# BAB 5

# BAHASA DAN PENGEMBANGAN PROGRAM MIKROPROSESOR

Ciri penting dari sebuah mikroprosesor adalah sifatnya yang *programmable*. Artinya sebuah mikroprosesor tidak dapat bekerja begitu saja secara perangkat keras tanpa perangkat lunak. Sebuah mikroprosesor selalu bekerja dengan program. Program adalah susunan sejumlah instruksi yang membentuk satu fungsi. Untuk mengembangkan program dalam mikroprosesor digunakan bahasa pemrograman. Bagaimana mengembangkan program mikroprosesor menjadi bagian penting dari isi buku ini.
Kata kunci: *program, bahasa pemrograman*

Kebanyakan orang berpikir bahwa komputer adalah sebuah peralatan yang sangat komplek, sulit dipelajari, dan dapat berpikir melebihi manusia. Pernyataan ini terlalu berlebihan, karena komputer hanya dapat bekerja jika ada perintah yang diberikan yang disebut **program.** Program adalah susunan atau urutan perintah-perintah sederhana yang diberikan kepada komputer untuk memecahkan beberapa permasalahan. Jika sebuah program telah ditulis dan dilakukan *debuging*, komputer akan dapat mengeksekusi program tersebut dengan sangat cepat dan dengan cara yang sama setiap saat tanpa kesalahan.

Disinilah kehebatan komputer, meskipun program tersusun dari perintah-perintah yang sangat sederhana, hasil akhir sangat menakjubkan sebab komputer dapat bekerja dengan kecepatan tinggi.

Kebanyakan mikroprosesor memiliki kesamaan dalam perintah atau instruksi. Intsruksi transfer data misalnya dapat menggunakan perintah LOAD atau MOVE, Instruksi matematika sederhana menggunakan perintah ADD, SUBTRACT, MULTIPLY, DEVIDE.

## 1. Bahasa Pemrograman

Untuk menjalankan program, sebuah mikrokomputer harus memiliki program yang tersimpan dalam format biner pada suatu lokasi memori. Ada tiga tingkat level bahasa yang dapat digunakan untuk menulis program pada sebuah mikrokomputer yaitu:

- **Bahasa MESIN**
- **Bahasa ASSEMBLY**
- **Bahasa Aras Tinggi (*High Level*)**

## 1.1. Bahasa Mesin

Bahasa mesin adalah bahasa dalam bentuk kode-kode biner sebagai sandi operasi (*operation code*) dari sebuah mikroprosesor. Bahasa mesin adalah bahasa yang langsung berhubungan dengan mikroprosesor yang ditulis dan dikembangkan dari set instruksi. Tanpa bantuan set instruksi bahasa mesin sangat sulit dimengerti atau difahami. Untuk dapat menulis bahasa mesin maka penguasaan set instruksi sebuah mikroprosesor adalah wajib.

## 1.2. Bahasa Assembly

Untuk membuat proses pemrograman menjadi lebih mudah, kebanyakan programmer menuliskan program dalam bentuk bahasa assembly. Kemudian mereka menterjemahkan bahasa assembly yang ditulisnya menjadi bahasa mesin sehingga dapat di *download* ke memori dan di run atau dijalankan. Penterjemahan bahasa assembly menjadi kode biner bahasa mesin dapat dilakukan secara manual atau menggunakan program yang disebut dengan *assembler*.

Bahasa assembly menggunakan sejumlah mnemonik untuk merepresentasi kan instruksi–instruksi. Mnemonik adalah singkatan dari suatu perintah atau instruksi sebagai piranti untuk membantu ingatan. Sebagai contoh :

| | | |
|---|---|---|
| Load | disingkat | LD |
| Add | → | ADD |
| Add With Carry | → | ADC |
| Subtract | → | SUB |
| Subtract With Carry | → | SBC |
| Complement | → | CPL |

Pernyataan bahasa assembly biasanya ditulis dalam bentuk standar seperti pola Gambar 5.1 di bawah.

| Label | Mnemonik | Operand | Komentar |
|---|---|---|---|
| Mulai: | LD | A, 3F | Isi Register A dengan data 3Fh |
| | LD | B, 5D | Isi Register B dengan data 5Dh |
| | ADD | A,B | Jumlahkan data A dengan data B |

Gambar 5.1. Format program assembly

Dari Gambar 5.1. terlihat Label adalah simbol atau kelompok simbol yang digunakan untuk merepresentasikan alamat yang tidak diketahui secara spesifik pada saat pernyataan-pernyataan ditulis. Label tidak dipersyaratkan dalam setiap pernyataan. Label dimasukkan bila diperlukan saja.

Mnemonik adalah kolom singkatan dari setiap perintah bahasa assembly. Bagian operand dapat memuat nama Register, alamat memori, alamat port, atau data

immediate dari sebuah instruksi. Operand adalah sasaran dari instruksi. Pada bagian operand terbagi menjadi dua bagian yaitu sumber data yang disebut dengan source dan tujuan data atau destinasi. Pada umumnya source ada disebelah kanan dari destinasi. Bagian komentar biasanya digunakan untuk memberi penjelasan singkat maksud atau sasaran dari instruksi disebelah kirinya.

Jika kita sudah menulis program dalam bahasa assembly, pertanyaannya bagaimana menterjemahkan menjadi bahasa mesin yang siap dieksekusi ke sistem mikroprosesor. Ada dua jawaban atas pertanyaan ini. Pertama secara manual dengan menerjemahkan setiap instruksi bahasa assembly menjadi bahasa mesin melalui set instruksi. Kemudian kedua secara otomatis dengan menggunakan *ASSEMBLER.* Assembler adalah sebuah program yang dapat dijalankan pada sebuah mikrokomputer atau pada sebuah *Microcomputer Development System.*

## 1.3. Bahasa Aras Tinggi

Cara lain dalam menulis program pada mikrokomputer adalah dengan bahasa aras tinggi seperti Bahasa C, bahasa Pascal, BASIC, FORTRAN, dan sebagainya. Secara umum bahasa aras tinggi dapat dikelompokkan menjadi dua yaitu sebagai ***Compiler*** **atau** ***Interpreter****.*

## 1.4. Tools Pengembangan Program Bahasa Assembly

Sesederhana apapun dari sebuah program dalam bahasa assembly memerlukan penggunaan *Computer Development System* dan *Program Development Tools (contoh : MPF-1, Guru Mikro Saya, MidiCom).* Sistem tersebut biasanya dilengkapi beberapa kilo byte memori RAM/RWM, Keyboard, Display, Floppy Disk, Hard Disk, dan Emulator. Untuk mengembangkan program dalam bahasa assembly diperlukan tool sebagai berikut:

## 1.5. Editor

Editor adalah sebuah program yang digunakan untuk mengetik atau mengedit program dalam bentuk pernyataan-pernyataan. Contoh program editor adalah TV Demo, EDLIN, ALTER. Fungsi utama program editor adalah untuk membantu programmer dalam membangun program dalam bahasa assembly dalam bentuk format yang benar sehingga assembler dapat menterjemahkan menjadi bahasa mesin yang benar. Hasil atau bentuk program hasil dari editor disebut ***Source Program.***

## 1.6. Assembler

Assembler adalah program yang digunakan untuk menterjemahkan mnemonik bahasa assembly menjadi kode biner yang benar untuk setiap instruksinya.

Assembler akan membaca File Source Program di disk dimana source program tersebut disimpan setelah diedit menggunakan editor. Assembler biasanya membaca file source program lebih dari sekali. Pertama untuk menentukan penggantian nama item data dan offset dari sebuah label, dan meletakkan informasi tersebut dalam sebuah *Symbol Table*. Kemudian yang kedua assembler membangkitkan kode biner (Op-code) untuk setiap instruksi.

Assembler membangkitkan dua file yaitu ***Object File*** yang berisi kode biner untuk instruksi-instruksi dan informasi tentang alamat instruksi. File yang kedua adalah ***Assembler List File*** yang memuat pernyataan bahasa assembly, kode biner untuk setiap instruksi, dan offset untuk setiap instruksi. File yang kedua ini yang biasanya di cetak ke printer sebagai pegangan pada saat melakukan testing dan troubleshooting.

## 1.7. Lingker

Lingker adalah program yang digunakan untuk menggabungkan bebebrapa object file menjadi satu object file yang lebih besar. Biasanya dalam membangun program yang sangat besar agar lebih efesient program tersebut dipecah dan dibagi-bagi menjadi beberapa program yang lebih kecil yang disebut dengan ***Modul***. Setiap modul dapat ditulis, ditest, dan di debugged secara individual. Jika semua modul-modul telah berfungsi dengan baik, selanjutnya dapat di *lingked* menjadi program dalam fungsinya yang besar.

Lingker menghasilkan ***Link file*** yang berisi kode biner untuk semua modul-modul yang dikombinasikan. Lingker juga menghasilkan ***Link map file*** yang berisi informasi alamat tentang file-file yang digabungkan.

Lingker tidak menetapkan alamat absolut pada program, ia hanya menetapkan alamat relatif mulai nol. Bentuk dari program ini dikatakan sebagai *relocateable* = dapat direlokasi, sebab dapat ditempat disetiap tempat di memori untuk dijalankan.

## 1.8. Lokator

Lokator adalah program yang digunakan untuk menetapkan alamat spesifik dimana object code diletakkan dalam memori. Lokator pada DOS IBM PC disebut EXE2BIN.

## 1.9. Debugger

Jika program kita tidak mempersyaratkan adanya perangkat keras luar atau hanya perangkat keras internal saja yang diakses secara langsung, kita membutuhkan debugger untuk menjalankan program tersebut. Debugger adalah sebuah program yang dapat digunakan untuk men-load program dalam bentuk object code pada sistem memori, mengesekusi, dan

melakukan troubleshooting atau *debug.* Dalam hal ini melakukan pelacakan ada tidaknya kesalahan dalam sebuah program.

Debugger menyediakan fasilitas untuk melihat isi register dan isi lokasi memori setelah sebuah instruksi dari program dijalankan. Disamping juga ada fasilitas untuk mengganti isi register dan data suatu lokasi memori.

### 1.10. Emulator

Cara lain untuk menjalankan sebuah program adalah menggunakan emulator. Emulator adalah gabungan diantara hardware dan software. Emulator biasanya digunakan untuk mengetes dan debug hardware dan software dari sebuah sistem eksternal seperti prototype dari sebuah instrumen berbasis mikroprosesor.

## 2. Langkah-Langkah Pengembangan Program

Menurut Douglas ada empat langkah yang harus diperhatikan dan dilakukan dalam mengembangkan sebuah program komputer. Keempat langkah itu adalah:

a. **Pendefinisian permasalahan.**
b. **Representasi kerja program.**
c. **Penemuan instruksi-instruksi yang benar, dan**
d. **Penulisan program.**

Jika ingin memiliki kemajuan yang baik dalam membuat program, maka ikuti dengan teratur empat langkah ini.

### 2.1. Pendefinisian Permasalahan

Langkah pertama yang harus dilakukan dalam menulis program adalah memikirkan dan mendefinsikan secara cermat permasalahan yang ingin diselesaikan menggunakan program komputer. Dengan kata lain apakah sesungguhnya yang ingin dikerjakan oleh sebuah program. Jika anda telah melakukan identifikasi permasalahan dan mendefinisikan permasalahan dengan jelas dan benar maka ini merupakan langkah awal yang sangat baik dalam menulis apa yang diinginkan dalam pembuatan program. Mari kita lihat satu contoh permasalahan berikut ini:

> "Menyeberang di jalan yang sangat ramai "

### 2.2. Algoritma

Contoh kasus menyeberang jalan sudah pasti contoh kasus yang biasa-biasa saja. Namun pengalaman dalam pelatihan dan pembelajaran ternyata contoh kasus ini tidak bisa diselesaikan dengan benar. Ada sejumlah hambatan antara lain: kebanyakan siswa terjebak dengan berpikir di level tinggi sehingga tidak bisa menurunkan langkah-langkah low level. Untuk memecahkan permasalahan bagaimana menyeberang di jalan yang sangat ramai diperlukan langkah-langkah atau sekuen atau formula kerja. Formula kerja yang digunakan untuk

memecahkan masalah pemrograman disebut ***Algoritma program***. Seorang programmer harus menggunakan daftar urutan pekerjaan. Dalam kasus permasalahan menyeberang jalan step perintah-perintah sederhana dapat dinyatakan seperti Gambar 5.2. Setelah anda membaca urutan step-step perintah pada Gambar 5.2. apa komentar anda ?

Perintah dalam program mengalir dari step awal (step 1) yaitu start ke step 2, step 3 dan seterusnya sampai dengan terminal stop. Kecuali ada perintah untuk melompat atau memanggil subrutin atau jika ada interupsi program akan mengalir terus kelangkah-langkah berikutnya. Pada step 3, step 4, dan step 5 terjadi proses pendeteksian dan pengambilan keputusan melangkah ke langkah 6 yaitu untuk memperhatikan apakah ada kendaraan yang lewat dari arah sebelah kiri. Pada step 6, step 7, dan step 8 juga terjadi proses pendeteksian dan pengambilan keputusan apakah melangkah step 9 untuk melihat ke arah kanan atau masih tetap melihat arah kiri pada step 6. Demikian juga pada step 9, step 10, dan step 11 juga terjadi proses pendeteksian dan pengambilan keputusan apakah melangkah step 12 untuk melangkah menyeberang atau masih tetap melihat arah kanan pada step 9.

Jika kesebelas langkah ini dilakukan maka anda akan selamat dan sukses menyeberang jalan. Masalahnya maukah anda berpikir dan bertindak secara disiplin seperti algoritma Gambar 5.2. Merubah cara berpikir dari level tinggi menjadi cara berpikir level rendah seperti bahasa bayi merupakan hambatan besar bagi programmer assembly pemula. Jika bisa merubah pola berfikir dari level tinggi ke level rendah maka niscaya anda akan bisa memprogram mikroprosesor dengan baik. Jika tidak disinilah hambatan saudara belajar memprogram. Cobalah kembali cermati dan baca untuk kedua kali step-step Gambar 5.2. Karena akan memberi gambaran yang baik mengapa komputer bisa bekerja dengan hasil yang baik. Banyak pemula menjabarkan step demi step langkah menyeberang jalan yang tidak operasional. Misalnya lihat kiri dan kanan yang tidak bisa dilakukan serentak.

| STEP | PERINTAH |
|---|---|
| **1.** | Start |
| **2.** | Berjalanlah ke sudut jalan dan berhenti |
| **3.** | Lihat dan cermati lampu pengatur lalu lintas |
| **4.** | Apakah lampu penyeberangan pada arah anda menyala merah |
| **5.** | Jika “ya”, kembali ke Step 3 (Untuk keadaan lain teruskan ke Step 6) |
| **6.** | Lihat ke arah kiri |
| **7.** | Apakah masih ada kendaraan yang lewat |
| **8.** | Jika “ya”, kembali ke Step 6 |
| **9.** | Lihat ke arah kanan |
| **10.** | Apakah masih ada kendaraan yang lewat |
| **11.** | Jika “ya”, kembali ke Step 9 |
| **12.** | Menyeberanglah dengan hati-hati |
| **13.** | Stop |

Gambar 5.2. Algoritma Menyeberang Jalan

Kesebelas langkah ini adalah bahasa bayi atau bahasa aras rendah atau bahasa level rendah, yang pada kenyataannya dilakukan langkah-langkah inilah yang dilakukan pada setiap menyeberang jalan dijalan yang sibuk dan ada lampu mengatur lalu lintas. Tidak akan pernah terjadi melihat lampu penyeberangan, melihat arah kiri,dan melihat arah kanan dilakukan secara serentak. Memang nampaknya seakan-akan demikian. Namun sesungguhnya tidaklah pernah demikian karena proses yang terjadi pasti merupakan sekuen langkah demi langkah seperti Gambar 5.2. Kesebelas sekuen step perintah ini disebut juga dengan Algoritma Program.

Semakin detail algoritma suatu program semakin baik karena program menghendaki kesuksesan 100% sesuai kebutuhan tuntutan permasalahan. Bisa saja step 5, step 6 dan step 7 tidak perlu ada manakala jalur lalu lintas hanya satu arah. Sangat berbahaya jika step 2, step 3, dan step 4 diabaikan. Jika ingin menyeberang jalan dan algoritma Gambar 5.2. digunakan dapat dipastikan penyeberang akan sukses dan selamat.

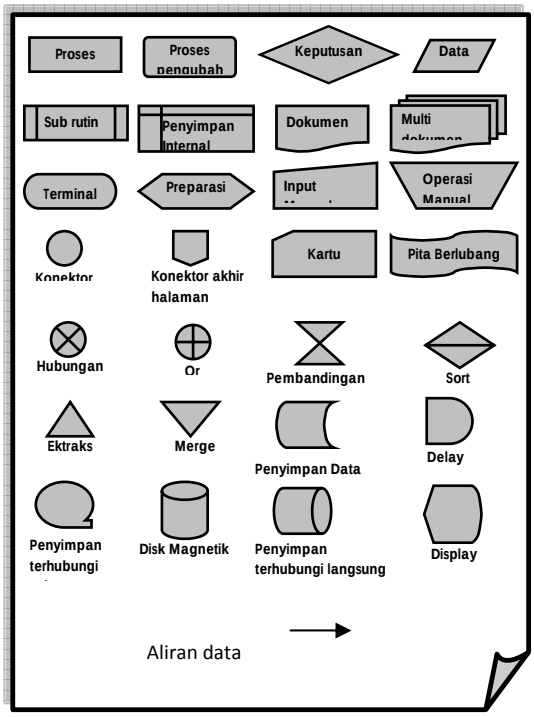
Kasus menyeberang di jalan yang sangat ramai dengan 11 step penyelesaian merupakan bentuk bahasa aras rendah. Mikroprosesor di dalam sistem bekerja dengan bahasa aras rendah. Agar anda sukses dan mudah belajar memprogram mikroprosesor maka anda harus memiliki kesiapan merubah kebiasaan berfikir pada aras tinggi kekebiasaan berfikir di aras rendah. Inilah kendala utama dalam belajar memprogram mikroprosesor.

## 2.3. Flowchart

Flowchart atau diagram alir digunakan untuk menunjukkan aliran proses sebuah program. Untuk menyajikan jenis operasi sebuah program digunakan bentuk-bentuk grafis standar. Ada dua puluh delapan jenis bentuk grafis yang digunakan untuk menyusun flowchart seperti pada Gambar 5.3. Bentuk-bentuk grafis penyusun flowchart dapat dilihat pada AutoShapes Flowchart program Microsoft Word. Pemilihan bentuk-bentuk grafis flowchart pada Gambar 5.3 tidak boleh sembarangan atau asal pilih apalagi membuat sendiri bentuk-bentuk lain secara bebas. Hal ini tidak dibenarkan karena semua bentuk grafis flowchart telah disepakati dan distandarkan secara internasional sebagai alat komunikasi.

Cermati satu per satu bentuk-bentuk grafis Flowchart Gambar 5.3. Fahami bentuk dan fungsinya masing-masing. Jika ada yang tidak jelas atau kurang bisa memahami bertanyalah kepada fasilitator atau guru anda disekolah.

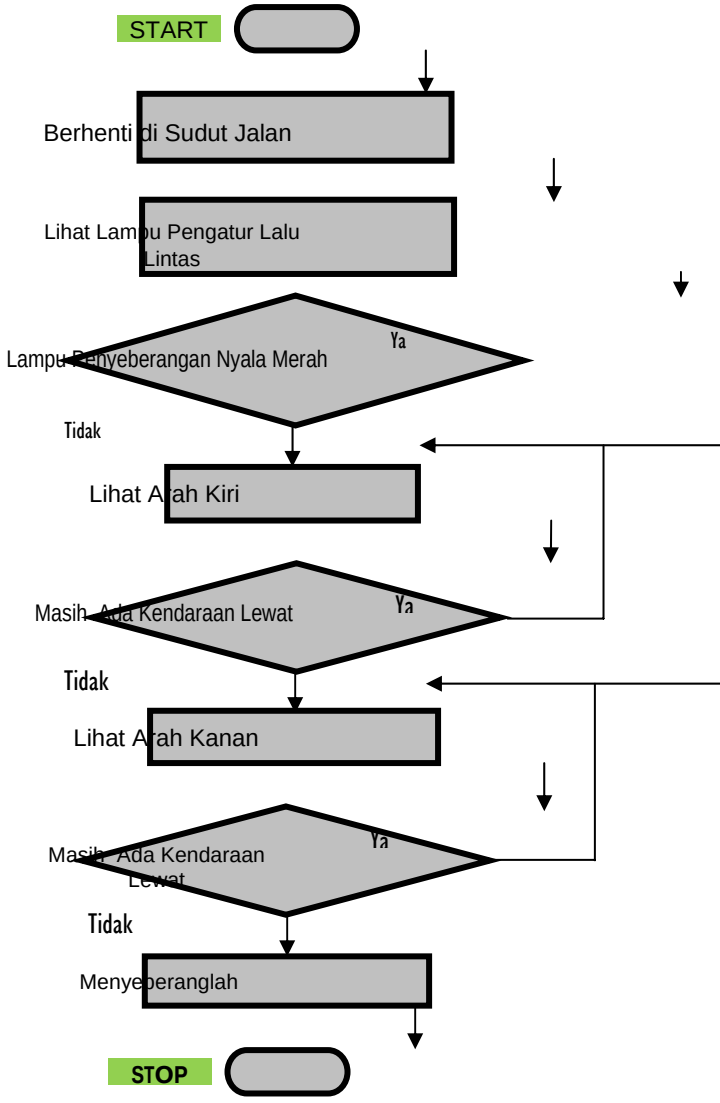
Berlatih dan berlatih diikuti dengan mengkonsultasikan kebenaran hasil latihan anda kepada fasilitator atau guru akan sangat menentukan kecepatan terbentuknya kompetensi pada diri anda.




Gambar 5.3. Bagan flowchart

Dua puluh delapan bentuk-bentuk grafis Flowchart pada Gambar 5.3 dapat dipilih salah satu sesuai jenis step perintah pada algoritma. Dari sebelas step algoritma program pada Gambar 5.2 dapat disusun sebuah flowchart sesuai kasus program atau berdasarkan algoritma yang telah disusun. Flowchart kasus menyeberang di jalan yang sangat ramai ditunjukkan seperti Gambar 5.4 disamping.

Flowchart menyeberang di jalan ramai pada Gambar 5.4 diawali dengan perintah START sebagai tanda mulai. Perintah start sebagai tanda mulai menggunakan bentuk grafis terminal. Selanjutnya diteruskan dengan dua proses ”berhenti di sudut jalan” dan ”melihat lampu pengatur lalu lintas”.




Gambar 5.4. Flowchart Menyeberang di Jalan Ramai

Kedua proses tersebut menggunakan bentuk grafis segi empat karena keduanya adalah langkah dalam bentuk proses. Pengambilan keputusan dengan pertanyaan apakah ”Lampu Penyeberangan Nyala Merah” menggunakan bentuk grafis belah ketupat. Proses pengambilan keputusan dengan pertanyaan apakah ”Lampu

Penyeberangan Nyala Merah" bermakna jika "ya" kembali atau tetap melihat lampu pengatur lalu lintas. Sebaliknya jika "tidak" menyala merah berarti menyala hijau mengandung pengertian bisa meneruskan ke step berikutnya yaitu proses melihat arah kiri.

Pertanyaan apakah masih ada kendaraan yang lewat merupakan step pengambilan keputusan. Jika ya harus tetap melihat arah kiri dan jika tidak ada kendaraan yang lewat bisa meneruskan ke step lihat arah kanan. Pertanyaan apakah masih ada kendaraan yang lewat dari arah kanan merupakan step pengambilan keputusan. Sampai pada proses menyeberang jika tidak lagi ada kendaraan yang lewat. Flowchart diakhiri dengan terminal STOP. Step melihat arah kiri dan melihat arah akan dalam kasus ini bisa saja ditukarkan posisinya. Jika masih ragu dengan bentuk grafis terminal lihat kembali bentuk-bentuk grafis flowchart Gambar 5.3.

Flowchart sebagai bentuk lain dari algoritma program memberi gambaran yang lebih jelas bisa diamati aliran proses yang terjadi. Flowchart memberikan kemudahan untuk menganalisis proses sehingga sering digunakan untuk menggambarkan aliran suatu proses.

Dua langkah selanjutnya dalam proses pembuatan program yaitu penemuan jenis instruksi yang benar dan tepat lalu diikuti dengan penulisan program. Memilih instruksi yang tepat dan benar merupakan kompetensi operasional agar program bisa berjalan dan benar. Memilih instruksi yang benar dan tepat jelas berkaitan dengan kemampuan penguasaan set instruksi yang baik. Dikatakan demikian karena tidak mungkin dapat memilih jika yang akan dipilih tidak dikenali, dimenegerti atau difahami.

Mengembangkan program komputer agar efektif harus mengikuti empat langkah sesuai saran atau cara Douglas. Sebelum sampai pada penulisan program pertama definisikan permasalahan program secara jelas lalu representasikan kerja program dengan algoritma dan flowchart. Yakinkan algoritma dan flowchart anda benar dan sesuai tuntutan permasalahan program. Kemudian baru menulis program dengan memanfaatkan instruksi-instruksi yang benar dan tepat.

Untuk membangun kompetensi pemrograman anda harus banyak berlatih. Semakin banyak melakukan latihan dengan berbagai kasus masalah pemrograman semakin baik. Dengan melakukan secara sungguh-sungguh mulai dari kasus yang sederhana sampai pada kasus yang komplek akan memberi peningkatan penguasaan kompetensi pemrograman.

Contoh kasus berikut yang dapat digunakan untuk berlatih misalnya

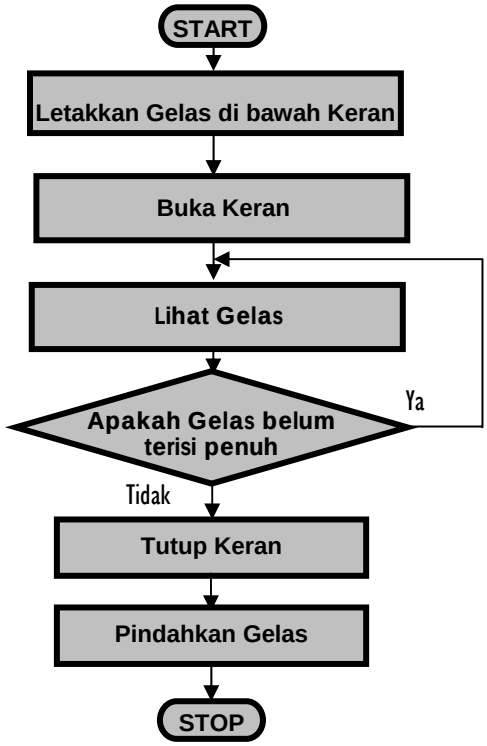
permasalahan mengisi gelas dengan air dari keran. Kasus mengisi gelas dengan air dari keran dapat dijabarkan dengan algoritma program seperti Gambar 5.5.

| STEP | PERINTAH |
|---|---|
| **1.** | Start |
| **2.** | Letakkan Gelas di bawah keran |
| **3.** | Buka keran |
| **4.** | Lihat Gelas |
| **5.** | Apakah Gelas belum terisi penuh |
| **6.** | Jika ya kembali ke STEP 4 |
| **7.** | Tutup keran |
| **8.** | Pindahkan Gelas dari bawah keran |
| **9.** | Stop |

Gambar 5.5. Algoritma Mengisi Gelas dengan Air dari Keran

Gambar 5.5. memberikan gambaran sebuah proses pengisian air ke dalam gelas yang diawali dengan start dan diakhiri dengan stop. Inti proses ada pada step 4, step 5 dan step 6. Terjadi pengamatan atau pendeteksian apakah gelas sudah penuh atau belum. Air dibiarkan mengalir jika gelas belum terisi penuh. Dan jika gelas telah terisi penuh maka harus ada step untuk menghentikan aliran air dengan cara menutup keran.

Aktivitas menuangkan air kedalam gelas mungkin sangat biasa dilakukan di rumah tetapi pada saat diminta menuangkan langkah demi langkah ternyata masih sering ada kesalahan. Step-step pengisian air kedalam gelas dapat digambar menggunakan flowchart seperti Gambar 5.6.




Gambar 5.6. Flowchart Mengisi Gelas dengan Air dari Keran

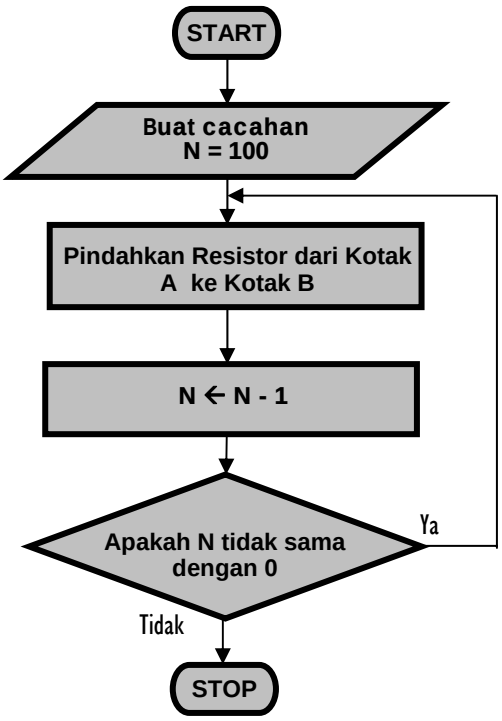
Gambar 5.6. menunjukkan aliran proses pengisian air kedalam gelas dari sebuah keran. Sangat jelas bagaimana air dialirkan ke dalam gelas dan ditunggu sampai gelas terisi penuh dengan air. Ada proses pengamatan dan pengambilan keputusan atas keadaan gelas apakah sudah penuh terisi air atau belum. Dua contoh kasus ini akan memberi gambaran bagaimana permasalahan dipecahkan dengan program.

Permasalahan lain yang bisa digunakan sebagai kasus adalah permasalahan memindahkan 100 buah Resistor dari Kotak A ke Kotak B. Kasus ini sedikit berbeda dengan dua kasus sebelumnya. Kasus ini memerlukan pengulangan sebuah proses pemindahan. Untuk memahami langkah-langkah penyelesaian kasus ini perhatikan algoritma program Gambar 5.6. dan flowchart Gambar 5.7.

| STEP | PERINTAH |
|---|---|
| **1.** | Start |
| **2.** | Buat cacahan jumlah Resistor N = 100 |
| **3.** | Pindahkan Resistor dari Kotak A ke Kotak B |
| **4.** | Kurangi nilai N = N - 1 |
| **5.** | Apakah Nilai N tidak sama dengan 0 |
| **6.** | Jika ya kembali ke STEP 3 |
| **7.** | Stop |

Gambar 5.7. Algoritma Memindahkan 100 buah Resistor dari Kotak A ke Kotak B

Gambar 5.6. menunjukkan adanya pengaturan nilai cacahan N=100 sebagai jumlah pengulangan. Dengan mengurangkan nilai N dengan 1 dan melihat apakah nilai N tidak sama dengan 0 lalu proses pemindahan diulangi kembali merupakan cara bagaimana komputer melakukan hal sama secara berulang-ulang. Pada Gambar 5.7. terlihat jelas bagaimana proses pemindahan resistor berlangsung 100 kali karena ada proses pengulangan sebagai akibat dari iterasi pengurangan nilai cacahan N. Nilai cacahan N dikurangi 1 dan kemudian ada proses pengambilan keputusan apakah N tidak sama dengan 0 untuk kembali lagi ke proses pemindahan resistor. Menjadi sangat sederhana pengulangan proses cukup dikendalikan dengan pengaturan isi cacahan N.



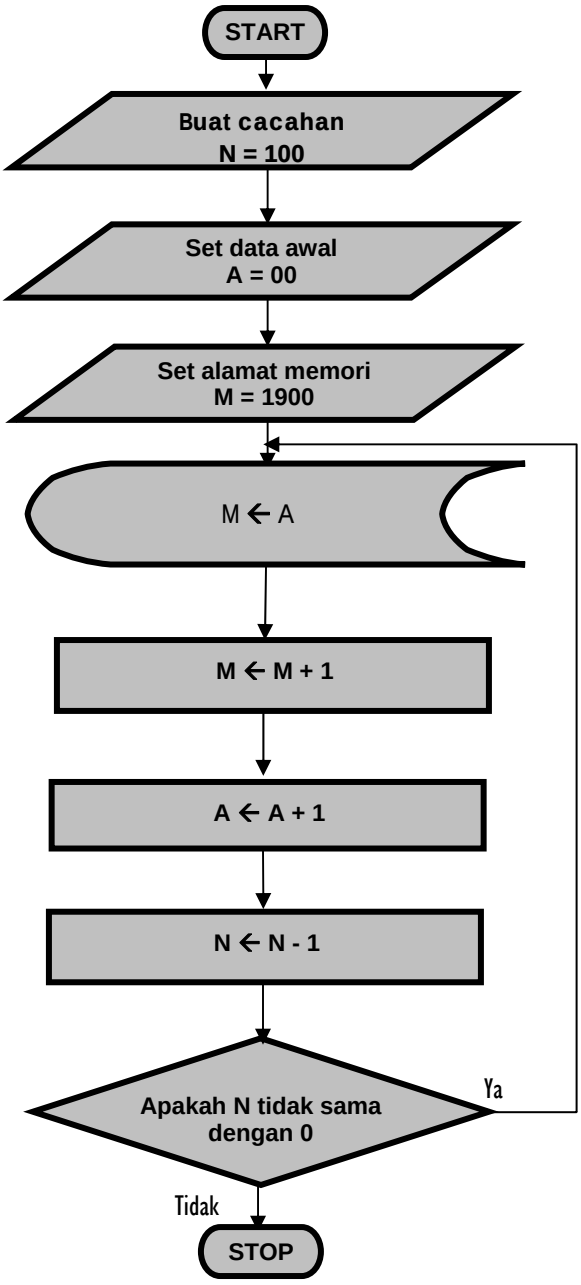

Gambar 5.8. Flowchart Memindahlkan 100 buah Resistor dari Kotak A ke Kotak B

Model ini menunjukkan bahwa proses yang sama jika merupakan pengulangan adalah hal yang sangat sederhana dalam proses sebuah komputer.

Permasalahan lain yaitu bagaimana membangkitkan 100 data bilangan desimal mulai dari 0 pada memori mulai alamat 1900 dapat digunakan sebagai kasus pengembangan algoritma dan flowchart program. Untuk membangkitkan bilangan desimal mulai 0 sampai dengan 99 pada memori mulai alamat 1900 dapat menggunakan algoritma dan flowchart program seperti ambar 5.8 dan 5.9.

| STEP | PERINTAH |
|---|---|
| **1.** | Start |
| **2.** | Buat cacahan jumlah bilangan N = 100 |
| **3.** | Set data awal A = 0 |
| **4.** | Set alamat memori M = 1900 |
| **5.** | Simpan data A di memori alamat M |
| **6.** | Tambahkan nilai alamat memori M = M + 1 |
| **7.** | Tambahkan nilai data A = A + 1 |
| **8.** | Kurangkan nilai cacahan N = N - 1 |
| **9.** | Apakah nilai N tidak sama dengan 0 |
| **10.** | Jika Ya kembali ke STEP 5 |
| **11.** | Selesai |

Gambar 5.9 Algoritma Membangkitkan 100 data bilangan desmal di memori alamat 1900




Gambar 5.10. Flowchart Membangkitkan 100 data bilangan desimal di memori alamat 1900