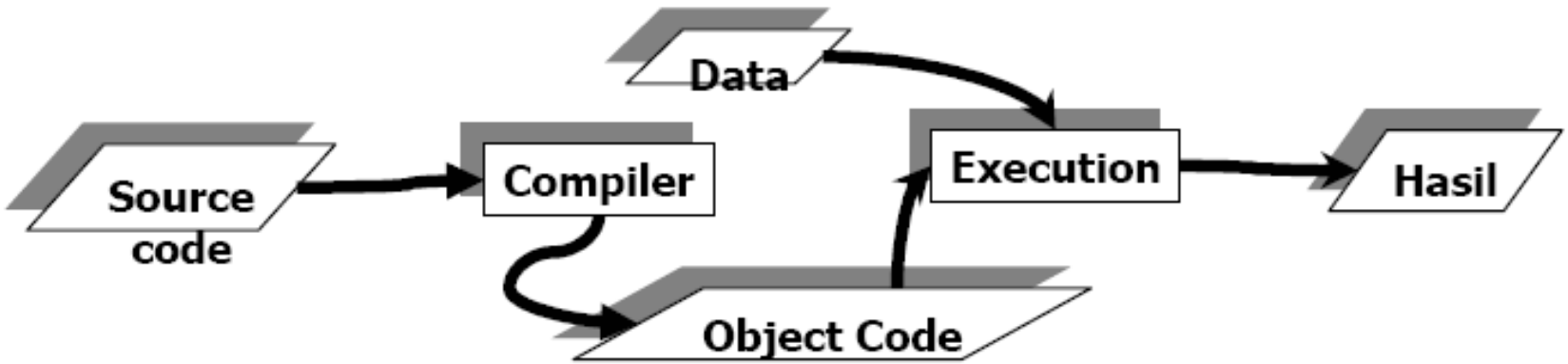


Gambar 3: Penterjemah assembler

Jenis Translator: COMPILER

2. Compiler

Source code adalah bahasa tingkat tinggi, *object code* adalah bahasa mesin atau bahasa *assembly*. Source code dan data diproses berbeda

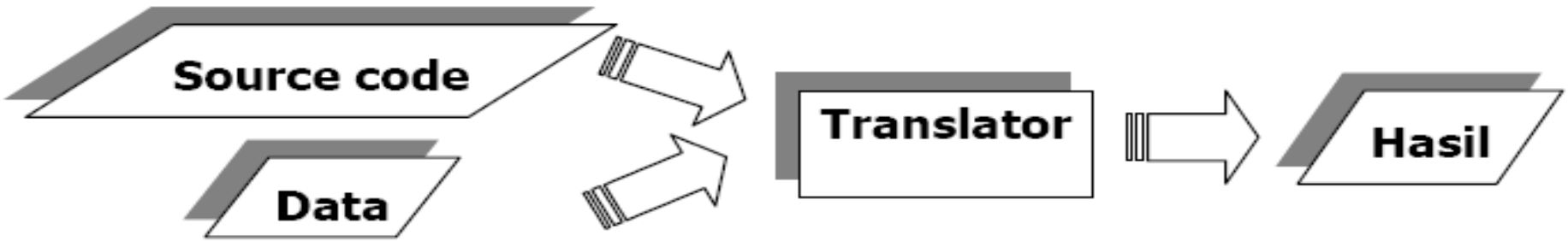


Gambar 3: Penterjemah Compiler

Jenis Translator: Interpreter

3. Interpreter

Interpreter tidak menghasilkan bentuk *object code*, tetapi hasil translasinya hanya dalam bentuk internal, dimana program induk harus selalu ada-berbeda dengan compiler



Gambar 4: Penterjemah interpreter

Why study compilers?

- Most CS students do not go on to write a commercial compiler someday, but that's not why we study compilers. We study compiler construction for the following reasons:
- Writing a compiler gives experience with large-scale applications development. Your compiler program may be the largest program you write as a student. Experience working with really big data structures and complex interactions between algorithms will help you out on your next big programming project.
- Compiler writing is one of the shining triumphs of CS theory. It demonstrates the value of theory over the impulse to just "hack up" a solution.
- Compiler writing is a basic element of programming language research. Many language researchers write compilers for the languages they design.
- Many applications have similar properties to one or more phases of a compiler, and compiler expertise and tools can help an application programmer working on other projects besides compilers.

- **There is no software development method for writing large programs that doesn't involve pain: pain is inevitable in software development (Berry's Theorem).**

- **There is no way to learn the skills necessary for writing big programs without pain. A good CS course includes pain, and teaches pain management and minimization.**

- **The questions we should ask, then, are:**

  (a) should CS majors be required to spend a lot of time becoming really good programmers?

  (b) are we providing students with the assistance and access to the tools and information they need to accomplish their goals with the minimal doses of inevitable pain that are required?

What Are They and What Kinds of Compilers are Out There?

- **The purpose of a compiler is:**
  to translate a program in some language (the *source language*) into a lower-level language (the *target language*).

The compiler itself is written in some language, called the *implementation language*.

To write a compiler you have to be very good at programming in the implementation language, and have to think about and understand the source language and target language.

Several major kinds of compilers

- **Native Code Compiler**
  Translates source code into hardware (assembly or machine code) instructions. Example: **gcc.**

- **Virtual Machine Compiler**
  Translates source code into an abstract machine code, for execution by a virtual machine interpreter. Example: **javac.**

- **JIT Compiler**
  Translates virtual machine code to native code. Operates within a virtual machine. Example: **Sun's HotSpot java machine**.

- **Preprocessor**
  Translates source code into simpler or slightly lower level source code, for compilation by another compiler. Examples: **cpp, m4**.

- **Pure interpreter**
  Executes source code on the fly, without generating machine code.
  Example: **Lisp**.

Phases of a Compiler

- **Lexical Analysis:**
  Converts a sequence of characters into words, or *tokens*

- **Syntax Analysis:**
  Converts a sequence of tokens into a *parse tree*

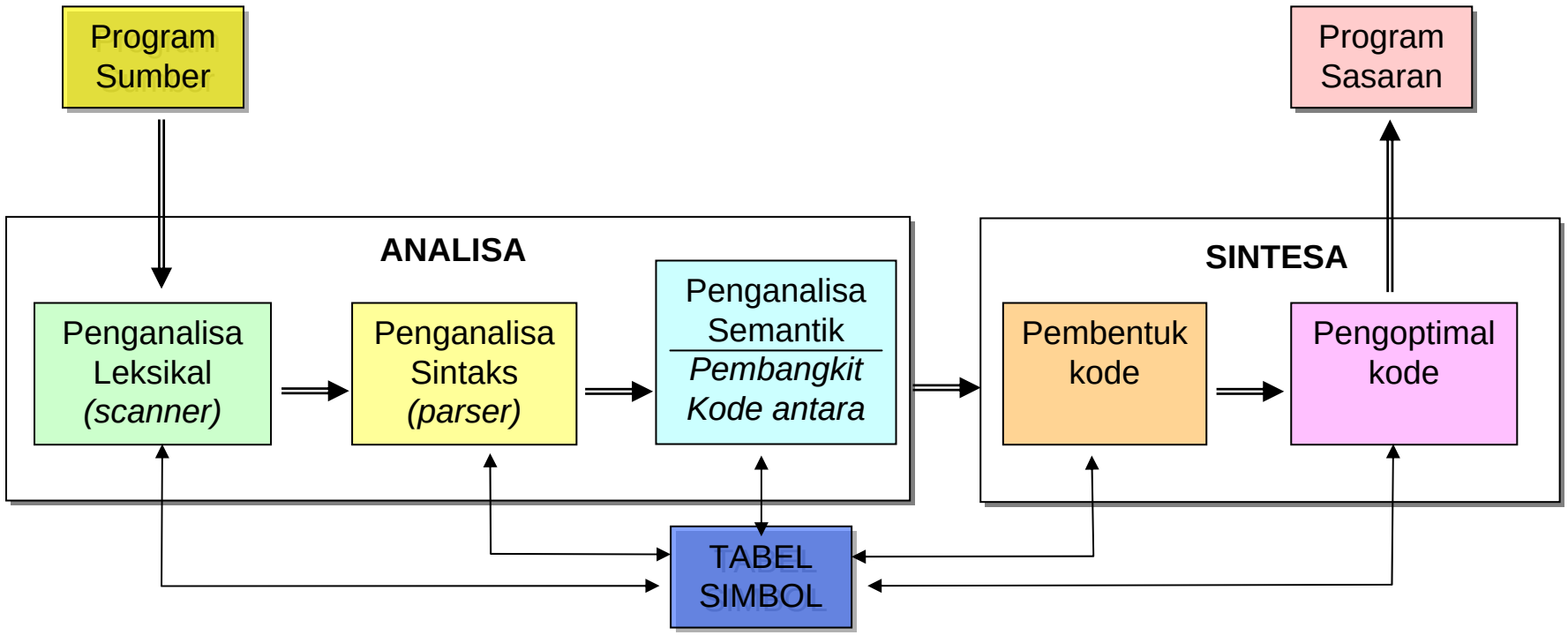
- **Semantic Analysis:**
  Manipulates parse tree to verify symbol and type information

- **Intermediate Code Generation:**
  Converts parse tree into a sequence of intermediate code instructions

- **Optimization:**
  Manipulates intermediate code to produce a more efficient program

- **Final Code Generation:**
  Translates intermediate code into final (machine/assembly) code

Blok Diagram



Bagan pokok proses kompilasi

Pembuatan compiler

Kompiler yang bagus adalah yang dapat bekerja dengan baik pada mesin-mesin computer dan tidak membutuhkan proses yang lama

Ada beberapa cara dalam membuat suatu penterjemah dalam hal ini misalnya compiler, dan bahasa yang digunakan untuk membuat compiler pun beragam misalnya ;

Bahasa mesin

- Sangat sukar dan sangat sedikit kemungkinannya untuk membuat compiler dengan bahasa ini, karena manusia susah mempelajari bahasa mesin,
- Sangat tergantung pada mesin,
- Bahasa Mesin kemungkinan digunakan pada saat pembuatan Assambler

Assembly

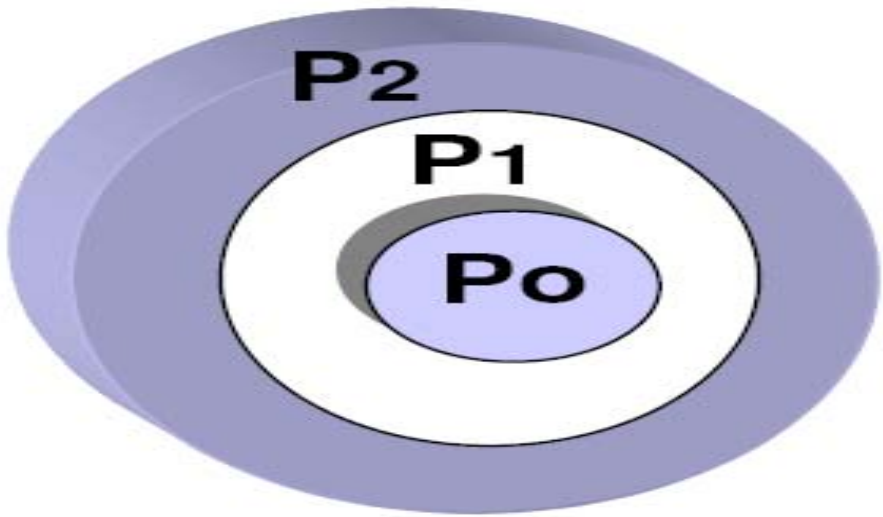
- Hasil dari program mempunyai Ukuran yang relatif kecil
- Sulit dimengerti karena statement/perintahnya singkat-singkat, butuh usaha yang besar untuk membuat compiler dengan bahasa ini
- Fasilitas yang dimiliki terbatas

Bahasa Tingkat Tinggi (high level language)

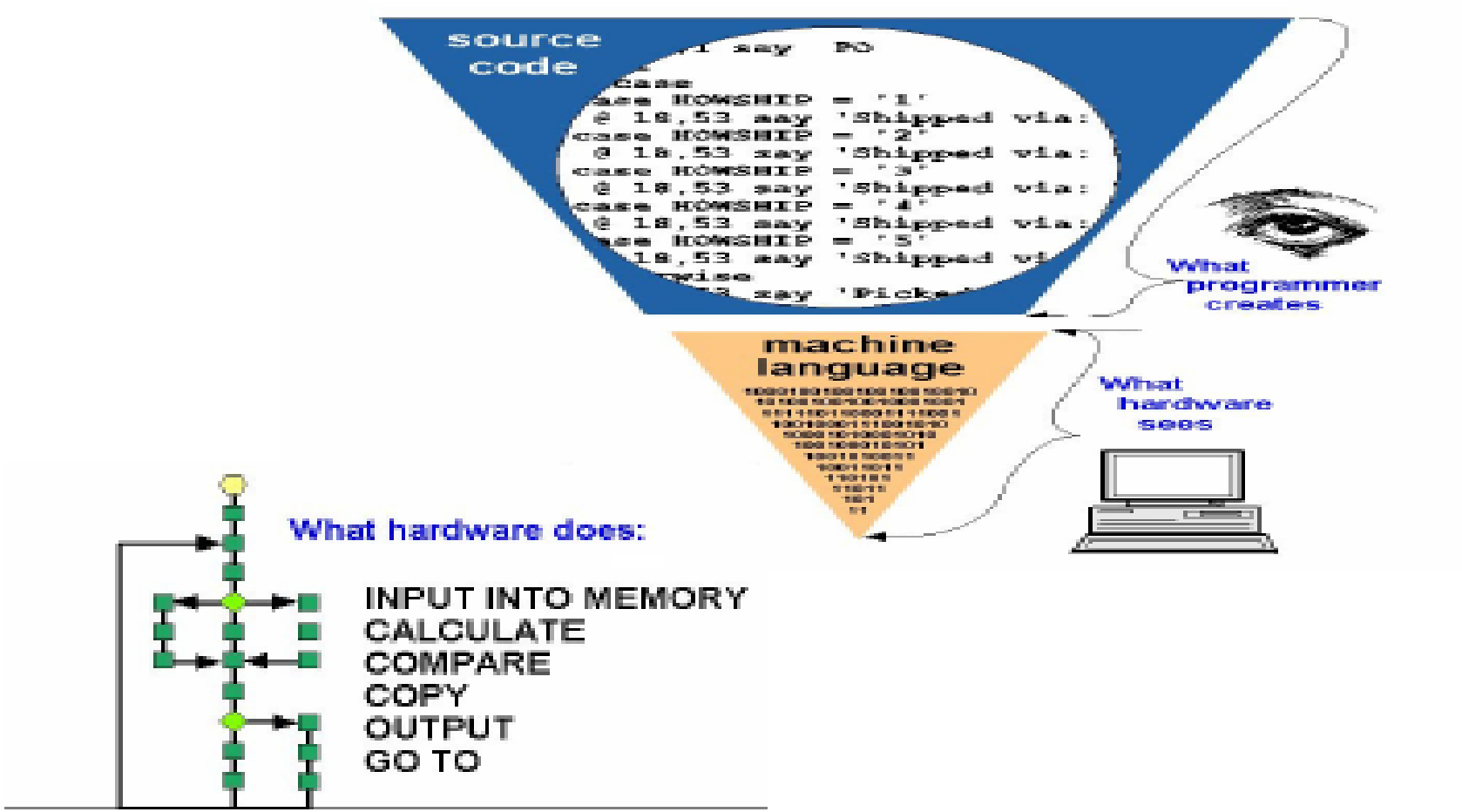
- Lebih mudah dipelajari
- Fasilitas yang dimiliki lebih baik (banyak)
- Memiliki ukuran yang relatif besar, misal membuat compiler pascal dengan menggunakan bahasa C
- Untuk mesin yang berbeda perlu dikembangkan tahapan-tahapan tambahan. Misal membuat compiler C pada Dos bedasarkan compiler C pada unix

BootStrap

- Untuk membangun sesuatu yang besar, dibangun/dibuat dulu bagian intinya (niklaus Wirth - sangat membuat pascal compiler)
- PO dibuat dengan assembly, P1 dibuat dari P0, dan P2 dibuat dari P1, jadi compiler untuk bahasa P dapat dibuat tidak harus dengan menggunakan assembly secara keseluruhan



Contoh Source Program ke dalam Kode Mesin:



Contoh Source Program ke dalam Kode Mesin

Source code	Assembly Language	Machine language

IF COUNT =10
GOTO DONE
ELSE
GOTO AGAIN
ENDIF
	Compare A to B
	If equal go to C
	Go to D
		Compare 3477 2883
		If = go to 23883
		Go to 23343

Actual machine code

```
10010101001010001010100
10101010010101001001010
10100101010001010010010
```

Konsep dan Notasi Bahasa

Untuk membuat penterjemah seperti compiler perlu dibuat standard atau aturan atau tata bahasa, seperti manusia berkomunikasi mempunyai tata bahasa agar lawan bicaranya dapat mengerti yang dibicarakan.

Demikian juga untuk menerterjemahkan kedalam mesin (computer) harus dibuat suatu aturan agar computer dapat mengerti apa yang diinginkan oleh manusia melalui program yang dibuatnya;

Konsep dan Notasi Bahasa

- Teknik Kompilasi merupakan kelanjutan dari konsep-konsep yang telah kita pelajari dalam teori bahasa dan automata
- Tata bahasa (grammar) adalah sekumpulan dari himpunan variabel-variabel, simbol-simbol terminal, simbol non-terminal, simbol awal yang dibatasi oleh aturan-aturan produksi

Konsep dan Notasi Bahasa

- Tahun 56-59 Noam chomsky melakukan penggolongan tingkatan dalam bahasa, yaitu menjadi 4 class
- Penggolongan tingkatan itu disebut dengan hirarki Comsky
- 1959 Backus memperkenalkan notasi formal baru untuk syntax bahasa yang lebih spesifik
- Peter Nour (1960) merevisi metode dari syntax. Sekarang dikenal dengan BNF (backus Nour Form)

Contoh Tata Bahasa Sederhana

<program> → **BEGIN** <Statement-list> **END**

<Statement-list> → <statement> | <statement>**;** <statement-list>

<statement> → <var> **:=** <expression>

<Expression> → <term> | <term><op1> <expression>

<term> → <factor> | <factor> <op2> <term>

<factor> → <var> | <constant>

<var> → **A**|**B**| ....| **Z**

<op1> → + | - | =

<op2> → ^ | * | /

<constant> → <real_number> | <integer_part>

<real_number> → <integer_part> . <fraction>

<integer_part> → <digit> | <integer_part> < digit>

<fraction> → <digit> | <digit> <fraction>

<digit> → 0|1|....|9

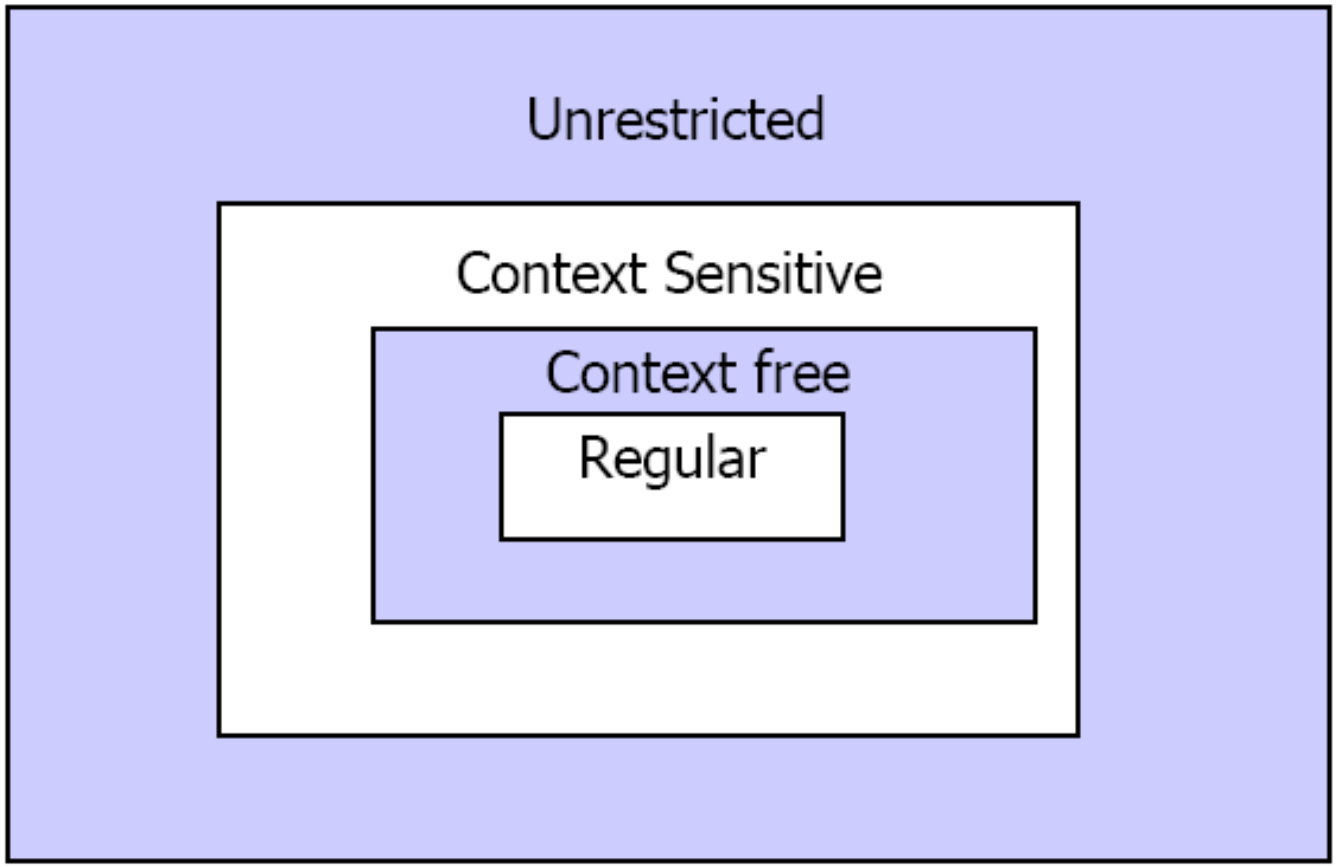
Contoh

```
Begin
        A := 1;
        B := A + 2
END
```

Tabel Aturan Produksi

Bahasa	Mesin Automata	Aturan Produksi
Tipe 3 Atau Regular	Finite state automata (FSA) meliputi; deterministic Finite Automata (DFA) & Non Deterministic Finite Automata (NFA)	α adalah simbol variabel β maksimal memiliki sebuah simbol variabel yang bila ada terletak diposisi paling kanan
Tipe 2 Atau Contex Free	Push Down Automata	α adalah simbol variabel
Tipe 1 Atau Contex Sensitive	Linier Bounded Automata	$\lvert\alpha\rvert <= \lvert\beta\rvert$
Tipe 0 Atau Unrestricted/ Phase Structure/ natural language	Mesin Turing	Tidak ada Batasan

Hirarki CHOMSKY



Keterangan Gambar

- **Tipe O** / Unrestricted: Tidak Ada batasan pada aturan produksi

  $Abc \rightarrow De$

- **Tipe 1** / Context sensitive: Panjang string ruas kiri harus lebih kecil atau sama dengan ruas kanan

  $Ab \rightarrow DeF$

  $CD \rightarrow eF$

- **Tipe 2** / Context free grammar: Ruas kiri haruslah tepat satu simbol variable

  $B \rightarrow CDeFg$

  $D \rightarrow BcDe$

- **Tipe 3** / Regular: Ruas kanan hanya memiliki maksimal 1 simbol non terminal dan diletakkan paling kanan sendiri

  $A \rightarrow e$

  $A \rightarrow efg$

  $A \rightarrow efgH$

  $C \rightarrow D$

ATURAN PRODUKSI

Aturan produksi digunakan agar penerapan pada pembuatan tata bahasa dikomputer dapat lebih mudah dan menghasilkan suatu penterjemah yang dapat diandalkan.

- Aturan produksi dinyatakan dalam bentuk $\alpha \rightarrow \beta$, α menghasilkan/ menurunkan β
- α simbol-simbol untuk ruas kiri dan β simbol-simbol untuk ruas kanan
- Simbol-simbol bisa berupa ternimal dan Non-terminal, dimana Non-terminal masih bisa diturunkan menjadi simbol yang lainnya

- Umumnya simbol terminal disimbolkan dengan huruf kecil (a,b,c dst), sedangkan untuk simbol non-terminal disimbolkan dengan huruf besar (A, B, C, dst)

- Contoh aturan produksi :

  $T \rightarrow a$, T menghasilkan a

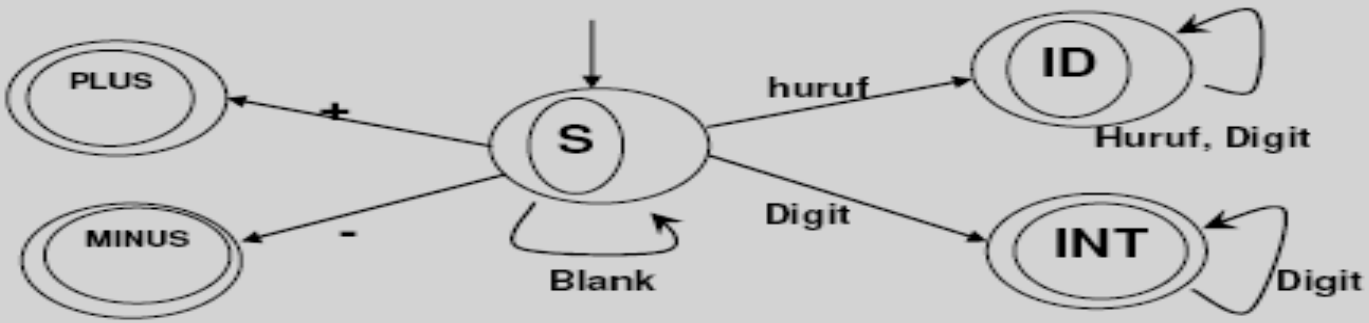
  $E \rightarrow T \mid T + E$, E mengasilkan T, atau E menghasilkan T + E

DIAGRAM STATE

Bagi pembuat penterjemah yaitu manusia harus sering menguji tata bahasa yang dibuat, salah satu ilustrasi pengujian agar yang diinginkan sesuai dengan yang diharapkan maka digunakan suatu gambar yaitu yang dinamakan dengan *diagram state*

- Digunakan untuk mendapatkan token, mempermudah melakukan analisis lexical
- *Token* adalah simbol terminal dari teori bahasa dan automata
- Contoh token ID untuk karakter huruf a-z, 0-9, token INT untuk digit, token PLUS untuk menjumlahan dan token MINUS untuk Pengurangan

Dibawah ini contoh gambar diagram state



Notasi BNF (Backus Normal Form)

Selain diagram state perlu adanya suatu notasi standard dalam penyimbolan

- Aturan Produksi bisa dinyatakan dengan notasi BNF
- BNF menggunakan abstraksi untuk struktur syntax

::=	sama identik dengan simbol →
|	sama dengan atau
< >	pengapit simbol non terminal
{ }	Pengulangan dari 0 sampai n kali

Misalkan aturan produksi sbb:

$E \rightarrow T \mid T+E \mid T-E$

$T \rightarrow a$

Notasi BNFnya adalah

E ::= <T> | <T> + <E> | <T> - <E>

T ::= a

DIAGRAM SYNTAX

Selain diagram state dan BNF diagram sintak merupakan alat Bantu.

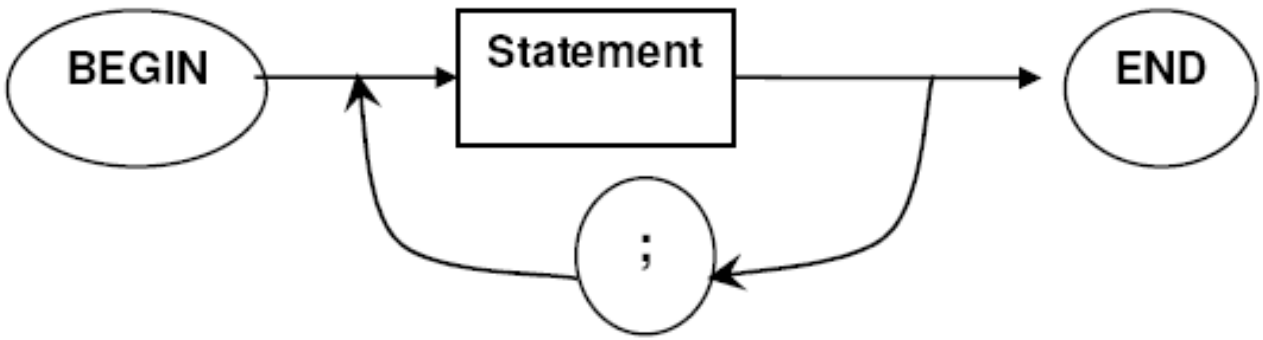
- Alat bantu (tools) dalam pembuatan parser/ analisis sintaksis
- Menggunakan simbol persegi panjang untuk non terminal
- Lingkaran untuk simbol terminal

DIAGRAM SYNTAX

Misalnya

$E \rightarrow T \mid T+E \mid T-E$

BNF: <Block> ::= BEGIN <statement> { SEMICOL <statement>} END



Kualitas dari Compiler:

- Waktu yang dibutuhkan untuk kompilasi
  - ✓ Algoritma compiler
  - ✓ Pembuat (compilator) Compiler itu sendiri
- Kualitas dari obyek program yang dihasilkan
  - ✓ Ukuran yang dihasilkan
- Fasilitas-fasilitas Integrasi yang lainnya
  - ✓ IDE (integrated Development Environment)

Lanjut ke TEKNIK KOMPILASI II.ppt