BAHASA PEMROGRAMAN BASIC UNTUK MIKROKONTROLER PIC16F84A

Santoso[1], Sumardi[2], R. Rizal Isnanto[2]
Jurusan Teknik Elektro Fakultas Teknik Universitas Diponegoro Semarang

ABSTRAK - *Bahasa rakitan mikrokontroler merupakan bahasa yang relatif sulit untuk dipelajari. Bahasa pemrograman lainnya, seperti bahasa pemrograman BASIC umumnya mudah untuk dipelajari. Oleh sebab itu dibuat suatu penerjemah bahasa mirip BASIC untuk mikrokontroler untuk memudahkan pembuatan perangkat lunak untuk mikrokontroler.*

Mikrokontroler yang digunakan dalam penelitian ini adalah PIC16F84A. Bahasa pemrograman 'sistem BASIC' dibuat dalam dua tahap. Pada tahap pertama dibuat kompiler di komputer pribadi (PC) yang menerjemahan kode sumber menjadi kode antara. Kode antara ini kemudian dikirim ke PIC16F84A. Pada tahap kedua dibuat penerjemah di PIC16F84A yang menginterpretasikan dan mengeksekusi aksi menurut kode antara. Pengujian dilakukan dengan membuat perangkat keras sederhana dan perangkat lunak yang menggunakan seluruh instruksi dan operator yang didukung oleh 'sistem BASIC'.

Meskipun interpreter diterapkan pada PIC16F84A yang memori programnya hanya berukuran 1 kiloword, berdasarkan hasil pengujian, seluruh instruksi dan operator mampu bekerja sesuai dengan perancangan.

***Kata Kunci** : Interpreter, Mikrokontroler PIC16F84A, Kompiler, Kode Antara, 'sistem BASIC'.*

I. PENDAHULUAN

Pemrograman mikrokontroler yang memakai bahasa tingkat tinggi ada beberapa metode, antara lain : kompilasi, kode sumber yang dibuat langsung diterjemahkan ke dalam bahasa rakitan mikrokontroler; dan interpretasi, kode sumber diterjemahkan dahulu oleh mikrokontroler sebelum dieksekusi. Gabungan kedua metode dipilih; untuk menyederhanakan kerja interpreter proses kompilasi dipilih, dan karena proses pembuatan program lebih mudah jika dilakukan di PC. Interpreter dipilih karena kesederhanaannya.

Tata bahasa BASIC merupakan tata bahasa yang populer karena tergolong mudah untuk dipelajari, dan cocok untuk diterapkan karena relatif tidak rumit penerapannya jika dibandingkan dengan tata bahasa tingkat tinggi lainnya, misalnya C dan Pascal.

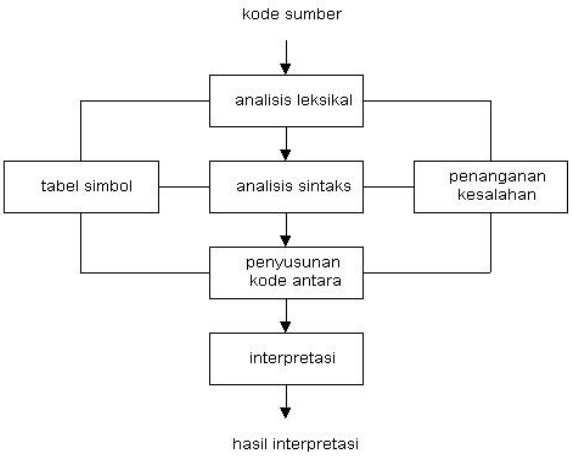
Untuk menerapkan interpreter sistem BASIC (sisBASIC), digunakan mikrokontroler Microchip PIC16F84A. Interpreter ini bekerja dengan membaca hasil kompilasi yang disimpan pada EEPROM Atmel AT24C64, menerjemahkan, dan mengeksekusinya. Kompiler dirancang untuk digunakan pada PC, memiliki fasilitas editor teks, jendela terminal port serial untuk sarana menampilkan pesan, dan transfer program ke sistem interpreter.

II. LANDASAN TEORI

Penerjemah (translator) berfungsi menerima masukan kode sumber dan mengubahnya menjadi suatu program objek atau target, dengan kode sumber ditulis dalam bahasa sumber dan program objek didefinisikan dalam bahasa objek [6]. Interpreter bekerja dengan menerjemahkan instruksi-instruksi yang diberikan oleh pemakai, sehingga komputer mampu memahami dan mengerjakannya. Interpreter dan kompiler dikategorikan ke dalam jenis penerjemah.

[1] Mahasiswa Teknik Elektro Undip
[2] Staf Pengajar Teknik Elektro Undip

Perbedaan antara interpreter dan kompiler terletak pada hasil kerja masing-masing. Interpreter memberikan hasil dari interpretasi dan pengerjaan instruksi-instruksi masukannya, sedangkan kompiler memberikan kode objek yang biasanya dalam bahasa mesin [6].



Gambar 2.1 Tahapan-tahapan kerja interpreter.

Proses kerja interpreter dibagi menjadi beberapa tahapan yang diperlihatkan oleh Gambar 2.1. Karena interpreter ini diterapkan pada sistem mikrokontroler, tahapan analisis leksikal, analisis sintaksis, dan penyusunan kode antara (*intermediate code*) diimplementasikan pada PC dan tahapan interpretasi diimplementasikan pada mikrokontroler. Pembagian ini menyebabkan tahapan interpretasi tidak bisa mengakses tahapan pengelolaan tabel dan penanganan kesalahan.

A. ANALISIS LEKSIKAL

Pada tahapan ini interpreter mencoba mengenali deretan karakter dari kode sumber. Penganalisis leksikal, yang juga dikenal dengan sebutan *scanner* atau lekser (*lexer*), biasanya mengambil deretan karakter berikutnya dari teks kode sumber, mengkategorikan sebagai suatu token tertentu kemudian diteruskan ke penganalisis sintaks [6].

Lekser bekerja dengan komponen pola, token, dan leksema (*lexeme*). Pola merupakan spesifikasi aturan yang mendefinisikan token. Token merupakan simbol terminal dalam tata bahasa sumber. Leksema adalah kata atau kalimat dalam bahasa sumber yang diwakilkan oleh token. Satu cara pembentukan spesifikasi terstruktur untuk token adalah menggunakan ekspresi reguler (*regular expression* / *regexs*) [1].

Diagram transisi dibentuk dari *regexs* [1]. Diagram transisi cenderung direpresentasikan dengan automata berhingga (*finite automata*). Automata berhingga ada dua jenis, yaitu deterministik (DFA) dan non-deterministik (NFA). Lekser yang diterapkan adalah simulator DFA dengan memakai algoritma [1]:

```
s := s0;
c := nextchar;
while c ≠ eof do
     s := move(s, c);
     c := nextchar;
end;
if s termasuk himpunan F then
     return "yes"
else
     return "no";
```

Masukan untuk algoritma ini adalah teks x yang berakhiran simbol karakter 'end-of-file' (eof) dengan suatu DFA D dengan keadaan mulai s_0 dan himpunan keadaan akhir F. Keluaran algoritma ini adalah "yes" jika deretan karakter dalam teks menyamai pola DFA D, "no" jika sebaliknya.

B. Analisis Sintaks

Dalam memeriksa struktur sintaks bahasa sumber, pengurai mencoba untuk merekontruksi bahasa sumber dari masukan spesifikasi tata-bahasa dan membangun suatu pohon sintaks [1]. Pohon sintaks dipakai untuk membantu membangkitkan kode antara dan juga untuk mendeteksi dan melaporkan jika terjadi kesalahan.

Setiap bahasa pemrograman mempunyai aturan yang merumuskan struktur sintaksisnya. Struktur sintaksis ini diterapkan dengan memakai tata bahasa bebas konteks (*context-free grammar*, CFG) [1].

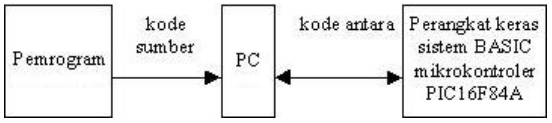
Penggabungan penguraian dan pembangkitan kode antara disebut dengan istilah penerjemahan berarah-sintaks (*syntax directed translation*). Skema penerjemahan (*translation scheme*) merupakan suatu penerapan penerjemahan berarah-sintaks [1].

Kode antara adalah suatu bentuk internal yang dihasilkan oleh penerjemah, yang merupakan bentuk yang dapat dikenali oleh interpreter.

C. *Backpatching*

Jika pengurai menemui suatu bentuk pernyataan yang memakai label untuk loncatan program, masalah dapat ditemui pada analisis sintaks *single pass* jika baris deklarasi label tersebut belum ditemui ketika baris yang memakai loncatan label tersebut dianalisis. Solusi untuk masalah ini adalah dengan menggunakan konsep *backpatching*. Ide dibalik konsep *backpatching* adalah dengan meninggalkan label tidak terdefinisi dan dianalisis belakangan setelah token pada teks kode sumber telah sukses dianalisis sintaksis [1].

III. Perancangan Alat

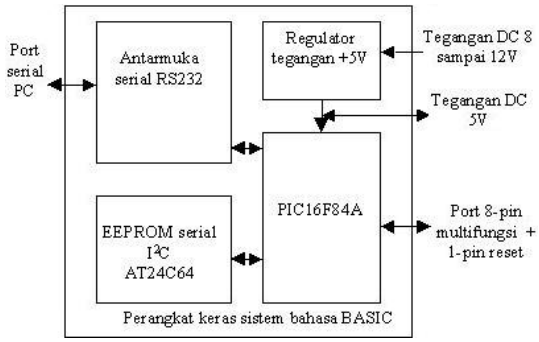


Gambar 3.1 Diagram kotak pemakaian sisBASIC.

SisBASIC bekerja dengan kode program BASIC yang dibuat pemakai pada PC dengan memakai perangkat lunak yang berfungsi memvalidasi masukan kode program dan mengubahnya menjadi kode antara untuk kemudian dikirim ke modul perangkat keras melalui protokol komunikasi serial RS232 untuk disimpan pada IC EEPROM. Hubungan ke PC hanya diperlukan saat pengiriman kode antara dan pelacakan kesalahan, setelah itu sebagai hasil akhirnya modul perangkat keras sistem PIC16F84A dapat berfungsi lepas dari PC dan bekerja berdasarkan kode antara.

A. Perancangan Perangkat Keras

Gambar 3.2 menunjukkan diagram kotak perangkat keras dari rangkaian interpreter sisBASIC. Diagram kotak ini terdiri atas beberapa bagian utama yaitu mikrokontroler PIC16F84A yang berfungsi sebagai interpreter kode antara, EEPROM Atmel AT24C64 yang berfungsi sebagai media penyimpanan kode antara dan antarmuka komunikasi serial RS-232 yaitu MAX232.

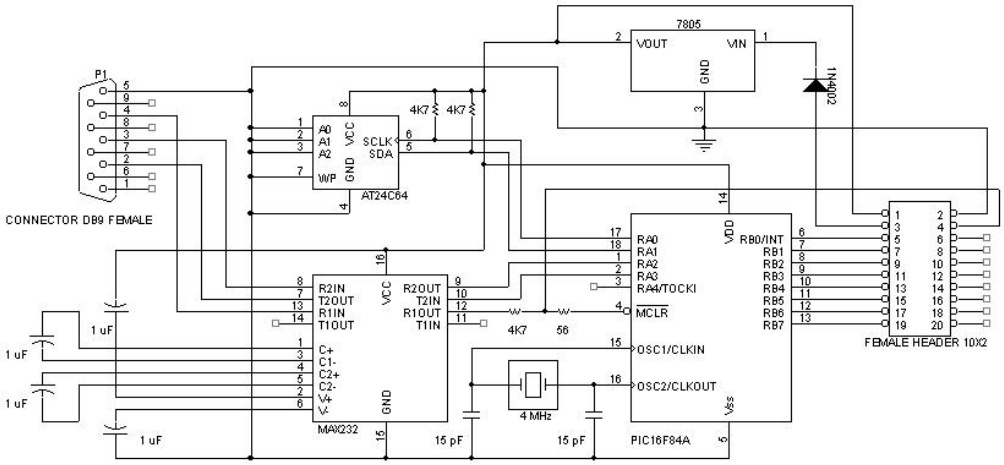


Gambar 3.2 Diagram kotak interpreter sisBASIC.

Untuk menyederhanakan pembuatan perangkat keras maka IC yang dipakai hanya berjumlah empat buah. Masing-masing IC tersebut memiliki fungsi yang diperlukan untuk kerja sistem minimum dari interpreter. Keempat IC tersebut adalah mikrokontroler PIC16F84A, serial EEPROM AT24C64, antarmuka komunikasi RS232 MAX232, dan Regulator +5V 7805.

B. Perancangan Perangkat Lunak Interpreter

SisBASIC PIC16F84A dibuat untuk suatu sistem kendali sederhana dan modul tampilan (display) tidak termasuk di dalamnya, karenanya sistem ini tidak bisa memberikan indikasi jika terjadi kesalahan saat proses interpretasi. Gambar 3.3 menunjukkan skematik perangkat keras interpreter sisBASIC.

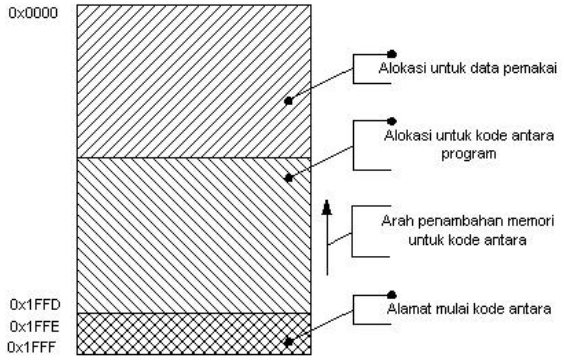


Gambar 3.3 Skematik perangkat keras interpreter sisBASIC.

Tahapan pengiriman kode antara dari PC ke modul perangkat keras PIC16F84A dibuat terlebih dahulu dan karena subrutin komunikasi RS232 dan I^2C merupakan subrutin penting dalam kerja tahapan ini maka dua subrutin ini akan dibuat terlebih dahulu.

1. ALOKASI KODE ANTARA PADA EEPROM AT24C64

Kode antara secara khusus dialokasikan pada bagian akhir AT24C64 dan 2-*byte* alamat terakhir AT24C64 dipakai untuk menunjuk pada alamat awal kode antara. Gambar 3.4 menunjukkan ilustrasi alokasi memori kode antara pada AT24C64.

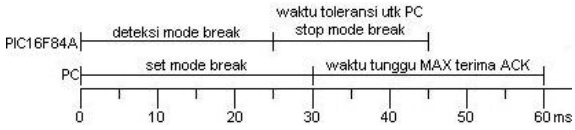


Gambar 3.4 Alokasi memori kode antara pada AT24C64.

2. *BOOTLOADER*

Proses pengiriman kode antara antara PC dan interpreter sisBASIC diterapkan pada saat tepat setelah reset interpreter. Diatur oleh subrutin *bootloader*.

Proses pengaktifan *bootloader* ini dimulai dengan PC mengaktifkan mode *break* RS232. Setelah itu PC mengirim logika rendah untuk mereset interpreter dan mode *break* terus aktif selama selang waktu ±30 milidetik, kemudian status komunikasi RS232 kembali ke mode normal selama interval waktu ±30 milidetik. Setelah itu interpreter mengirim karakter **#** ke PC untuk menandakan *bootloader* aktif dan siap menerima perintah. Diagram pewaktuan pengaktifan *bootloader* dapat dilihat pada Gambar 3.5.



Gambar 3.5 Pewaktuan *break* untuk pengaktifan *bootloader*.

3. REGISTER SISBASIC PIC16F84A

Register yang dimaksudkan merupakan variabel khusus yang dipakai oleh kode sumber sisBASIC untuk mengakses port 8-pin perangkat keras. Register memiliki nama yang baku dan tidak bisa diubah. Tiap-tiap register ini memiliki panjang 8-bit, yang tiap bitnya diasosiasikan dengan pin port sistem. Ada 3 register yang terdefinisi untuk sistem ini yaitu :

a. Register INS - alamat 06_{16} pada PIC16F84A.
b. Register OUTS - alamat 06_{16} pada PIC16F84A.
c. Register DIRS - alamat 86_{16} pada PIC16F84A.

D. IDENTIFIER

Identifier merupakan kata yang tidak termasuk kata-kata yang dicadangkan (*reserved words*). Kata yang dipakai untuk *identifier* harus didefinisikan lebih dahulu. *Identifier* pada sisBASIC digolongkan menjadi 2 bagian yaitu variabel dan konstanta.

a. VARIABEL

Tipe variabel yang didukung hanya ada dua yaitu *word* dan *byte*. Variabel tipe *word* memiliki panjang 16-bit, dapat menampung nilai dari 0-65535, sedangkan variabel tipe *byte* memiliki panjang 8-bit, dapat menampung nilai dari 0-255.

Variasi lainnya yang bisa dipakai dalam variabel ini antara lain bentuk larik (*array*). Variabel disini juga mendukung akses bit secara langsung dengan sintaks: `nama_variabel.BITx`. Setiap bit dapat diakses dengan cara seperti ini dengan x merupakan posisi bit yang ingin diakses, variabel tipe *byte* hanya memiliki BIT0-BIT7 sedangkan tipe *word* memiliki akses BIT0-BIT15. `nama_variabel` disini dapat berupa variabel sederhana atau variabel *array*.

Karena PIC16F84A merupakan mikrokontroler 8-bit dan variabel terbesar yang ditangani sebesar 16-bit

maka format penempatan variabel pada RAM menggunakan suatu metode *little-endian* yaitu 8-bit bawah disimpan lebih dulu kemudian 8-bit atas disimpan pada alamat memori yang lebih besar berikutnya secara berurutan.

b. KONSTANTA

Kode antara untuk konstanta dapat memiliki antara 2 atau 3-*byte* data. Diantara ketiganya, 1-*byte* merupakan *byte* perintah kode antara, dan sisanya adalah nilai dari konstanta itu sendiri. Untuk menghemat *byte* data kode antara maka jika konstanta memiliki nilai kurang dari 256 maka hanya diperlukan 1-*byte* data. Nilai konstanta yang melebihi atau sama dengan 256 memiliki alokasi 2-*byte* data.

E. PROSES INTERPRETASI OPERATOR

Proses interpretasi operator mempunyai alurnya sendiri dan diterapkan sebagai subrutin. Untuk menyederhanakan proses perhitungan maka semua tipe variabel disamakan dengan tipe variabel berukuran terbesar yaitu 16-bit. Karena variabel yang didukung hanya 8-bit dan 16-bit maka variabel 8-bit diubah menjadi 16-bit. Proses ini dilakukan dengan menempatkan nilai variabel pada 8-bit LSB dan menambahkan nilai 0 untuk 8-bit MSB. Jika dalam proses interpretasi instruksi harus menyelesaikan persamaan matematis maka subrutin ini dapat dipanggil dengan perintah bahasa rakitan `call` dan hasil persamaan berupa bilangan 16-bit akan ada pada variabel yang selalu sama yaitu [VAH:VAL]. Berikut adalah 31 operator yang didukung sisBASIC.

1. [+]	2. [SHL]	3. [~]
4. [–]	5. [SHR]	6. [–]
7. [*]	8. [REV]	9. [NOT]
10. [**]	11. [AND]	12. [<>]
13. [/]	14. [OR]	15. [=]
16. [MOD]	17. [XOR]	18. [>]
19. [MIN]	20. [ABS]	21. [<=]
22. [MAX]	23. [DCD]	24. [<]
25. [DIG]	26. [NCD]	27. [>=]
28. Baca Variabel Ke Tumpukan Atas		
29. Tulis Variabel Dari Tumpukan Atas		
30. Masukan Nilai Konstanta Dalam Tumpukan		
31. Keluar Dari Subrutin Interpretasi Operator		

Alur eksekusi operator memakai suatu tabel lompatan. Dengan memakai kode antara operator sebagai indeks tabel untuk perintah bahasa rakitan `goto` untuk melompat ke subrutin yang tertentu untuk menanganinya. Jenis operator yang ditangani ada sebanyak 31 yang memiliki fungsi sebagai operator aritmetika, operator logika, operator akses variabel, operator akses konstanta dan operator keluar dari subrutin interpretasi operator. Indeks operator terkode dalam 5-bit LSB kode antara.

F. PROSES INTERPRETASI INSTRUKSI

Karena PIC16F84A bekerja secara *single-tasking* dan seluruh instruksi kode antara di eksekusi dengan emulasi oleh interpreter maka setiap baris kode sumber dianggap sebagai suatu program independen. Penentuan eksekusi instruksi berdasarkan kode antara dilakukan dengan cara yang sama seperti pada subrutin operator. Berikut ini adalah instruksi-instruksi yang didukung interpreter sisBASIC.

1. BUTTON	2. GOSUB	3. GOTO
4. IF … THEN	5. LET	6. PAUSE
7. PRINT	8. RANDOM	9. READ
10. RETURN	11. SCAN	12. SERIN
13. SEROUT	14. STOP	15. WRITE

C. PERANCANGAN PERANGKAT LUNAK KOMPILER

Perangkat lunak kompiler dibuat dengan menggunakan Microsoft Visual BASIC 6.0 karena program ini tergolong mudah dipakai. Perangkat lunak kompiler dipecah menjadi beberapa modul utama yaitu modul penganalisis leksikal, penganalisis sintaksis dan pembangkitan kode antara.

1. ANALISIS LEKSIKAL

a. EKSPRESI REGULER UNTUK TOKEN KODE SUMBER

Akhir dari baris kalimat dikenali dengan deretan karakter *carriage return* (CR) dan *linefeed* (LF). Deretan ini memiliki representasi token **TOK_EOL**. Berikut merupakan ekspresi reguler untuk **TOK_EOL**,

TOK_EOL → CRLF

Komentar hanya diterapkan perbaris dan dimulai dengan karakter khusus (‘). Jika ditemui karakter ini maka semua karakter yang mengikutinya hingga akhir baris diabaikan.

Baris yang tidak ada kalimat atau baris kosong merupakan baris yang hanya memiliki token **TOK_EOL** dan mungkin ada juga komentar. Jeda spasi (20_{16}) atau tabulasi (09_{16}) antara token dinamakan juga *whitespace*.

Identifier dan kata-kata yang dicadangkan (*reserved words*) digolongkan sebagai **nama**. Penerapan ini sangat menyederhanakan analisis leksikal karena tidak perlu membuat ekspresi reguler untuk setiap kata-kata yang dicadangkan.

Pengenalan **nama** dilakukan berdasarkan aturan ekspresi reguler untuk **nama**. Jika ditemukan **nama** yang sama dengan kata-kata yang dicadangkan maka token keluaran analisis ini adalah token dari kata-kata yang dicadangkan tersebut. Jika tidak ada **nama** yang sama maka **nama** tersebut diklasifikasikan sebagai token *identifier* (**TOK_ID**). Berikut adalah ekspresi reguler untuk **nama** :

nama → **huruf** (**huruf** | **digit** | _) *
huruf → A | B | … | Z | a | b | … | z
digit → 0 | … | 9

Token untuk konstanta kalimat adalah.

TOK_LITERAL → **“konstantakarakter**+**“**
konstantakarakter → semua karakter ASCII 8-bit kecuali “, CR, dan LF

Token bilangan ada tiga macam yang dikenali yaitu desimal, heksadesimal dan biner. Berikut adalah ekspresi regulernya.

TOK_NUM → **digit**+
TOK_NUMHEX → $(**digit**|**heksa**)+
TOK_NUMBIN → %(0|1)+
heksa → A | B | … | F | a | b | … | f

Karena sederhananya simbol operator maka sebagian besar memiliki ekspresi regulernya sendiri. Tabel 3.1 berisi ekspresi reguler untuk sebagian simbol operator dan simbol lainnya.

Tabel 3.1 Ekspresi reguler untuk sebagian token simbol.

TOK_EQU	→	=	**TOK_AIOR**	→	|
TOK_NEQ	→	<>	**TOK_AXOR**	→	^
TOK_GTH	→	>	**TOK_COM**	→	~
TOK_GEQ	→	>=	**TOK_LPAREN**	→	(
TOK_LTH	→	<	**TOK_RPAREN**	→	)
TOK_LEQ	→	<=	**TOK_LCLOSE**	→	[
TOK_ADD	→	+	**TOK_RCLOSE**	→	]
TOK_SUB	→	–	**TOK_ADDRESS**	→	@
TOK_MLL	→	*	**TOK_DOT**	→	.
TOK_MLH	→	**	**TOK_DOTDOT**	→	:
TOK_DIV	→	/	**TOK_COMMA**	→	,
TOK_AAND	→	&	**TOK_BCKSLSH**	→	\

b. SIMULATOR DFA

Simulator DFA dibuat dengan menggunakan algoritma simulator DFA. Sejumlah token memakai token **nama** sebagai awal penentuan jenis token kemudian memakai tabel simbol untuk mengkategorikan jenis token spesifik. Tabel 3.2 berisi kata-kata yang dicadangkan dari bahasa sisBASIC.

c. TATA BAHASA DAN SKEMA PENERJEMAHAN

Karena terbatasnya instruksi yang bisa didukung langsung oleh interpreter maka instruksi FOR … NEXT dibuat dengan menggunakan penerjemahan instruksi.

Tata bahasa sisBASIC dibuat untuk setiap instruksi. Setiap tata bahasa instruksi hanya untuk satu baris program sumber. Pemeriksaan akhir baris dilakukan dengan memeriksa token **TOK_EOL**. Skema penerjemahan juga dibuat untuk tiap tata bahasa.

Tabel 3.2 Kata-kata yang dicadangkan pada tabel simbol.

Token	Leksema	Flags	Token	Leksema
TOK_NOT	NOT		TOK_ BIT0	BIT0
TOK_AND	AND		TOK_ BIT1	BIT1
TOK_OR	OR		TOK_ BIT2	BIT2
TOK_XOR	XOR		TOK_ BIT3	BIT3
TOK_MOD	MOD		TOK_ BIT4	BIT4
TOK_MAX	MAX		TOK_ BIT5	BIT5
TOK_MIN	MIN		TOK_ BIT6	BIT6
TOK_DIG	DIG		TOK_ BIT7	BIT7
TOK_SHL	SHL		TOK_ BIT8	BIT8
TOK_SHR	SHR		TOK_ BIT9	BIT9
TOK_REV	REV		TOK_ BIT10	BIT10
TOK_ABS	ABS		TOK_ BIT11	BIT11
TOK_DCD	DCD		TOK_ BIT12	BIT12
TOK_NCD	NCD		TOK_ BIT13	BIT13
TOK_BYTE	BYTE		TOK_ BIT14	BIT14
TOK_WORD	WORD		TOK_ BIT15	BIT15
TOK_INS	INS	0	TOK_NEXT	NEXT
TOK_OUTS	OUTS	0	TOK_GOTO	GOTO
TOK_DIRS	DIRS	0	TOK_LET	LET
TOK_DIM	DIM		TOK_READ	READ
TOK_AS	AS		TOK_WRITE	WRITE
TOK_CONST	CONST		TOK_BUTTON	BUTTON
TOK_DATA	DATA		TOK_DEBUG	PRINT
TOK_GOSUB	GOSUB		TOK_SEROUT	SEROUT
TOK_RETURN	RETURN		TOK_DEBUGIN	SCAN
TOK_IF	IF		TOK_SERIN	SERIN
TOK_THEN	THEN		TOK_PAUSE	PAUSE
TOK_FOR	FOR		TOK_STOP	STOP
TOK_TO	TO		TOK_RANDOM	RANDOM
TOK_STEP	STEP		TOK_DEC	DEC
TOK_N1200	N1200		TOK_I1200	I1200
TOK_N2400	N2400		TOK_I2400	I2400
TOK_N4800	N4800		TOK_I4800	I4800
TOK_N9600	N9600		TOK_I9600	I9600

2. PENGURAI PREDIKTIF

Pengurai prediktif dibuat dengan konsep hampir sama dengan simulator DFA. Yang membedakan adalah masukannya berupa token bukan karakter dan pemrosesan dilakukan dalam subrutin terpisah supaya bisa melakukan penguraian secara rekursif.

Tata bahasa telah dibuat sedemikian rupa sehingga token awalnya unik supaya penentuan arah penguraian dapat ditetapkan dengan mudah sesuai konsep pengurai prediktif. Subrutin pengurai instruksi dibuat dengan aturan tata bahasa dan aksi yang dilakukan disesuaikan dengan skema translasinya.

Pada penguraian dibentuk deretan kalimat yang dipisahkan per baris dari aksi skema penerjemahan `emits`. Aksi semantik pada tahap ini hanya berkaitan dengan batasan nilai konstanta dan pemakaian variabel sesuai jenisnya. Pemeriksaan jenis variabel dapat dilakukan karena saat TO_ID dimasukan sebagai variabel saat deklarasi DIM jenisnya juga dimasukan dalam kolom *flags* dari tabel simbol (*symtable*).

Pemeriksaan semantik tahap kedua berhubungan dengan *backpatching* nama label dan penempatan data oleh instruksi DATA pada AT24C64. Saat tahap dua ini semua label sudah masuk dalam *symtable* sehingga semua loncatan yang memakai TOK_ID dan label bisa diperiksa. Jika ada TOK_ID yang belum dideklarasi sebagai label maka akan dibangkitkan pesan kesalahan. Instruksi DATA memperbolehkan data dimasukan dari program sumber ke alamat mana saja dari AT24C64. Tetapi sebagian lokasi sudah terpakai untuk kode antara hasil kompilasi yang tidak boleh ditumpuk. Jika ada data yang saling tertumpuk pada alamat yang sama maka akan dibangkitkan pesan kesalahan.

3. PENYUSUNAN KODE ANTARA

Pada tahap ini hasil penguraian analisis sintaks yang berupa deretan kata tertentu dengan kombinasi khusus diterjemahkan ke kode angka untuk kemudian dikirim ke sistem perangkat keras interpreter. Penyusunan kode antara dilakukan dalam tiga tahap.

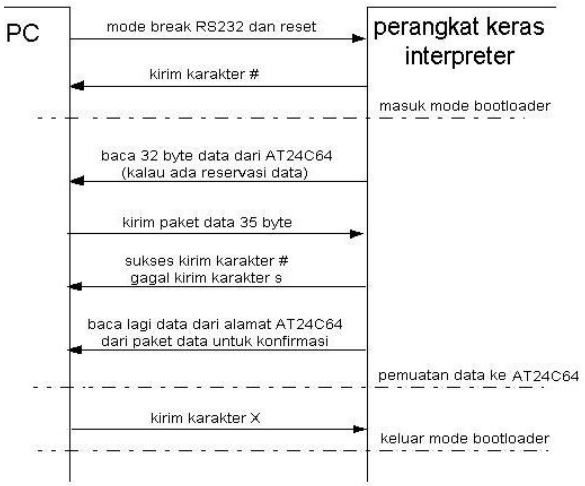
Tahap pertama dilakukan penerjemahan hasil analisis sintaks menjadi kode angka, perhitungan alamat absolut tiap label, pemeriksaan semantik tahap kedua.

Pada tahap kedua disisipkan 2-*byte* lokasi kode angka label yang telah dipersiapkan pada tahap satu, memetakan kode antara kedalam larik variabel *byte* berukuran 8192 (`bytemap`), menghitung posisi alamat awal kode antara dan mengisi larik variabel *byte* berukuran 8192 lainnya (`byteflag`) untuk menandai posisi *byte* kode antara. Alamat awal kode antara yang berukuran 2-*byte* ditempatkan pada AT24C64 alamat $1FFE_{16}$ untuk 1-*byte* MSB dan alamat $1FFF_{16}$ untuk 1-*byte* LSB.

Tahap ketiga menentukan alokasi data dari instruksi DATA dan membuat paket data sebanyak yang diperlukan.

4. PENGIRIMAN KODE ANTARA KE PERANGKAT KERAS INTERPRETER

Gambar 3.6 menunjukkan protokol pengiriman paket data dari PC ke perangkat keras interpreter.



Gambar 3.6 protokol pengiriman paket data.

Pertama-tama mode *break* RS232 diaktifkan dan sinyal reset dikirim. Setelah waktu tunda 30 milidetik maka mode *break* RS232 di nonaktifkan dan waktu tunda 30 milidetik untuk menunggu diterimanya karakter #. Kalau karakter # diterima berarti mode *bootloader* berhasil dimasuki.

Paket kode antara dikirim ke AT24C64 dan setelah waktu tunda 30 milidetik, diperiksa karakter yang diterima. Kalau karakter yang diterima adalah (**s**) maka pemeriksaan *checksum* gagal yang berarti terjadi gangguan saat pengiriman paket. Kalau karakter yang diterima (**#**) artinya paket diterima dan data telah ditulis ke AT24C64. Jika semua paket data berhasil dikirimkan kemudian dapat dikirim karakter (**X**) untuk keluar dari mode *bootloader*.

IV. PENGUJIAN DAN ANALISIS

Ada tiga aplikasi uji yang dibuat yaitu : tes operator, demo dadu acak, dan demo 'lampu berjalan'. Perangkat lunak standar komunikasi RS232 pada PC yang digunakan untuk tampilan adalah aplikasi HyperTerminal dari sistem operasi Windows 98.

HyperTerminal diatur parameter komunikasinya RS232 menjadi 4800 bps, data 8-bit, 1-bit stop. Kode sumber untuk uji coba sisBASIC ada pada Lampiran.

A. PENGUJIAN TES OPERATOR

Jika ditekan tombol 1 pada papan-ketik PC maka aplikasi tes operator dieksekusi kemudian saat tampil pesan untuk meminta angka uji. Setelah angka uji A dan B dimasukkan maka aplikasi langsung menghitung dengan menggunakkan operator yang diuji..

Tiga pengujian dilakukan terhadap fungsi seluruh operator yang didukung sisBASIC. Perbandingan dilakukan dengan aplikasi kalkulator bawaan sistem operasi Microsoft Windows 98.

Pengujian pertama untuk operator *unary*. Tabel 4.1 berisi hasil pengujian operator *unary*.

Tabel 4.1 Hasil pengujian operator *unary*.

Operator	A = 10	A = 12345	A = 42371	A = 63576
ABS A	00010	12345	23165	01960
~ A	65525	53190	23164	01959
– A	65526	53191	23165	01960
DCD A	01024	00512	00008	00256
NCD A	00003	00013	00015	00015

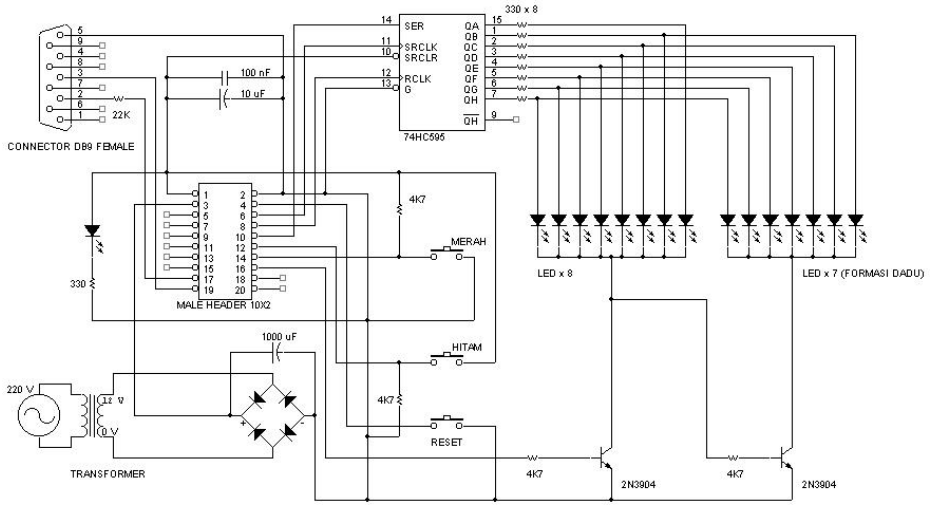
Perbandingan dengan hasil perhitungan operator *unary* menunjukkan kesesuaian hasil untuk bilangan masukan yang sama.

Pengujian kedua untuk operator *binary*. Hasil pengujian yang dibandingkan dengan hasil perhitungan operator *binary* menunjukkan kesesuaian hasil untuk bilangan masukan yang sama.

Pengujian ketiga untuk operator relasi dan logika. Hasil pengujian yang dibandingkan dengan hasil perhitungan operator relasi dan logika menunjukkan kesesuaian hasil untuk bilangan masukan yang sama.

B. PENGUJIAN DADU ACAK

Aplikasi kedua memakai instruksi **RANDOM** untuk membuat dadu acak elektronik. Tombol hitam dan merah bersebelahan dipakai untuk interaksi dengan penguji. Tombol hitam memakai konfigurasi aktif *high* dan tombol merah memakai konfigurasi aktif *low*. Tombol hitam dipakai untuk mengacak dadu dan tombol merah dipakai untuk kembali ke menu pemilihan aplikasi. Secara keseluruhan aplikasi dadu acak ini bekerja sesuai yang diprogramkan.



Gambar 4.1 Skematik rangkaian pengujian sisBASIC.

C. PENGUJIAN LAMPU BERJALAN

Aplikasi ketiga memakai instruksi **FOR … NEXT** untuk perulangan berhingga, dan operator **DCD** untuk membentuk tampilan pola LED 'berjalan'. Tombol hitam pada aplikasi ini memiliki fungsi untuk membuat variasi waktu tundaan untuk menyalakan LED. Tombol merah berfungsi sama seperti aplikasi kedua.

Perangkat lunaknya dibuat supaya LED menyala bergantian satu per satu dari posisi LSB ke MSB berbalik lagi ke LSB dan berulang terus-menerus. LED menyala pertama kali dengan waktu tundaan tercepat. Setiap kali tombol hitam ditekan maka waktu tundaannya akan meningkat sehingga pergantian nyala LED terlihat melambat. Jika waktu tunda terus ditingkatkan sampai maksimum maka penambahan berikutnya akan menyebabkan waktu tundaan berbalik lagi menjadi tercepat. Secara keseluruhan aplikasi dadu acak ini bekerja sesuai yang diprogramkan.

V. PENUTUP

A. KESIMPULAN

Berdasarkan hasil data yang diperoleh penulis melalui pengujian dan pengamatan, penulis dapat menarik kesimpulan sebagai berikut.

1. Format komunikasi yang tidak didukung oleh perangkat keras PIC16F84A dapat diterapkan secara perangkat lunak meskipun memiliki keterbatasan, dalam hal kecepatan pertukaran data yang lambat, fungsi lainnya terhenti ketika komunikasi sedang berlangsung, dan sebagainya.
2. Terapan interpreter dalam memori program PIC16F84A yang berukuran 1 *kiloword* cukup memadai meskipun banyak aturan khusus yang dibuat untuk membatasi kemampuan interpreter.
3. Operator perkalian dan pembagian yang tidak didukung oleh bahasa rakitan PIC16F84A bisa dilakukan dengan kombinasi operasi dasar seperti penjumlahan, pengurangan dan penggeseran bit.
4. Instruksi **RANDOM** pada aplikasi uji dadu acak hanya mampu menghasilkan bilangan acak jika diberi nilai masukan yang berbeda-beda.
5. Kombinasi instruksi **FOR … NEXT** dan operator **DCD** dipakai pada aplikasi uji lampu jalan untuk menghasilkan pola lampu jalan. Sisipan instruksi **PAUSE** untuk waktu tundaan penyalaan LED mempengaruhi respon pendeteksian tombol oleh instruksi **BUTTON**.

B. SARAN

Saran-saran untuk pengembangan Tugas Akhir ini antara lain adalah :

1. Supaya dukungan instruksi lebih banyak pada interpreter mikrokontroler maka dapat digunakan mikrokontroler dengan memori program yang lebih besar, misalnya : AT89C4051 dari ATMEL yang memiliki memori program 4 *kilobyte*.
2. Kecepatan interpretasi dapat ditingkatkan dengan memakai mikrokontroler yang mampu memakai kristal berkecepatan tinggi, seperti mikrokontroler SCENIX SX28/DP dari UBICOM yang dengan mode turbo mampu mengeksekusi rata-rata 1 *clock* per 1 bahasa rakitan. Dengan memakai kristal sampai 75MHz, waktu eksekusi instruksi tercepat 13.3 nanodetik.

DAFTAR PUSTAKA

[1] Aho, A.V., R. Sethi, and J.D. Ullman, *Compilers, principles, techniques, and tools*, Addison-Wesley, Reading, 1988.

[2] Edwards, S., *The PIC Source Book*, http://www.dontronics.com/see.html, 2004.

[3] Palacherla, A., *Application Note 526 PIC16C5X / PIC16CXXX Math Utility Routines*, Microchip Technology, http://www.microchip.com, 1997.

[4] Palacherla, A., *Application Note 544 Math Utility Routines*, Microchip Technology, http://www.microchip.com, 1997.

[5] Slamet, S. dan H. Suhartanto, *Teknik Kompilasi*, Elex Media Komputindo, Jakarta, 1992

[6] Tremblay, J.P. and P.G. Sorenson, *The Theory and Practice of Compiler Writing*, McGraw-Hill, Singapore, 1985.

[7] --------, *+5V-Powered, Multichannel RS-232 Drivers/Receivers MAX220–MAX249*, Maxim Integrated Products, http://www.maxim-ic.com, 1999.

[8] --------, *2 Wire Serial EEPROM AT24C64*, Atmel Corporation, http://www.atmel.com, 2003.

[9] --------, *Application Note Interfacing AT24CXX Serial EEPROMs with AT89CX051 Microcontrollers*, Atmel Corporation, http://www.atmel.com, 2001.

[10] --------, *PIC16F84A 18-pin Enhanced Flash/EEPROM 8-Bit Microcontroller*, Microchip Technology, http://www.microchip.com, 1998.

Santoso (NIM : L2F097672)
Mahasiswa Jurusan Teknik Elektro Universitas Diponegoro Semarang, Konsentrasi di Bidang Kontrol. Angkatan'97

Mengetahui,

Pembimbing II

R. Rizal Isnanto, S.T., M.M., M.T
NIP. 132 288 515

LAMPIRAN
SENARAI KODE SUMBER SISBASIC UNTUK PENGUJIAN

```
const PATTERN = 10
const seripin = 6
const seropin = 7
dim a as word
dim b as word
dim i as byte
dim j as byte
dim tmp as byte
dim rand as word

'************************************************************
' BAGIAN RUTIN UTAMA
'************************************************************
' inisialisasi custom user data
data @PATTERN,%00010000   ' dadu angka satu
data %01000100            ' dadu angka dua
data %00110010            ' dadu angka tiga
data %10101010            ' dadu angka empat
data %10111010            ' dadu angka lima
data %11101110            ' dadu angka enam
'************************************************************
' tampil pada jendela test perangkat lunak kompiler di PC
'************************************************************
' inisialisasi perangkat keras
print ["sedang inisialisasi..."]
outs = 0
tmp = $FF
gosub KIRIMBYTE           ' SIPO diinisialisasi = 0
print ["tekan sembarang tombol keyboard PC untuk lanjut"]
scan [tmp]                ' program berhenti sampai ada masukan
'************************************************************
' tampil pada sembarang perangkat lunak terminal di PC
'************************************************************
mainmenu:
   serout seropin,i4800,[13,10,"1. tes operator",13,10]
   serout seropin,i4800,["2. demo dadu acak",13,10]
   serout seropin,i4800,["3. demo 'lampu berjalan'",13,10]
   serout seropin,i4800,["4. mengakhiri program",13,10]

masukan:
   serout seropin,i4800,["pilihan(1-4)..."]
   serin seripin,i4800,[tmp]
   if tmp = $31 then tes_operator
   if tmp = $32 then dadu_acak
   if tmp = $33 then lampu_jalan
   if tmp = $34 then selesai
   serout seropin,i4800,[13,"pilihan '",tmp,"' salah, "]
   goto masukan
selesai:
   read 0,tmp
   print [13,10,"data terakhir = ",dec tmp]
   print [13,10,"masukan data baru (0-255)..."]
   scan [dec tmp]
   write 0,tmp
   stop
'************************************************************
' PILIHAN TES OPERATOR
'************************************************************
tes_operator:
   serout seropin,i4800,[13,10,"Tes : A operator B = ..."]
   serout seropin,i4800,[13,10,"masukan nilai A, B ... "]
   serin seripin,i4800,[dec a,dec b]
   serout seropin,i4800,[13,10,"nilai A = ",dec a]
   serout seropin,i4800,[13,10,"nilai B = ",dec b]
   serout seropin,i4800,[13,10," 1. ABS A   = ",dec abs a,13,10]
   serout seropin,i4800,[" 2. ~ A     = ",dec ~ a,13,10]
   serout seropin,i4800,[" 3. - A     = ",dec - a,13,10]
   serout seropin,i4800,[" 4. DCD A   = ",dec dcd a,13,10]
   serout seropin,i4800,[" 5. NCD A   = ",dec ncd a,13,10]
   serout seropin,i4800,[" 6. A + B   = ",dec a + b,13,10]
   serout seropin,i4800,[" 7. A - B   = ",dec a - b,13,10]
   serout seropin,i4800,[" 8. A * B   = ",dec a * b,13,10]
   serout seropin,i4800,[" 9. A ** B  = ",dec a ** b,13,10]
   serout seropin,i4800,["10. A / B   = ",dec a / b,13,10]
   serout seropin,i4800,["11. A MOD B = ",dec a mod b,13,10]
   serout seropin,i4800,["12. A MIN B = ",dec a min b,13,10]
   serout seropin,i4800,["13. A MAX B = ",dec a max b,13,10]
   serout seropin,i4800,["14. A DIG B = ",dec a dig b,13,10]
   serout seropin,i4800,["15. A SHL B = ",dec a shl b,13,10]
   serout seropin,i4800,["16. A SHR B = ",dec a shr b,13,10]
   serout seropin,i4800,["17. A REV B = ",dec a rev b,13,10]
   serout seropin,i4800,["18. A & B   = ",dec a & b,13,10]
   serout seropin,i4800,["19. A | B   = ",dec a | b,13,10]
   serout seropin,i4800,["20. A ^ B   = ",dec a ^ b,13,10]
   serout seropin,i4800,["21. A = B   = ",dec a = b,13,10]
   serout seropin,i4800,["22. A <> B  = ",dec a <> b,13,10]
```

```
   serout seropin,i4800,["23. A > B   = ",dec a > b,13,10]
   serout seropin,i4800,["24. A >= B  = ",dec a >= b,13,10]
   serout seropin,i4800,["25. A < B   = ",dec a < b,13,10]
   serout seropin,i4800,["26. A <= B  = ",dec a <= b,13,10]
   serout seropin,i4800,["26. NOT A   = ",dec not a,13,10]
askagain:
   serout seropin,i4800,[13,10,"ulangi tes operator lagi ? (Y/N)",13,10]
   serin seripin,i4800,[tmp]
   tmp.bit5 = 0                    ' huruf kecil -> huruf besar
   if tmp = $4E then mainmenu
   if tmp = $59 then tes_operator
   goto askagain
'*************************************************************
' PILIHAN DADU ACAK
'*************************************************************
dadu_acak:
   rand = $3045
   ask_dice:
   serout seropin,i4800,[13,10,"tekan tombol hitam untuk acak dadu"]
   serout seropin,i4800,[13,10,"tekan tombol merah untuk ke menu utama"]
rescan_dice:
   button 3,1,1,acak_dadu
   button 4,0,0,rescan_dice
   goto mainmenu
acak_dadu:
   random rand
   tmp = rand & $07
   if tmp > 5 then acak_dadu
   read PATTERN+tmp,j
   tmp = j
   gosub KIRIMBYTE
   goto ask_dice
'*************************************************************
' PILIHAN 'LAMPU BERJALAN'
'*************************************************************
lampu_jalan:
   serout seropin,i4800,[13,10,"tekan tombol hitam untuk ubah tundaan"]
   serout seropin,i4800,[13,10,"tekan tombol merah untuk ke menu utama"]
   dirs.bit5 = 0
   outs.bit5 = 1
   j = 0
   rand = 0
   gosub jalan_lampu_ganti_delay
jalan_maju:
   for j = 0 to 7
      tmp = dcd j
      gosub kirimbyte
      button 4,0,1,mainmenu
      button 3,1,0,skip_jalan_maju
      gosub jalan_lampu_ganti_delay
skip_jalan_maju:
      pause rand
   next
   for j = 6 to 1 step -1
      tmp = dcd j
      gosub KIRIMBYTE
      button 4,0,1,mainmenu
      button 3,1,0,skip_jalan_mundur
      gosub jalan_lampu_ganti_delay
skip_jalan_mundur:
      pause rand
   next
goto jalan_maju
'*************************************************************
' BAGIAN SUBRUTIN
'*************************************************************
jalan_lampu_ganti_delay:
   rand = rand + 100
   if rand < 600 then jalan_lampu_ganti_delay_keluar
   rand = 100
jalan_lampu_ganti_delay_keluar:
   serout seropin,i4800,[13,10,"delay = ",dec rand," ms"]
   return

KIRIMBYTE:
   dirs = dirs & $F8      ' pin 0,1,2 = output
   for i = 0 to 7
      outs.bit2 = tmp.bit7        ' ambil bit data utk dikirim
      outs.bit0 = 1       ' clock up
      outs.bit0 = 0       ' clock down
      tmp = tmp shl 1     ' shift next bit
   next
   outs.bit1 = 1          ' transfer ke latch
   outs.bit1 = 0
   return
```