BAB II

LANDASAN TEORI

2.1 *Programmable Logic Controller* (PLC)

PLC merupakan suatu piranti elektronik yang dirancang untuk beroperasi secara *digital* dengan menggunakan memori sebagai media penyimpanan instruksi-instruksi *internal* untuk menjalankan fungsi-fungsi logika, seperti fungsi pencacah, fungsi urutan proses, fungsi pewaktu, fungsi aritmatika, dan fungsi yang lainnya dengan cara memprogramnya. Program-program dibuat kemudian dimasukkan dalam PLC melalui *programmer*/monitor. Pembuatan program dapat dilakukan melalui komputer sehingga dapat mempercepat hasil pekerjaan. PLC dapat digunakan untuk memonitor jalannya proses pengendalian yang sedang berlangsung, sehingga dapat dengan mudah dikenali urutan kerja (*work squence*) proses pengendalian yang terjadi pada saat itu (Budiyanto. M, 2003:1)

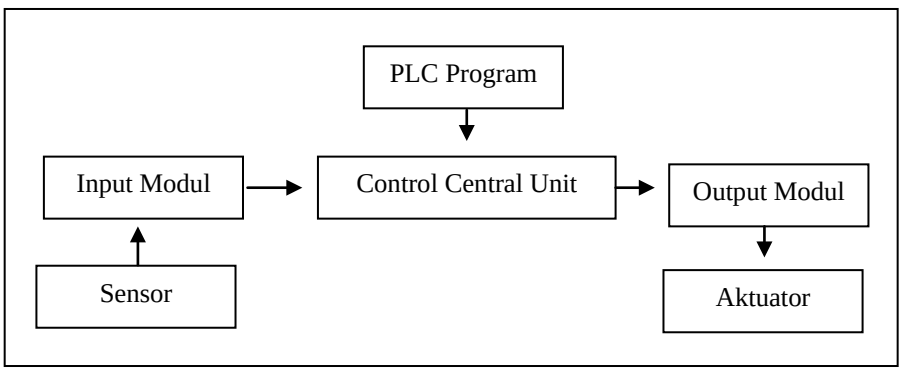
PLC pertama kali digunakan sekitar pada tahun 1960-an untuk menggantikan peralatan konvensional yang begitu banyak. Perkembangan PLC saat ini terus mengalami perkembangan sehingga bentuk dan ukurannya semakin kecil. Saat ini terdapat PLC yang dapat dimasukkan dalam saku karena bentuk dan ukurannya yang sangatlah kecil, dan dalam perkembangannya, dimasa yang akan datang akan diperkenalkan PLC dengan bentuk dan ukuran sebesar kotak rokok.

Pada tahun 1980-an harga PLC masih terhitung mahal, namun saat ini dapat dengan mudah ditemukan dengan harga yang relatif murah. Beberapa

perusahaan komputer dan elektronik menjadikan PLC menjadi produk terbesar yang terjual saat itu. Pertumbuhan pemasaran PLC mencapai jumlah 80 juta dolar di tahun 1978 dan 1 milyar dolar pertahun hingga tahun 2000 dan angka ini terus berkembang, mengingat penggunaan yang semakin luas, terutama untuk proses pengontrolan di industri, pada alat-alat kedokteran, alat-alat rumah tangga.

Pabrik pembuat PLC mendesain sedemikian rupa sehingga pengguna dapat dengan mudah menguasai fungsi-fungsi dan logika-logika hanya dalam beberapa jam saja. Fungsi-fungsi dasar yang banyak digunakan antara lain: kontak-kontak logika, pewaktu (*timer*), pencacah (*counter*), dan sebagainya. Bagi yang mempunyai latar belakang logika-logika *digital* akan dengan mudah menguasainya dalam beberapa jam saja, berlainan halnya dengan orang yang tidak memiliki latar belakang ini akan memakan waktu agak lama untuk menguasai fungsi dan logika-logika kendali PLC.

PLC atau biasa disebut *Programmable Controller* (PC) adalah suatu perangkat yang dapat dengan mudah diprogram untuk mengontrol peralatan. PLC sederhana mempunyai komponen utama berupa *Central Control Unit* (CCU), *Unit* I/O, *Programing Console*, *Rack* atau *Mounting assembly* dan catu daya, sistem komponen dari PLC adalah seperti gambar dibawah ini.



Gambar 2.1. Sistem Komponen Dari sebuah PLC

A. *Central Control Unit* (CCU)

CCU merupakan *unit* pusat pengolah data yang digunakan untuk melakukan proses pengolahan data dalam PLC. CCU merupakan sebuah *microprocessor*, jenis *processor* yang dipergunakan tergantung pada vendor pembuat PLC, untuk PLC FESTO DIDACTIC SERI FPC 100 menggunakan *microcontroller* 8031.

B. *Unit Input Output*

Fungsi dari sebuah modul *input* adalah untuk mengubah sinyal *input* dari sensor ke PLC untuk diproses dibagian *Central Control Unit*, sedangkan modul *output* adalah kebalikannya, mengubah sinyal PLC ke dalam sinyal yang sesuai untuk menggerakkan aktuator.

Fungsi terpenting dari sebuah modul *input* adalah sebagai berikut :

a. mendeteksi sinyal masukan.
b. mengatur tegangan kontrol untuk batas tegangan logika masukan yang diijinkan.
c. melindungi peralatan elektronik yang *sensitive* terhadap tegangan luar.
d. menampilkan sinyal masukan tersebut.

Fungsi terpenting dari sebuah modul *output* adalah sebagai berikut :

a. Mengatur tegangan kontrol untuk batas tegangan logika keluaran yang diijinkan.
b. Melindungi peralatan elektronik yang *sensitive* terhadap tegangan luar.
c. Memberikan penguatan sinyal *output* sebelum dikeluarkan sehingga cukup kuat untuk mengerakkan aktuator.

d. Memberikan perlindungan terhadap arus hubungan singkat (*short-circuit*) dan pembebanan lebih (*overload*).

Beberapa kelebihan yang dimiliki oleh PLC dibanding dengan kontrol relay konvensional, adalah:

1. *Fleksibel*

   Sebelum ditemukannya PLC, setiap mesin mempunyai alat kontrol atau pengendali tersendiri dimisalkan terdapat 15 buah mesin, maka alat pengendali yang diperlukan juga terdapat 15 buah. Lain halnya sekarang ini dengan adanya PLC maka untuk beberapa mesin hanya memerlukan l buah PLC saja.

2. Deteksi dan koreksi kesalahan lebih mudah

   Setelah desain program kontrol telah selesai dibuat, kemudian dimasukkan dalam PLC dengan cara memprogramnya, maka program tersebut dapat dengan mudah diubah dengan menggunakan keyboard hanya dalam beberapa menit saja. Setelah itu program kembali dapat dijalankan, jika masih terdapat kesalahan maka dapat dikoreksi dengan menggunakan diagram tangga *(ladder diagram)* sehingga koreksinya dapat dengan segera dilaksanakan.

3. Harga relatif murah

   Perkembangan teknologi memungkinkan untuk meningkatkan beberapa fungsi dengan bentuk ukuran yang semakin kecil. Tentunya hal ini juga akan menurunkan harga pembuatan yang mahal. Salah satu fungsi yang terus ditingkatkan adalah modul I/O (masukkan/keluaran). Saat ini kita mendapatkan PLC dengan jumlah masukkan dan keluaran yang banyak dengan harga yang relatif murah.

4. Pengamatan visual *(visual observation)*

   Operasi PLC saat menjalankan program yang telah dibuat dapat dilihat dengan teliti dengan menggunakan *layer CRT* (*Cathode Ray Tube*), sehingga ini sangat memudahkan dalam proses pencarian, pengamatan, atau dalam pembenahan program. Dengan demikian proses pembenahan hanya membutuhkan waktu yang relatif singkat.

5. Kecepatan operasi *(speed of operation)*

   Kecepatan operasi PLC sangatlah cepat. Kecepatan operasi ini adalah untuk mengaktifkan fungsi-fungsi logika hanya dalam waktu beberapa milidetik, dikarenakan menggunakan rangkaian elektronik sehingga operasinya sangatlah cepat, berlainan saat digunakan relay magnetik, yang mempunyai kecepatan operasinya lebih lambat.

6. Lebih sederhana dan mudah dalam penggunaannya, memodifikasi lebih mudah tanpa tambahan biaya.

Beberapa kekurangan yang dimiliki oleh PLC dibanding dengan kontrol relay konvensional, adalah:

1. Teknologi baru, sehingga dibutuhkan waktu untuk mengubah sistem konvensional yang telah ada.
2. Keadaan lingkungan. Untuk proses seperti pada lingkungan panas yang tinggi, *vibrasi* yang tinggi penggunaannya kurang cocok, karena dapat merusak PLC.

2.1.1 Konsep PLC

Konsep dari PLC sesuai dengan namanya, adalah sebagai berikut:

a. *Programmable*

Menunjukkan kemampuannya yang dapat dengan mudah diubah-ubah sesuai program yang dibuat.

b. *Logic*

Menunjukkan kemampuan dalam memproses *input* secara aritmetik (membandingkan, menjumlah, membagi dan sebagainya).

c. *Controller*

Kemampuan dalam mengontrol dan mengatur proses sehingga menghasilkan *output* yang diinginkan.

2.1.2 Fungsi PLC

Fungsi dari PLC dapat dibagi secara umum dan secara khusus. Secara umum fungsi PLC adalah sebagai berikut :

a. *Control Sequence*

PLC memproses *input* sinyal *biner* menjadi sinyal *output* yang digunakan untuk keperluan pemrosesan teknik dan yang secara berurutan *(sequence).* PLC menjaga agar semua *STEP* dalam proses *sequence* berlangsung dalam urutan yang tepat.

b. *Monitoring Plant*

PLC secara terus-menerus memonitor status suatu sistem (misalnya temperatur, tekanan, tingkat ketinggian) dan mengambil tindakan yang diperlukan sehubungan dengan proses yang dikontrol (misalnya nilai telah melebihi batas) atau menampilkan pesan tersebut pada operator.

Sedangkan fungsi PLC secara khusus adalah memberikan *input* ke *Computerized Numerical Control* (CNC). Beberapa PLC dapat memberikan *input* ke CNC untuk kepentingan pemrosesan lebih lanjut. CNC bila

dibandingkan dengan PLC mempunyai ketelitian yang lebih tinggi dan lebih mahal harganya. CNC biasanya dipakai untuk proses *finishing*, membentuk benda kerja, digunakan pada unit press, moulding.

2.1.3 Kontrol Konversional

Kontrol konvensional yang menggunakan relay atau kontraktor mempunyai keuntungan dan kerugian bila digunakan sebagai rangkaian kontrol bila dibandingkan kontrol dengan menggunakan PLC.

Relay sendiri merupakan kontrol elektronik, karena terdapat koil atau kumparan yang akan menggerakkan kontak membuka atau menutup bila kumparannya diberi arus listrik. Berikut ini adalah keuntungan dan kerugian menggunakan relay:

Keuntungan :

a. Mudah diadaptasikan untuk tegangan yang berbeda.
b. Tidak banyak dipengaruhi oleh temperatur sekitarnya. Relay terus beroperasi pada temperatur 353 K (80 derajat celcius) sampai 240 K (-33 derajat celcius).
c. Tahanan yang relatif tinggi antara kontak kerja pada saat terbuka.
d. Beberapa sirkuit terpisah dapat dihidupkan.
e. Sirkuit yang mengontrol relay dan sirkuit yang membawa arus yang terhubung.
f. Fisik terpisah satu sama lainnya.

Kerugian :

a. Kontak dibatasi pada keausan dari bunga api atau dari oksidasi (material kontak yang terbaik adalah platina, emas, perak). Menghabiskan banyak tempat dibandingkan dengan transistor.
b. Menimbulkan bunyi selama proses kontak.
c. Kecepatan kontak yang terbatas 3 ms sampai 17 ms.
d. Kontaminasi (debu) dapat mempengaruhi umur kontak.

2.1.4 PLC FESTO

Salah satu PLC yang dimiliki STIKOM dan digunakan untuk praktikum adalah PLC FESTO dari Jerman, seri FPC 101 B-LED dan FPC 101 AF. PLC ini mempunyai kelebihan dapat mengenal program dengan bahasa pemrograman tingkat tinggi (*high level languange*), yaitu *Statement List* atau STL, selain menggunakan *Ladder Diagram* yang sudah umum dan menggunakan pemrograman *matriks* MAT. Bahkan untuk seri tertentu dapat diprogram dengan menggunakan bahasa *BASIC* atau *function chart* FUC (Indrijono Dwi,1999:1)

PLC FPC 101B-LED memiliki spesifikasi yaitu:

a. Indikator untuk status dan *error*.
b. Pemrograman yang mudah melalui PC dengan *ladder Diagram* dan *Statement List.*
c. Perlindungan *output* dari *short-circuit*.
d. Pengaman polaritas *power* suplay.
e. LED indikator untuk *input* dan *output.*

Data teknik PLC FPC 101B-LED

a. 21 *input*.

b. 14 *output*.

c. 32 *timer*.

d. 16 *counter*.

e. 64 *register*.

f. 256 *flag*.

g. 12 *KBytes user memory*.

h. 7,5 W untuk tiap *output*.

Sensor – sensor yang digunakan di Laboratorium PLC adalah:

a. Push button switch.

b. Switch toggle.

c. Sensor *capasitive.*

d. Sensor *induktive.*

e. Sensor *optik.*

f. Limit switch.

Aktuator yang digunakan pada Laboratorium PLC adalah:

a. Single selenoid.

b. Double selenoid.

c. Indikator Lamp.

d. Buzzer.

2.1.5 Bahasa Pemrograman

Terdapat banyak pilihan bahasa untuk membuat program dalam PLC. Masing-masing bahasa mempunyai keuntungan dan kerugian sendiri-sendiri

tergantung dari sudut pandang kita sebagai *user*. *Ladder Diagram* adalah bahasa yang dimiliki oleh setiap PLC.

A. *Ladder Diagram* (LDR)

Ladder diagram menggambarkan program dalam bentuk grafik. *Diagram* ini dikembangkan dari kontak-kontak relay yang terstruktur dan menggambarkan aliran arus listrik. Dalam *ladder diagram* ini terdapat dua buah garis vertikal. Garis vertikal sebelah kiri dihubungkan dengan sumber tegangan positif catu daya aktif sedangkan garis sebelah kanan dengan sumber tegangan negatif catu daya pasif.

Diantara dua garis ini dipasang kontak-kontak yang menggambarkan kontrol dari *switch*, sensor atau *output*. Satu baris dari diagram disebut dengan satu *rung*. *Input* menggunakan simbol "[ ]" (kontak, normal *open*) dan "[/]" (negasi kontak, normal *closed*). *Output* mempunyai simbol "( )" yang terletak paling kanan menempel garis vertikal kanan.

Selama pemrograman setiap simbol yang diberikan adalah alamat PLC sesungguhnya atau merupakan alamat simbolik (misalnya S1, S2, S3, H1).

B. *Statement List* (STL)

Statement list adalah bahasa pemrograman tingkat tinggi. Semua hubungan logika dan kontrol *sequence* dapat diprogram dengan menggunakan perintah dalam bahasa ini.

Perintah-perintah yang digunakan adalah mirip dengan bahasa tingkat tinggi seperti *pascal*. Terdapat kontrol untuk perulangan, *jump* dan sebagainya.

Misalnya:

```
IF              I1.0              “Jika input 1.0 aktif
THEN            SET   T6          “maka aktifkan timer T6.
```

Struktur dari *statement list* secara umum dapat dituliskan sebagai berikut:

PROGRAM

STEP

STATEMENT

BAGIAN KONDISI

BAGIAN PELAKSANA

B.1 *Statement*

Statement merupakan pembentuk dasar dari organisasi program. Masing-masing *statement* terdiri dari bagian kondisi dan bagian pelaksana. Bagian kondisi mengandung satu atau beberapa buah kondisi yang akan diuji (benar atau salah) pada saat program berjalan. Bagian kondisi selalu dimulai dengan kata *IF* (jika). Jika kondisi bernilai benar maka instruksi yang ditulis pada bagian pelaksana akan dijalankan. Awal dari bagian pelaksana dimulai dengan kata *THEN* (maka).

Contoh:

```
IF          I6            “(jika input 6 memberikan sinyal
THEN        SET    O1     “maka nyalakan output 1)
IF          I6            “(Jika input 6 memberikan sinyal
            AND    I2     “dan input 2 memberikan sinyal)
THEN        RESET  O5     “jika ya, matikan output 5,
            SET    O4     “nyalakan output 4)
```

B.2 *STEP*

Program yang tidak menggunakan instruksi *STEP* dapat diproses dengan cara paralel. Tetapi STL menyediakan instruksi *STEP* yang membagi program menjadi bagian-bagian yang lebih kecil.

Dalam sebuah program dapat berisi sampai 256 *STEP* (0 sampai 255). Setiap *STEP* dapat diberi label atau tidak, dan hanya dibutuhkan jika setiap *STEP* tersebut merupakan target dari instruksi *JUMP*.

Bentuk paling sederhana dari instruksi *STEP* paling sedikit mengandung satu *statement*, misalnya:

```
STEP mulai
IF                  I1
THEN      SET       O2
```

Program akan menunggu pada *STEP* ini sampai kondisinya benar, yaitu bagian pelaksana akan dilaksanakan dulu baru setelah itu program akan berlanjut ke *STEP* berikutnya, dalam sebuah *STEP* dapat berisi beberapa *statement* :

```
STEP mulai
IF                  I2
THEN      SET       O5
IF                  I3
THEN      RESET  O3
          SET       O2
```

Jika kondisi *IF* terakhir salah (IF I3) maka program tidak akan berlanjut ke *STEP* berikutnya dan akan kembali ke *statement* pertama dalam *STEP* tersebut (IF I2). Dengan kata lain program akan menunggu sampai kondisi terakhir benar. Aturan pelaksanaan *STEP* :

a. Jika kondisi dari sebuah *statement* terpenuhi maka bagian pelaksana akan dijalankan. Dan Jika kondisi dari sebuah *statement* dalam suatu

STEP tidak terpenuhi maka program akan berpindah ke *statement* berikutnya dalam *STEP* tersebut.

b. Jika kondisi dari *statement* terakhir dalam suatu *STEP* terpenuhi maka bagian pelaksana akan dijalankan dan program berlanjut ke *STEP* berikutnya.

c. Jika kondisi dari *statement* terakhir dalam sebuah *STEP* tidak terpenuhi maka program akan kembali ke *statement* pertama dari *STEP* yang sekarang.

B.3 Instruksi NOP

Instruksi NOP dapat diletakkan pada bagian kondisi atau bagian pelaksana dari sebuah *statement*. Bila digunakan dalam bagian kondisi, instruksi NOP selalu bernilai benar. Dengan kata lain NOP menyebabkan pelaksanaan tanpa suatu kondisi.

```
IF              NOP     “akan selalu bernilai benar
THEN      SET   O1      “(Jadi  output  1  akan  selalu  aktif
                        “pada  saat Program pergi ke STEP
                        “berikutnya).
```

Jika digunakan dalam bagian pelaksana pengertian NOP adalah “tidak melakukan sesuatu”. Hal ini sering digunakan pada saat program harus menunggu untuk kondisi tertentu lalu pindah ke *STEP* berikutnya.

2.1.6 *Timer*

Banyak dari kontrol industri yang memerlukan pemrograman dengan waktu. Sebagai contoh, silinder 2 akan maju jika silinder 1 telah maju lebih dahulu tetapi hanya setelah lima detik. Hal seperti ini dikenal dengan *switch-on*

delay. Penundaan sinyal *switch-on* pada *switching* rangkaian *power* sangat dibutuhkan demi alasan keamanan.

Timer dalam PLC direalisasikan dalam bentuk modul *software* yang didasarkan pada pembangkitan *timing* secara *digital* dari generator pulsa *microprosessor*. Lamanya waktu yang diperlukan ditetapkan dalam program kontrol.

A. Komponen *Timer*

Masing-masing *timer* dalam bahasa pemrograman STL terdiri dari beberapa bagian :

a. *Timer* Status *Bit*, penulisannya "Tn" yang berfungsi menguji apakah *timer* sedang aktif atau tidak. Nilai *bit* berubah menjadi aktif (1) pada saat *timer* dimulai dengan (*SET*). Pada saat periode waktu yang diprogram selesai atau jika *timer* dihentikan (*RESET*) status *bit* berubah menjadi tidak aktif (0).

b. *Timer Preselect*, penulisannya "TPn" yang berfungsi sebagai *operand* 16 *bit* yang berisi nilai awal untuk sebuah *timer* n.

c. *Timer Word*, penulisannya "TWn" yang berfungsi sebagai *operand* 16 *bit* yang secara otomatis memiliki nilai yang sama dengan TP pada saat *timer* dimulai (*SET*). Isinya akan secara otomatis dikurangi oleh sistem pada *interval* yang teratur.

B. Memulai Suatu *Timer*

Memulai *timer* hanya digunakan instruksi *SET* dan menentukan *timer* yang akan dimulai :

```
STEP inisialisasi
```

```
THEN        LOAD        V100        "Nilai 100 = 1 detik
            TO          TP6         "Dimasukkan ke TP6
STEP mulai
IF          I1.0                    "Jika input 1.0 aktif
THEN        SET         T6          "maka aktifkan timer 6
```

Pada saat instruksi *SET* Tn dijalankan, yang terjadi adalah :

1. Nilai yang tersimpan dalam TPn di *copy* ke TWn
2. Tn (*Timer* Status n) menjadi aktif.
3. *Controller* secara otomatis mengurangi nilai yang tersimpan dalam TWn pada *interval* yang teratur, yaitu 10 ms.
4. Pada saat nilai yang tersimpan dalam TWn mencapai 0, Tn menjadi tidak aktif.

C. Menghentikan Suatu *Timer*

Menghentikan suatu *timer* hanya memerlukan perintah *RESET* dan menentukan *timer* yang akan dihentikan :

```
IF          I1.0                    "Jika input 1.0 aktif
THEN        RESET   T6              "Matikan timer 6
```

Pada saat instruksi *RESET* Tn dijalankan *Timer Status Bit* (Tn) menjadi 0 (tidak aktif). Jika *timer* tersebut sebelumnya sudah tidak aktif, tidak ada pengaruhnya jika kita jalankan perintah *RESET* Tn tersebut.

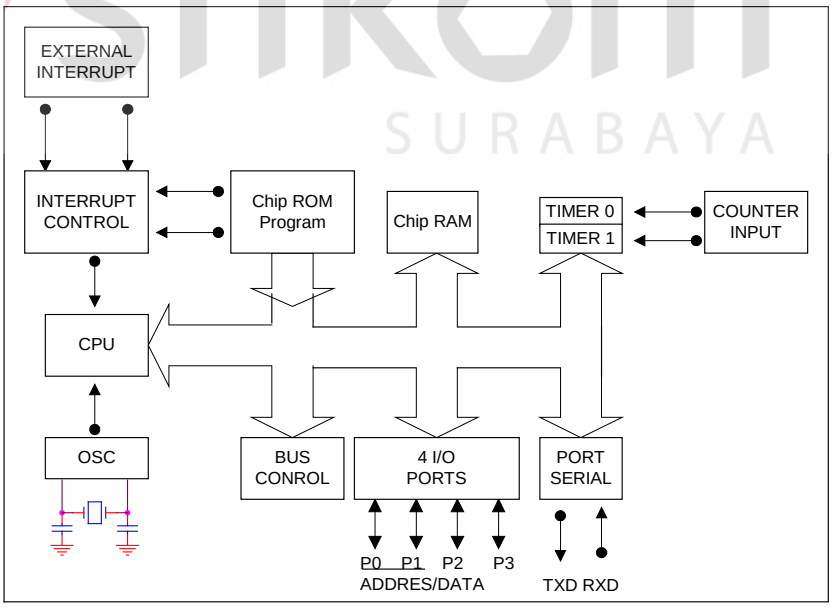
2.2. *Microcontroller* MCS-51

Sebuah *microcontroller* mempunyai sebuah CPU (*Central Processing Unit*) dan terdapat tambahan pemasangan sejumlah RAM *(Random Access Memory)*, ROM *(Read Only Memory)* dan I/O *(Input/Output) port* dan sebuah *Timer* yang semuanya terdapat dalam satu *chip*. Dengan kata lain *processor*,

RAM, ROM, I/O *port* dan *Timer* adalah terpasang bersama dalam satu *chip*. *Microcontroller* 8051 adalah asli dari Intel, beberapa perusahaan juga memproduksi 8051 seperti Atmel, Phillips, AMD, Siemens, Matra dan Dallas Semiconductor.

2.2.1. *Microcontroller* 8051

Pada tahun 1981, perusahaan Intel mengenalkan sebuah *microcontroller* 8 *bit* yang disebut dengan 8051. *Microcontroller* ini mempunyai RAM sebesar 128 *byte*, ROM sebesar 4 *Kbyte*, dua *timer*, satu *serial port*, dan empat *port* (masing-masing sebesar delapan *bit*) semuanya dalam satu *chip*. 8051 adalah sebuah prosesor 8 *bit*, artinya bahwa *CPU* dapat bekerja hanya pada data sebesar 8 *bit* pada waktu yang sama. Data yang lebih besar dari 8 *bit* harus dipecah menjadi 8 *bit* setelah itu diproses oleh *CPU*. 8051 mempunyai 4 *I/O Port* masing-masing sebesar 8 *bit*, lihat Gambar 2.2.



Gambar 2.2. Blok Diagram *Microcontroller* 8051

8051 menjadi sangat terkenal setelah Intel mengijinkan perusahaan lain untuk membuat dan memasarkan beberapa jenis 8051 dan mengikuti syarat program yang kompatibel dengan 8051. Disamping itu terdapat bermacam-macam versi dari 8051 dengan kecepatan yang berbeda dan jumlah dari ROM dipasarkan oleh lebih dari 12 perusahaan.

Tabel 2.1. Perbandingan dari Anggota Keluarga 8051

Feature	**8051**	**8052**	**8031**
ROM	4K	8K	0K
RAM	128	256	128
Timer	2	3	2
I/O Pins	32	32	32
Serial Port	1	1	1
Interrupt Source	6	8	6

8051 adalah anggota yang asli dari keluarga 8051, Intel mengacu kepada *microcontroller* ini sebagai MCS-51, Tabel 2.1 menunjukkan ciri khusus dari *microcontroller* 8051. Ada dua anggota lain dari keluarga *microcontroller* 8051 yaitu 8052 dan 8031.

2.2.2. *Microcontroller* 8052

microcontroller 8052 adalah anggota lain dari keluarga 8051. *microcontroller* 8052 mempunyai semua standar dari 8051 serta terdapat tambahan RAM sebesar 128 *bytes* dan sebuah tambahan *timer*, sehingga 8052 mempunyai RAM sebesar 256 *bytes* dan 3 *timer*. Selain itu juga mempunyai ROM sebesar 8 *Kbytes* untuk program yang terdapat di dalam *chip*, seperti pada tabel 2.1.

2.2.3. *Microcontroller* 8031

Anggota lain dari keluarga 8051 adalah *microcontroller* 8031. *Chip* ini sering dikenal sebagai 8051 tanpa ROM karena tidak mempunyai ROM di dalam *chip*nya, seperti tabel 2.1. Untuk menggunakan *chip* ini harus menambah eksternal ROM. Eksternal ROM ini berisi progam yang akan diambil dan dieksekusi oleh 8031. ROM yang berisi progam untuk *microcontroller* 8031 dapat sebesar 64 *Kbyte*. Dalam proses penambahan ROM eksternal untuk 8031, akan kehilangan 2 *port*. Yang tersisa hanya 2 *port* (dari 4 *port* yang tersedia) untuk I/O. Untuk memecahkan masalah ini, perlu ditambahkan eksternal I/O untuk 8031. Sehingga diperlukan *interfacing* 8031 dengan memori dan I/O *port* seperti menggunakan IC (*Integrated Circuit*) 8255.

2.2.4. *Microcontroller* 8751

Microcontroller 8751 mempunyai 4 *Kbytes* UV (*Ultra Violet*) EPROM di dalam *chip*nya. Menggunakan *chip* ini untuk pengembangan diperlukan PROM *burner* dan *eraser* UV-EPROM sebelum *microcontroller* 8751 diprogram lagi. Pada kenyataannya penghapusan program ROM dari IC 8751 membutuhkan waktu sekitar 20 menit.

2.2.5. Atmel AT89C51

Keluarga *microcontroller* 8051 yang terkenal ini mempunyai ROM yang di dalam IC dalam bentuk *flash memory*. Ini ideal untuk perkembangan yang sangat cepat sejak *flash memory* dapat menghapus dalam hitungan detik dibandingkan 8751 yang memerlukan 20 menit atau lebih untuk menghapus.

Untuk alasan ini, AT89C51 melengkapi 8751 untuk menghilangkan waktu tunggu yang lama untuk menghapus IC. Dengan cara ini kita dapat mengembangkan kecepatan menjadi lebih tinggi. Dalam menggunakan AT89C51, untuk mengembangkan sistem dasar *microcontroller* memerlukan sebuah *ROM burner* yang *support* dengan *flash memory*, maka dengan ini *ROM eraser* tidak diperlukan. Untuk memprogram ulang *flash memory*, isi yang ada di dalamnya harus dihapus lebih dahulu. Penghapusan sebuah *flash memory* dilakukan oleh *ROM burner* dan hal ini menunjukkan mengapa *eraser* yang terpisah tidak diperlukan. *Atmel* versi AT89C51 juga dapat diprogram melalui *serial* COM *port* dari sebuah IBM PC sehingga *ROM burner* tidak diperlukan lagi. Selain itu kapasitas dari ROM pada Atmel berbeda tergantung pada jenisnya, seperti yang terdapat pada tabel dibawah ini:

Tabel 2.2. Macam-macam 8051 dari ATMEL

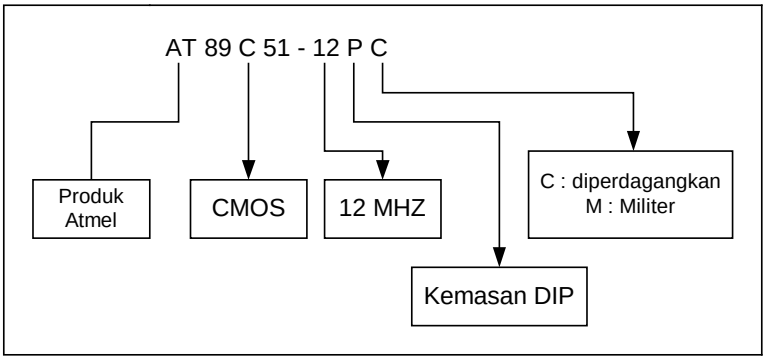
Part Number	ROM	RAM	I/O	TIMER	INTR	VCC
AT89C51	4K	128	32	2	6	5V
AT89LV51	4K	128	32	2	6	3V
AT89C1052	1K	64	15	1	3	3V
AT89C2051	2K	128	15	2	6	3V
AT89C52	8K	128	32	3	8	5V
AT89LV52	8K	128	32	3	8	5V

Tabel 2.3. Macam Kecepatan 8051 dari Atmel

Part Number	Speed	Pins	Kemasan	Digunakan
AT89C51-12PC	12 MHz	40	DIP plastic	Comercial
AT89C51-16PC	16 MHz	40	DIP plastic	Comercial
AT89C51-20PC	20 MHz	40	DIP plastic	Comercial

Ada bermacam-macam versi kecepatan dan kemasan dalam sebuah produk seperti pada gambar 2.3 Untuk contoh, AT89C51-12 PC dimana “C” sebelum 51 adalah untuk CMOS, yang mempunyai konsumsi daya yang kecil,

"12" indikasi 12 MHz, "P" adalah untuk kemasan plastik DIP, dan "C" untuk diperdagangkan, sedangkan "M" untuk keperluan militer. Yang sering digunakan oleh mahasiswa untuk proyek adalah AT89C51-12 PC.



Gambar 2.3. Program AT89C51

2.2.6. DS5000 Dallas Semiconductor

Versi populer yang lain dari 8051 adalah IC DS5000 dari dallas semiconductor. ROM yang terdapat dalam *chip* DS5000 dalam bentuk NV-RAM. Kemampuan membaca dan menulis NV-RAM memperbolehkan mengisi program ke dalam ROM ketika NV-RAM berada dalam sebuah sistem. Ini dapat dilakukan melalui *serial* COM *port* IBM PC. Kemampuan NV-RAM untuk mengubah isi ROM beberapa *bytes* setiap waktu. Dibandingkan dengan UV-EPROM dan *flash memory* yang mana isi ROM harus dihapus dahulu sebelum diprogram ulang.

Table 2.4. Dallas Semiconductor's Soft *Microcontroller*

Part Number	ROM	RAM	I/O	Timers	Interrupt	Vcc	Packaging
DS5000-8	8K	128	32	2	6	5V	40
DS5000-32	32K	128	32	2	6	5V	40
DS5000T-8	8K	128	32	2	6	5V	40
DS5000T-8	32K	128	32	2	6	5V	40

Tabel 2.5 Macam-macam Kecepatan Dallas Semiconductor

Part Number	NV-RAM	Kecepatan
DS5000-8-8	8K	8 MHz
DS5000-8-12	8K	12 MHz
DS5000-32-8	32K	8 MHz
DS5000T-32-8	32K	8 MHz (dengan RTC)
DS5000-32-12	32K	12 MHz
DS5000-8-12	8K	12 MHz (dengan RTC)

2.2.7. Phillip

Keluarga 8051 yang lain adalah perusahaan Phillip. Beberapa produk ini memiliki ciri khusus seperti adanya *A-to-D converter, D-to-A converter, I/O ports* dan *OTP* serta *flash memory* (Mazidi, 2000 : 28).

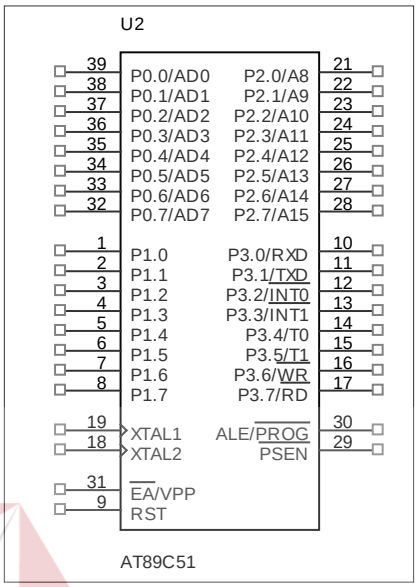
2.2.8. I/O *Port*

Keluarga 8051 anggota (8751, 89C51, DS5000) mempunyai kemasan yang berbeda, seperti DIP (*dual in-line package*), QFP (*quad flat package*), dan LLC (*leadless chip carrier*). Keluarga 8051 mempunyai 40 pin, dimana masing-masing pin mempunyai bermacam-macam fungsi seperti I/O, RD, WR, alamat data, dan *interrupt*. Selain 40 pin, keluarga 8051 juga mengeluarkan 20 pin dengan fungsi yang hampir sama.

Gambar 2.4 adalah *microcontroller* yang mempunyai 40 pin, 32 pin digunakan sebagai *port* yang terdiri dari 4 *port* yaitu P0, P1, P2 dan P3 masing-masing *port* mempunyai 8 pin. Sisa pin didesain sebagai VCC, GND, XTAL1, XTAL2, RST, $\overline{EA}$, $\overline{PSEN}$ dan ALE. Dari 8 pin, enam diantaranya (VCC, GND, XTAL1, XTAL2, RST dan $\overline{EA}$) digunakan oleh keluarga 8051 dan 8052. Dan dua pin yang lain adalah $\overline{\text{PSEN}}$ dan ALE, biasanya digunakan 8031. Berikut ini pembahasan tentang fungsi masing-masing pin :

A. VCC

Pin 40 digunakan sebagai catu daya dengan tegangan sumber sebesar 5V



Gambar 2.4. Pin diagram 8051

B. GND

Pin 20 adalah Ground.

C. XTAL 1 dan XTAL 2

Di dalam IC *Microcontroller* 8051 terdapat osilator, tetapi memerlukan sebuah eksternal *clock* untuk menjalankannya. Sebuah kristal yang dihubungkan ke XTAL1 (pin 19) dan XTAL2 (pins 18) salah satu kakinya dihubungkan ke kapasitor sebesar 30 pF sedangkan kaki kapasitor lainnya dihubungkan ke *ground*.

Keluarga 8051 mempunyai kecepatan yang bervariasi, kristal yang digunakan harus sama atau kurang dari kecepatan yang dimiliki oleh *chip* tersebut. Misalkan *microcontroller* mempunyai kecepatan maksimum sebesar 12 MHz, maka kristal yang digunakan harus sama dengan 12 MHz atau kurang.

D. RST

Pin 9 adalah *reset* dengan kondisi *active high*. Kondisi ini biasanya disebut sebagai *power-on-reset*. Jika terjadi *power-on-reset* semua aplikasi berhenti dan semua nilai yang terdapat pada *register* semuanya hilang. Tabel 2.6 menunjukkan sebagian nilai *register* apabila terjadi *reset.*

Tabel 2.6. Kondisi *Reset*

Register	Nilai *Reset*
PC	0000
ACC	0000
B	0000
PSW	0000
DPTR	0000

PC (*Program Counter*) bernilai 0 jika terjadi *reset*, lalu CPU akan mengambil progam pertama dari ROM pada lokasi 0000H. Ini artinya jika meletakkan program harus terdapat di alamat ROM yang ke 0000H. Untuk *input reset* yang efektif, *reset* harus mempunyai durasi 2 *machine cycle*.

E. $\overline{\text{EA}}$

Anggota keluarga 8051 yang mempunyai ROM *internal* adalah 8751, 89C51 atau DS5000. Untuk menyimpan program ke ROM *internal*, $\overline{\text{EA}}$ harus dihubungkan ke VCC.

Lain dengan 8031 dan 8032, dimana di dalam *chip* tidak terdapat ROM *internal* sehingga untuk menyimpan program menggunakan ROM eksternal. Oleh karena itu $\overline{\text{EA}}$ dihubungkan ke GND.

F. $\overline{\text{PSEN}}$

Untuk penyimpanan program di ROM eksternal, $\overline{\text{PSEN}}$ dihubungkan dengan $\overline{\text{OE}}$ pada ROM eksternal untuk mengambil satu instruksi. $\overline{\text{PSEN}}$ ini digunakan di *microcontroller* 8031 dan keluarga 8051 yang menggunakan ROM eksternal.

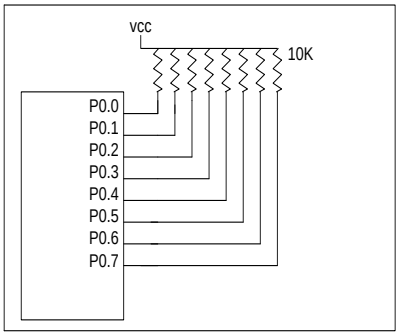
G. ALE

ALE (*Address Latch Enable*) adalah suatu *output* pin yang mempunyai *active high*. Ketika 8031 menggunakan memori eksternal, *port* 0 menyediakan alamat dan data. ALE digunakan untuk *demuxtiplexing* alamat dan data dengan menghubungkan ALE ke G (74ls573).

H. *Port* 0

Port 0 sebanyak 8 pin (pin 32-39) digunakan sebagai *input* maupun *output*, dimana masing-masing pin dihubungkan ke tahanan *pull up* sebesar 10K, seperti gambar 2.5. Pada kenyataannya P0 adalah *open drain*, tidak seperti P1, P2 dan P3. *Open drain* digunakan untuk MOS *chip* sama seperti *open collector* yang digunakan oleh TTL *chip*.

Selain sebagai *output*, *port* 0 juga dapat digunakan untuk *input*. Sebelum data dibaca oleh *port* 0 maka pada semua pin data ditulis dengan nilai 1(*high*). Setelah itu data dapat dibaca oleh *port* 0.



Gambar 2.5. Tahanan *Pull-Up*

I. *Port* 1

Port 1 sebanyak 8 pin (pin 1-8). digunakan sebagai *input* maupun *output* pada *port* ini tidak memerlukan tahanan *pull up* karena sudah tersedia *internal pull up*.

Selain sebagai *output port* 0 juga dapat digunakan untuk *input*. Sebelum data dibaca oleh *port* 1 maka pada semua pin data ditulis dengan nilai 1(*high*) setelah itu data dapat dibaca oleh *port* 1.

J. *Port* 2

Port 2 sebanyak 8 pin (pin 21-28) digunakan sebagai *input* maupun *output*, dan tidak memerlukan tahanam *pull up* karena sudah tersedia *internal pull up*.

Selain sebagai *output*, *port* 0 juga dapat digunakan untuk *input*. Sebelum data dibaca oleh *port* 2, maka pada semua pin data ditulis dengan nilai 1(*high*) setelah itu data dapat dibaca oleh port 2.

K. *Port* 3

Port 3 sebanyak 8 pin (pin 10-17) digunakan sebagai *input* atau *output* dan tidak memerlukan tahanan *pull-up*, sama dengan P1 dan P2. *Port* 3

mempunyai fungsi tambahan seperti *interrupts*. Tabel 2.7 adalah informasi mengenai fungsi P3 yang digunakan oleh kedua *chip* 8051 dan 8031.

P3.0 dan P3.1 digunakan untuk sinyal komunikasi *serial* RxD dan TxD. P3.2 dan P3.3 digunakan untuk *interrupt* eksternal. P3.4 dan P3.5 digunakan untuk *timer* 0 dan 1. Pada akhirnya, P3.6 dan P3.7 digunakan untuk sinyal $\overline{\text{WR}}$ dan $\overline{\text{RD}}$ (Mazidi, 2000 : 90).

Tabel 2.7. Fungsi *Port* 3

P3 Bit	Fungsi	Pin
P3.0	RxD	10
P3.1	TxD	11
P3.2	$\overline{INT0}$	12
P3.3	$\overline{INT1}$	13
P3.4	$\overline{T0}$	14
P3.5	$\overline{T1}$	15
P3.6	$\overline{\text{WR}}$	16
P3.7	$\overline{\text{RD}}$	17

2.2.9 Memori Eksternal

Arsitektur MCS-51 menyediakan kapasitas program memori eksternal dan data memori eksternal sebesar 64K. Tambahan ROM dan RAM dapat diberikan jika diperlukan begitu pula *interface* untuk menambahkan I/O.

Ketika memori eksternal digunakan *port* 0 tidak dapat digunakan sebagai I/O. *Port* 0 menjadi *multiplexed* alamat (A0-A7) dan data (D0-D7) *bus* dan *port* 2 digunakan untuk alamat *bus* dengan *byte* tinggi (A8-A15).

A. Pengaksesan Memory Program eksternal.

Program memori eksternal adalah memori yang hanya dapat dibaca dan diaktifkan oleh sinyal $\overline{\text{PSEN}}$. Ketika program eksternal digunakan, kedua *port* 0 dan *port* 2 tidak dapat digunakan sebagai *port* pada umumnya.

Untuk mengakses memori eksternal memerlukan $\overline{\text{PSEN}}$ dan ALE. Dimana $\overline{\text{PSEN}}$ dihubungkan ke $\overline{\text{OE}}$ untuk mengambil program dalam ROM. Sedangkan ALE digunakan untuk *demuxtiplexing* alamat dan data yang menghubungkan ALE ke G (74ls573).

B. Pengaksesan Memori Data eksternal

Data memori eksternal adalah memori yang dapat dibaca atau ditulis oleh $\overline{\text{RD}}$ dan $\overline{\text{WR}}$. Instruksi yang dapat mengakses memori eksternal adalah *movx*, sedangkan *register* untuk mengakses memori eksternal adalah DPTR (16 bit data) dan R0 atau R1 untuk alamat *register*.

Untuk mengakses RAM agar dapat dibaca atau ditulis oleh *microcontroller*, menghubungkan $\overline{\text{RD}}$ pada *microcontroller* dihubungkan dengan $\overline{\text{OE}}$ yang ada di RAM untuk membaca data sedangkan $\overline{\text{WR}}$ pada *microcontroller* dihubungkan dengan $\overline{\text{W}}$ pada RAM. Untuk menghubungkan alamat dan data *bus* sama dengan ROM.

Ketika sebuah instruksi *MOVX @DPTR,A* dijalankan maka pin $\overline{\text{WR}}$ menjadi berlogika 0 (*low*) sedangkan pin $\overline{\text{RD}}$ berlogika 1 (*high*), proses diatas adalah proses menulis (I. Scott MacKensi,1999).

2.2.10 Program *Counter* di MCS-51

Register terpenting lain pada MCS-51 adalah PC yang nilainya berdasarkan instruksi yang di akses. Progam *counter* di MCS-51 mempunyai lebar 16 bit data, sehingga *microcontroller* dapat mengakses dari alamat 0000H sampai FFFFH.

Pertama kali *microcontroller* mendapatkan daya nilai dari program *counter* adalah 0000H, sehingga instruksi yang diakses oleh *microcontroller* berada pada alamat 0000H. Ini artinya untuk penulisan didalam ROM harus diletakkan pada alamat 0000H.

A. Penempatan program dalam ROM

Untuk mendapatkan pengertian yang lebih baik dari aturan program *counter* dalam mengambil dan mengeksekusi sebuah program dijelaskan dalam bab ini. Pertama harus mengerti penempatan program pada ROM pada MCS-51 seperti tabel di bawah ini.

Tabel 2.8. Eksekusi Program

ROM alamat	Bahasa mesin	Bahasa assembly
0000	7D25	MOV R5,#10h
0002	7400	MOV A,#01H
0004	2D	ADD A,R5

Setelah program dimasukkan kedalam ROM, *opcode* dan *operand* ditempatkan di memori ROM yang lokasinya di alamat 0000H.

Tabel 2.9. Isi ROM

Alamat	Code
0000	7D
0001	25
0002	74
0003	01
0004	2D

Pada alamat 0000H berisi 7DH *MOV* R5 dan alamat 0001H berisi *operand* bernilai 25H, dimana program ini untuk memindahkan *operand* 25H kedalam R5. Alamat 0002H berisi 74 dan alamat 0003H berisi 01H dengan instruksi *MOV* A,01H. Sedangkan untuk proses penjumlahan ADD A,R5 berada pada alamat 0004H.

B. Eksekusi sebuah Program

Asumsi tentang program yang di masukkan kedalam ROM MCS-51, ikuti langkah dibawah ini yang menggambarkan aksi dari MCS-51 setelah *power* dihidupkan :

a) Ketika daya MCS-51 dihidupkan, program *counter* mempunyai nilai 0000H dan mengambil program kedalam alamat memori 0000H. Alamat 0000H yang diambil oleh CPU adalah 7D, setelah instruksi tersebut dijalankan, CPU mengambil *operand* 25 untuk dimasukkan kedalam R5.

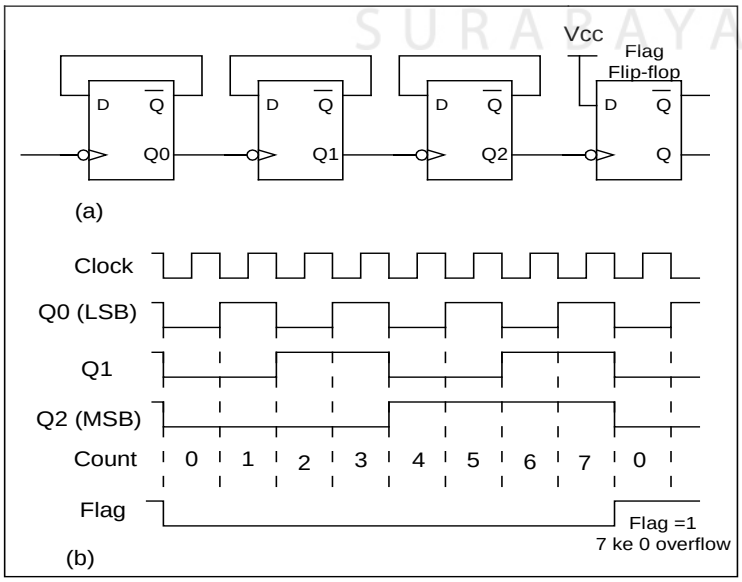
b) Setelah mengeksekusi *opcode* 74H, nilai *operand* 01H dipindahkan ke dalam R7 lalu program *counter* ditambah menjadi 0003H.

c) Sekarang PC = 0004H, instruksi selanjutnya adalah “ADD A,R5”. Instruksi ini mempunyai 1 *byte* instruksi, sehingga PC = 00004H (Mazidi, 2000 : 46).

2.2.11 Operasi *Timer*

Timer adalah suatu rangkaian *flip-flop* pembagi dua yang menerima sinyal *input* sebagai *clock*. *Clock* dihubungkan ke *flip-flop* yang pertama, yang membagi frekuensi *clock* menjadi 2. *Output* dari *flip-flop* yang pertama dihubungkan ke *clock flip-flop* yang ke dua, dimana yang membagi frekuensi *clock* menjadi 2 dan seterusnya, seperti pada gambar 2.5 Karena masing-masing *flip-flop* dibagi dua, sebuah *timer* dengan langkah ke n sama dengan 2^n frekuensi *clock*. *Output flip-flop* yang terakhir adalah sebuah *timer overflow flip-flop* atau *flag*, yang dapat dicoba melalui *software* atau dibangkitkan dengan *interrupt*. Nilai *biner* di dalam *timer flip-flop* dapat melalui perhitungan jumlah pulsa dari *clock* sejak *timer* di *start*.

Ada dua *timer* 16 bit yang masing-masing mempunyai empat *mode*. *Timer* dapat digunakan untuk *interval* waktu, menghitung keadaan, dan *Baud rate* untuk *serial* (I. Scott MacKensi, 1999 : 82).



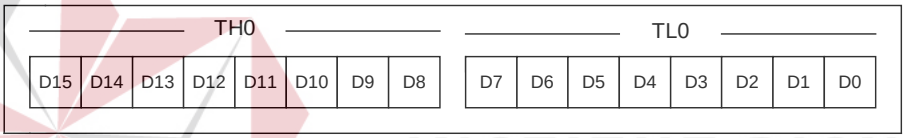
Gambar 2.6. 3 bit *timer* (a) *Schematics*, (b) *Timing Diagram*

A. *Register Timer*

MCS-51 mempunyai dua *timer* yaitu *timer* 0 dan *timer* 1, kedua *timer* tersebut mempunyai panjang 16 *bit*. MCS-51 mempunyai arsitektur 8 *bit*, masing-masing *timer* mengakses dua bagian dari *register byte* rendah dan *byte* tinggi.

A.1. *Register Timer* 0

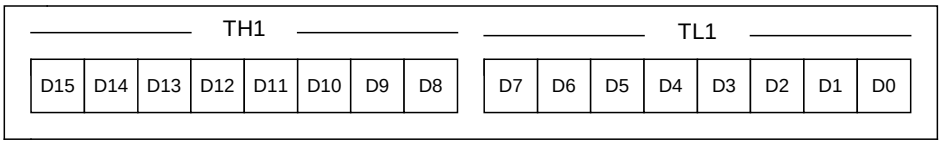
Timer 0 mengakses *byte* rendah dan *byte* tinggi dari *register* 16 *bit*. *Byte register* rendah disebut sebagai TL0 (*Timer* 0 *Low byte*) dan *Byte register* tinggi di sebut sebagai TH0 (*Timer* 0 *high byte*).



Gambar 2.7. *Register Timer* 0

A.2. *Register Timer* 1

Timer 1 mengakses *byte* rendah dan *byte* tinggi dari *register* 16 bit. *Byte register* rendah disebut sebagai TL1 (*Timer* 1 *Low byte*) dan *byte* register tinggi di sebut sebagai TH1 (*Timer* 1 *High byte*)(Mazidi, 2000 : 158).

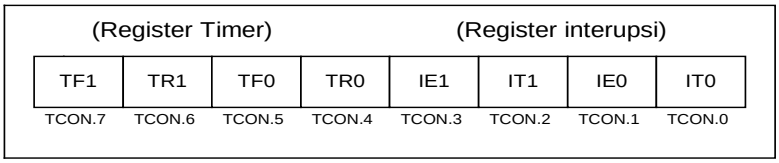


Gambar 2.8. *Register Timer* 1

A.3. *Timer* atau *Counter Control Register*

Register yang berfungsinya menghubungkan dengan *timer* ada empat yaitu TCON4, TCON5, TCON6 dan TCON7. *Register* ini bersifat *Addresstable*

bit sehingga *bit* TF1 disebut sebagai TCON.7, TR1 disebut sebagai TCON.6 dan seterusnya.



Gambar 2.9. Register TCON

TCON.7 atau TF1 : *Timer* 1 *overflow flag* di*set* jika *timer overflow*. *Bit* ini dapat dibersihkan oleh *software* .

TCON.6 atau TR1: 1 : *Timer* 1 aktif

0 : *Timer* 1 *non*-aktif

TCON.5 atau TF0: *Timer* 0 *overflow flag* di*set* jika *timer overflow*. *Bit* ini dapat dibersihkan oleh *software* .

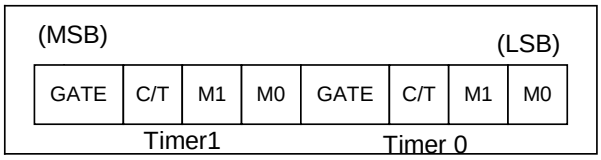
TCON.4 atau TR0: 1 : *Timer* 0 aktif

0 : *Timer* 0 *non*-aktif

TCON.3 sampai dengan TCON.0 dibahas pada bagian *interrupt* (Paulus Andi Nalwan, 2003 : 33).

A.4. *Register* TMOD (*timer mode*)

Dua *timer* 0 dan 1 yang menggunakan *register* TMOD, untuk *setting* macam *mode* operasi *timer*. TMOD adalah sebuah *timer* 8 *bit register* dimana 4 *bit* rendah untuk *timer* 0 dan 4 *bit* yang tinggi untuk *timer* 1.



Gambar 2.10. TMOD *Register*

A.4.1. C/T (*Clock* atau *Counter*)

Bit ini dalam *register* TMOD digunakan untuk memilih *timer* atau *counter*. Jika bit C/T = 0 yang digunakan adalah *timer*. Sumber *clock* dari *timer* menggunakan frekuensi dari kristal.

Meskipun macam sistem MCS-51 mempunyai kristal dengan frekuensi dari 10 MHz sampai 40 MHz, yang sering digunakan untuk proyek MCS-51 adalah kristal 11,0592 MHz (Mazidi, 2000 : 160)

A.4.2. Mengaktifkan dan Nonaktifkan *Timer*

Metode untuk mengaktifkan dan nonaktifkan dari sebuah *timer* ada dua cara yaitu secara *software* dan secara *hardware*. Untuk menggunakan metode *software* GATE=0, sedangkan kontrolnya terletak pada bit TRx saja, dimana TR0 untuk *timer* 0 dan TR1 untuk *timer* 1.

Selain menggunakan metode diatas untuk mengontrol *timer* dapat menggunakan metode *hardware* yaitu dengan menggunakan $\overline{\text{INTx}}$ (*interrupt*). Terlebih lebih dahulu *setting* GATE=1 setelah itu *timer* dapat dikontrol melalui $\overline{\text{INTx}}$. Dengan GATE = 1 dan TR0 = 1, ketika $\overline{\text{INT0}}$ = 1 maka *timer* 0 aktif, apabila $\overline{\text{INT0}}$ = 1 maka *timer* 0 nonaktif (I. Scott MacKensi, 1999 : 88).

A.4.3. M1 dan M0 (*mode* operasi *timer*)

M0 dan M1 digunakan untuk memilih *mode timer*. Ada 4 *mode* dalam *timer* yaitu *mode* 0 adalah *timer* 13 *bit*, *mode* 1 adalah *timer* 16 *bit*, *mode* 2 adalah *timer* 8 *bit auto reload* dan *mode* 3 adalah *timer mode split*. Untuk mencari nilai TLx dan THx dapat menggunakan rumus dibawah ini:

- Jumlah bit <= 8 *bit* :

$$T = (\text{jumlah bit} - TLx) \times \text{Clock} \quad (2.1)$$

Jumlah bit maksimal bernilai 255.

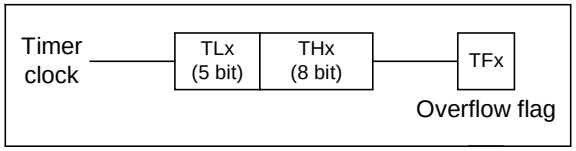
- 9 *bit* >= Jumlah *bit* <= 16 *bit*

$$T = (\text{jumlah bit} - THxTLx + 1) \times \text{Clock} \quad (2.2)$$

Jumlah bit maksimal bernilai 65535.

a) *Timer Mode* 0

Timer mode 0 adalah sebuah *timer* 13 *bit* yang disediakan oleh MCS-51. untuk *byte* tinggi menggunakan 8 *bit* MSB dari THx dan untuk *byte* rendah menggunakan 5 *bit* yang LSB dari TLx. Nilai yang diperbolehkan *timer* 13 *bit* adalah 0000H sampai dengan 1FFFH. Jika *timer* telah mencapai maksimal 1FFFH dan berubah menjadi 0000H maka *flag overflow* di *set* (TFx = 1).

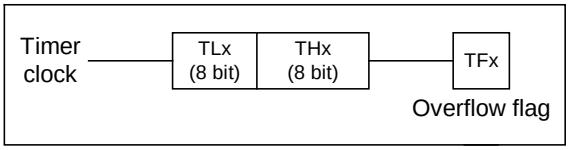


Gambar 2.11. *Mode* 0

b) *Timer Mode* 1

Mode 1 adalah *timer* 16 *bit* yang disediakan oleh MCS-51. *Timer* ini menggunakan *register byte* rendah dan *byte* tinggi (THx dan TLx). Jika menerima

pulsa *clock*, maka *timer* menghitung naik dari : 0000H,0001H,0002H dan lain-lain. Maksimal nilai yang diperbolehkan oleh *timer* 16 *bit* adalah 0000H sampai dengan FFFFH. *Overflow* terjadi apabila ada perubahan dari FFFFH ke 0000H (I.Scott MacKensi, 1999 : 86)

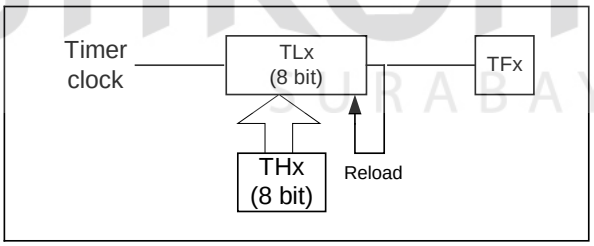


Gambar 2.12. *Mode* 1

c) *Timer Mode* 2

Mode 2 adalah 8 *bit auto reload*. *Register byte* tinggi THx digunakan untuk mengisi nilai kedalamnya, lalu dikirmkan secara automatis ke dalam TLx. Setelah itu TLx menghitung sampai maksimal FFH. *Overflow* terjadi apabila ada perubahan dari FFH ke 00H (I. Scott MacKensi, 1999 : 86).

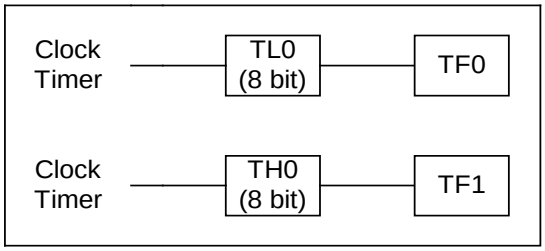
.



Gambar 2.13. *Mode* 2

d) *Timer Mode* 3

Mode 3 adalah *mode timer split*. *Timer* 0 pada *mode* 3 terpisah menjadi 2 *timer* 8 *bit*. TL0 *flag* timer *overflow* menggunakan TF0, sedangkan TH0 *flag timer overflow* menggunakan TF1. Pada *mode* 3 *timer* 1 tidak aktif , karena TF1 sudah digunakan oleh TH0 (I. Scott MacKensi, 1999 : 86).



Gambar 2.14. Mode 3

2.3 Komunikasi *Serial*

Komunikasi data berarti pengiriman data antara dua komputer, antara sebuah komputer dengan terminal, atau antara terminal dengan terminal yang lain.

Komunikasi data dapat dilakukan dengan dua cara: paralel dan *serial*. Dalam transfer data paralel, sering 8 atau lebih jalur (konduktor kabel) digunakan untuk mentransfer data ke suatu *device* yang berjarak hanya beberapa kaki. Contoh transfer paralel adalah *printer* dan *hard disk* yang menggunakan kabel dengan banyak jalur. Meskipun dalam kasus-kasus seperti ini banyak data bisa ditransfer dalam waktu singkat dengan menggunakan banyak kabel yang disusun paralel, tetapi jaraknya tidak bisa jauh dan biayanya relatif lebih mahal. Untuk mentransfer data ke suatu *device* yang terletak sejauh beberapa meter, digunakan metode *serial*. Dalam komunikasi *serial*, data dikirim satu bit dalam suatu waktu, berbeda dengan komunikasi paralel, dalam mana data dikirim satu *byte* atau lebih dalam suatu waktu. Kelebihan metode *serial* ini adalah selain ia dapat digunakan dalam jarak yang jauh, ia juga memerlukan biaya yang lebih murah jika dibandingkan dengan metode paralel.

Komunikasi data *serial* menggunakan dua metode, asinkron dan sinkron. Metode sinkron mentransfer suatu blok data (karakter) pada suatu waktu

sedangkan asinkron mentransfer suatu *byte* tunggal pada suatu waktu. Ada kemungkinan untuk membuat *software* untuk digunakan dengan metode di atas, tetapi programnya bisa membosankan dan panjang. Karena itu, ada IC khusus yang dibuat oleh banyak pabrik untuk komunikasi data *serial*. IC ini secara umum dikenal sebagai UART (*Universal Asynchronous Receiver-Transmitter*) dan USART (*universal synchronous-asynchronous receiver-transmitter*). *Port* COM pada IBM PC menggunakan UART 8250.

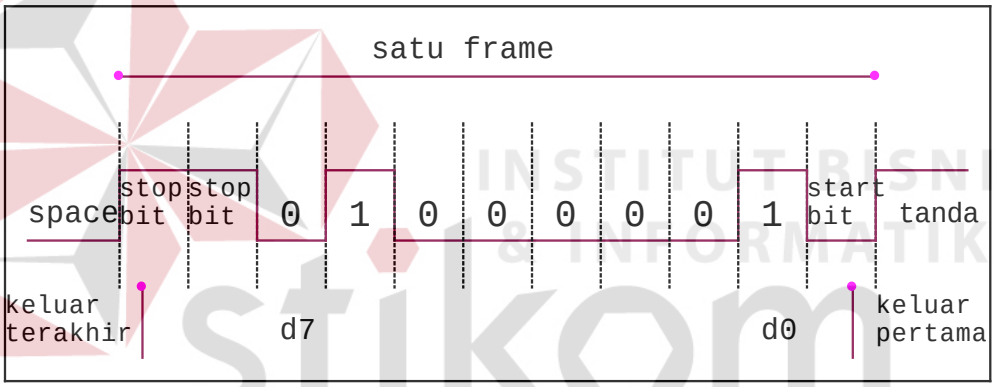
2.3.1 Transmisi *Half* dan *Full duplex*

Jika data bisa ditransmisikan dan diterima, itu disebut transmisi *duplex*. Berbeda dengan transmisi *simplex* seperti *printer*, yang mana komputer hanya mengirim data. Transmisi *duplex* bisa *half* atau *full duplex*, tergantung pada transfer datanya bisa sekaligus atau tidak. Jika data dapat ditransmisikan satu jalan pada suatu waktu, ini disebut *half duplex*. Dan bila data bisa melewati dua jalan pada waktu yang sama, itulah *full duplex*. Tentu saja, *full duplex* membutuhkan dua kabel konduktor (selain *ground*), satu untuk transmisi dan satu untuk penerimaan, agar bisa mentransfer dan menerima data secara sekaligus.

2.3.2 Komunikasi *serial* asinkron dan data *framing*

Data yang masuk pada akhir penerimaan dari jalur data dalam transfer data *serial* semuanya dalam program 0 dan 1, sangat sulit untuk memahami data kecuali pengirim dan penerima menyepakati seperangkat peraturan, sebuah protokol, tentang bagaimana data dipaketkan dan berapa banyak *bit constitute* sebuah karakter, dan kapan data mulai dan berakhir.

Komunikasi data *serial* asinkron digunakan secara luas untuk transmisi berorientasi-karakter, dan transfer data berorientasi-blok yang menggunakan metode sinkron. Dalam metode asinkron, setiap karakter diletakkan antara *start bit* dan *stop bit*, ini disebut *framing*. Dalam data *framing* untuk komunikasi asinkron, data, seperti karakter-karakter ASCII, dipaketkan di antara sebuah *start bit* dan sebuah *stop bit*. *Start bit* selalu satu *bit* tetapi *stop bit* bisa satu atau dua *bit*. *Start bit* selalu '0' (*low*) dan *stop bit* adalah '1' (*high*). Sebagai contoh, lihat gambar 2.15 dimana karakter ASCII "A", *biner* 0100 0001, di*frame* di antara *start bit* dan 2 *stop bit*. Perhatikan bahwa LSB-nya dikirim dulu.



Gambar 2.15. *Framing* ASCII "A" (41H)

Dalam gambar diatas, ketika tidak ada sinyal transfer yang bernilai 1 (*high*), yang disebut sebagai *mark* dan 0 (*low*) disebut sebagai *space*. Perhatikan bahwa transmisi dimulai dengan *start bit* yang diikuti oleh D0 (LSB), kemudian sisa *bit-bit* sampai MSB (D7), dan akhirnya, 2 *stop bit* yang menunjukkan akhir dari karakter "A".

Dalam komunikasi *serial* asinkron, *chip-chip peripheral* dan *modem* bisa diprogram untuk data selebar 5, 6, 7, atau 8 *bit*. Ini sebagai tambahan dari jumlah *stop bit*, 1 atau 2. Sementara dalam sistem yang lebih lama karakter-karakter

ASCII adalah 7 *bit* tapi dengan adanya karakter ASCII *extended*, dibutuhkan 8 *bit* untuk setiap karakter. Keyboard non-ASCII kecil menggunakan karakter-karakter 5 dan 6 *bit*.

Dalam beberapa sistem lama, disebabkan kelambatan peralatan mekanik yang menerima. 2 *stop bit* digunakan untuk memberikan peralatan tersebut cukup waktu untuk mengorganisasi dirinya sendiri sebelum transmisi dari *byte* berikutnya. Tetapi pada PC *modern* penggunaan 1 *stop bit* adalah umum. Dengan asumsi bahwa kita mentransfer file text dari karakter ASCII menggunakan 2 *stop bit* sehingga total untuk tiap karakter 11 *bit* yaitu 8 *bit* untuk program ASCII-nya, dan 1 dan 2 *bit* masing-masing untuk *start bit* dan *stop bit*. Oleh karena itu, untuk setiap karakter 8 *bit* ada 3 *bit* ekstra, atau lebih dari 25%.

Dalam beberapa sistem untuk menangani integritas data, *bit parity* dari *byte* karakter dimasukkan dalam data *frame*. Ini berarti bahwa untuk setiap karakter (7 *bit* atau 8 *bit*, tergantung pada sistem) kita punya *bit parity* tunggal sebagai tambahan dari *start bit* dan *stop bit*. *Bit parity* adalah ganjil atau genap. Dalam *bit parity*-ganjil jumlah total *bit* data, termasuk *bit parity* adalah ganjil dari 1-an. Serupa dengan itu, dalam sebuah *bit parity*-genap jumlah total *bit* data termasuk *bit parity* adalah genap. Sebagai contoh, karakter ASCII "A", *biner* 0100 0001, punya 0 untuk *bit parity*-genap. *Chip-chip* UART memungkinkan pemrograman *bit parity*-ganjil, *parity*-genap, dan pilihan tanpa *parity*, seperti terdapat dalam pembahasan berikutnya. Jika suatu sistem membutuhkan *parity*, *bit parity* ditransmisikan setelah MSB, dan diikuti oleh *stop bit*.

2.4 Transistor

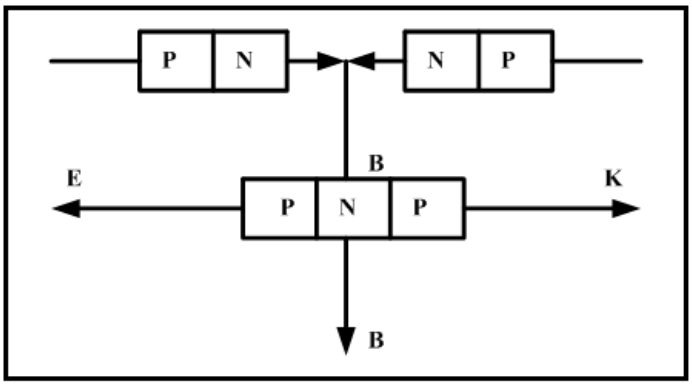
Transistor ditemukan oleh tiga orang sarjana Amerika, yang bernama *J. Barden, WH Brattain* dan *W Schockley* pada tahun 1948, nama transistor berasal dari kata *tranfer* dan resistor, *transfer* artinya mengalihkan atau membuat perubahan sedangkan resistor adalah suatu bahan yang tidak dapat menghantarkan arus listrik, jadi arti dari transistor adalah merubah bahan yang tidak dapat menghantarkan aliran listrik menjadi bahan penghantar atau setengah penghantar atau disebut juga bahan *semikonduktor*.

Transistor pada umumnya dipergunakan sebagai penguat atau *amplifier*, transistor sendiri sebenarnya adalah hasil pengembangan dari dua buah jenis dioda jenis PN dan NP yang dipertemukan sehingga akan membentuk satu elektroda yang berfungsi sebagai pengontrol pertemuan antara bahan PN dan NP tersebut. Prinsip terjadinya pertemuan kedua bahan tersebut seperti yang terlihat pada gambar 2.16, bila kedua bahan yang dipertemukan bahan jenis N nya maka akan diperoleh transistor jenis PNP, (PN + NP = PNP).

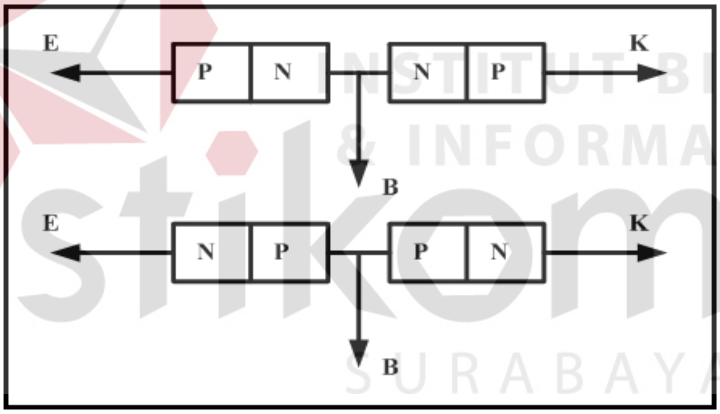
Sedangkan pada gambar 2.17 terlihat juga bila kedua bahan yang dipertemukan bahan jenis P nya, maka akan terbentuk transistor jenis NPN, NP + PN = NPN dari hasil pertemuan kedua bagan P dan N tersebut akan menghasilkan sebuah transistor yang memiliki 3 buah elektroda yang membentuk 3 buah kaki yaitu:

1. *Emitor* disingkat E.
2. *Basis* disingkat B.
3. *Colector* disingkat C.

Berdasarkan prinsip tersebut maka dalam dalam teknik elektronika, transistor tersebut dinamakan transistor *bipolar* jenis PNP dan NPN.



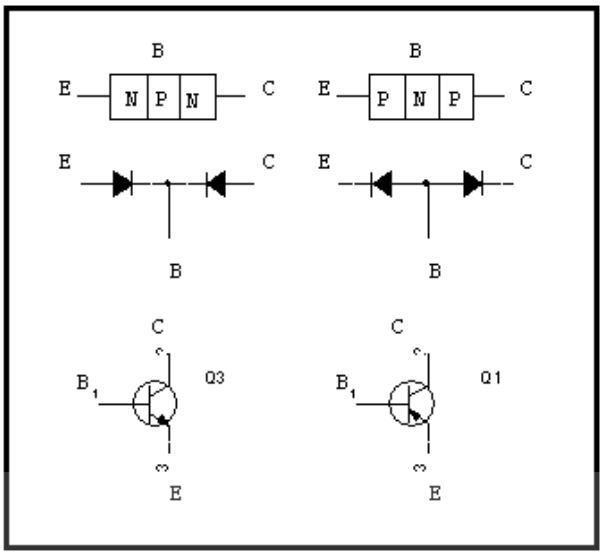
Gambar 2.16. Prinsip Transistor PNP



Gambar 2.17. Prinsip Transistor NPN

Transistor merupakan alat dengan tiga terminal *basis* (B), *colector* (C), *emitor* (E), transitor dapat dibedakan dalam dua jenis yaitu NPN dan PNP. Disebut transistor PNP karena terdiri dari bahan *semikonduktor* jenis P, N, dan P. dan disebut transistor NPN karena terdiri dari bahan *semikonduktor* jenis N, P, dan N. Transistor dapat dianggap sebagai dua jenis dioda yang saling dipertemukan. Transistor PNP memiliki simbol panah masuk, sedangkan

transistor NPN mempunyai simbul panah keluar seperti yang terlihat pada gambar 2.18.



Gambar 2.18. Simbol Transistor.

2.4.1. Transistor Pertemuan (*Junction Transistor*)

Azas kerja transistor seperti yang terlihat pada gambar 2.19 yaitu: (Wasito,1995:180).

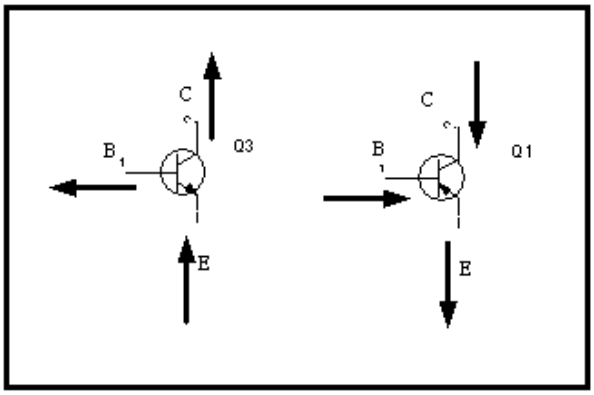
a. Akan ada arus diantara terminal-terminal *colector-emitor* (arus I_C), hanya apabila, ada arus yang mengalir diantara terminal-terminal *basis-emitor* (arus I_B).

b. Perbandingan antara kuat I_C dan kuat I_B disebut "bandingan-hantaran arus maju" (*forward current transfer ratio),* disingkat: $H_{FE,}$

dengan rumus:

$$H_{FE} = I_C : I_B \qquad (2.3)$$

c. Untuk penguatan frekuensi tinggi, ada transistor dengan harga H_{FE} = 1000 atau lebih



Gambar 2.19. Arah Arus pada Transistor

2.4.2 Parameter-parameter pada Transistor

Ada beberapa hal yang perlu diketahui mengenai parameter-parameter pada transistor yaitu: (Wasito,1995:180).

a. Parameter-parameter transistor tidaklah konstan, meskipun tipe sama, parameter dapat berbeda.
b. Parameter juga akan berlainan bagi arus yang berlainan, tapi dalam prakteknya dapat kita anggap bahwa parameter-parameter adalah konstan.
c. Konduktansi,

$G_M = i_e : V_{be}$ ma/V (ms) (2.4)

i_e : arus isyarat ac di antara *colector - emitor*

V_{be} **:** tegangan isyarat ac antara *basis - emitor*

2.5 Potensiometer

Potensiometer adalah transduser elektromekanik yang mengubah energi mekanik menjadi energi listrik. Perubahan mekanik disini adalah berkaitan dengan *displacement*, yang dapat berupa gerakan putar atau lurus. Masukan potensiometer adalah perpindahan mekanik, baik *linier* atau putaran. Ketika suatu

tegangan dipasang melintasi ujung tertentu dari potensiometer, tegangan keluaran, yang diukur melintasi ujung variabel dan *ground* adalah sebanding dengan perpindahan masukan, baik bersifat *linier* ataupun menurut beberapa hubungan *nonlinier*.

Potensiometer putar dapat ditemukan secara komersil dalam bentuk satu putaran atau banyak putaran, dengan gerak putaran yang terbatas ataupun tidak terbatas. Potensiometer disebut juga resistor variabel atau disingkat VR (*variable Resistor*).

Berdasarkan bahannya potensiometer dibedakan menjadi :

a. Potensiometer arang yang dapat diputar atau digeser.
b. Potensiometer kawat logam (bahan resistansi konduktif).

Berdasarkan perubahan nilai hambatannya potensiometer dibedakan menjadi :

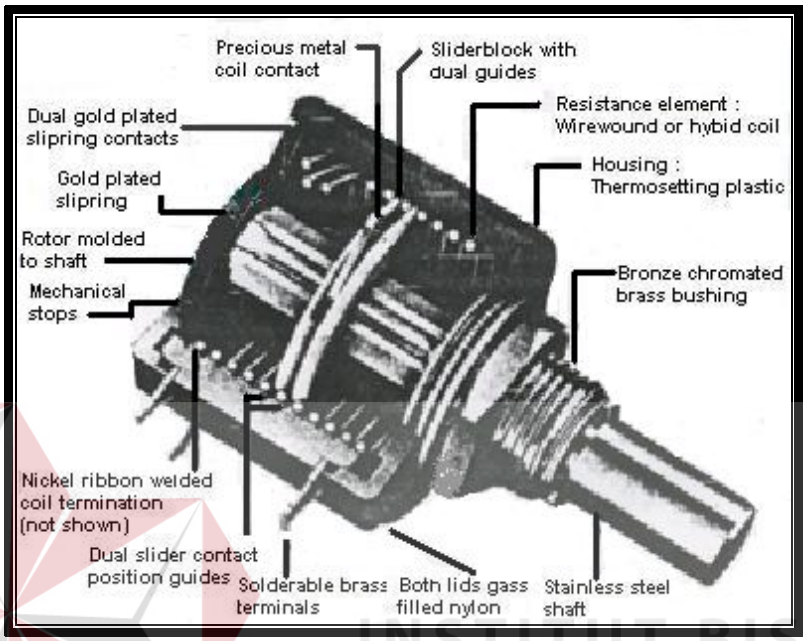
a. Potensiometer *linier*

Adalah potensiometer yang perubahan nilai hambatannya beraturan sesuai dengan sudut putarnya atau jarak gesernya. Biasa diberi kode B. misal diputar 10° hambatan A-C 100Ω dan diputar 20° hambatan A-C menjadi 200Ω dan seterusnya. Potensiometer *linier* inilah yang kemudian biasa digunakan sebagai sensor *level*.

b. Potensiometer *Log*

adalah potensiometer yang perubahan nilai hambatannya tidak beraturan. Biasa diberi kode A. Misal diputar 10° hambatan A-C 100Ω dan diputar 20° hambatan A-C menjadi 400Ω. Potensiometer *log* banyak digunakan pada pengatur volume, bass, treble, oleh karena itu jika kita putar potensiometer volume sedikit

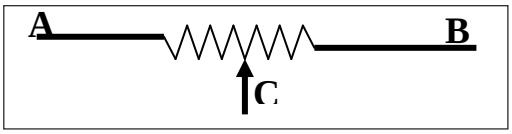
saja, suara sudah cukup keras, tapi bila putarannya sudah lewat dari separuh pertambahannya tidak terasa.



Gambar 2.20. Potensiometer 10 kali putaran

Berdasarkan pengemasannya potensiometer dibedakan menjadi :

a. Potensiometer *mono* adalah potensiometer yang terdiri dari satu VR dalam satu kemasan.
b. Potensiometer *stereo* adalah potensiometer yang terdiri dari 2 buah VR yang seporos, artinya bila tuas diputar, maka dua-duanya ikut berubah.
c. Potensiometer ber – CT (*Center Tap* = cabang tengah) adalah potensiometer yang mempunyai 1 kaki tambahan yang bila diukur terhadap kaki utama nilai resistansinya separuh nilai resistansi total.



Gambar 2.21. Representasi Potensiometer

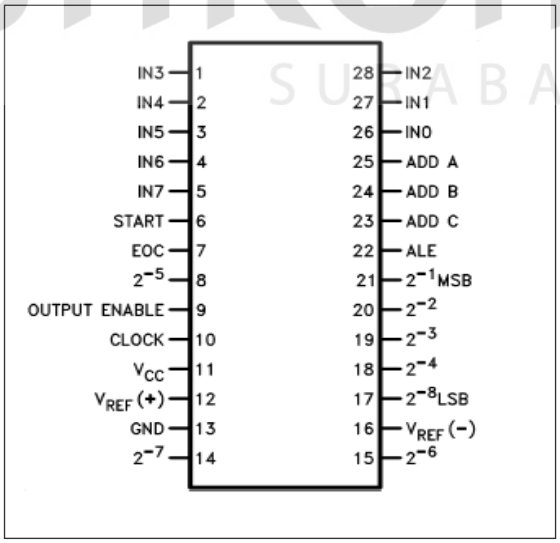
Potensiometer standar mempunyai tiga kaki (seperti gambar 2.21. diatas). Kaki A dan B merupakan kaki utama yang mempunyai hambatan tetap sedangkan kaki C adalah kaki yang hambatannya dapat berubah dalam *range* hambatan kaki utama (kaki A dan B). Besarnya hambatan kaki C dapat diukur relatif terhadap kaki A atau kaki B. misalnya hambatan di kaki utama adalah 1KΩ dan hambatan kaki C terhadap kaki A adalah 250Ω, maka besar nilai hambatan kaki C diukur relatif terhadap kaki B adalah 750Ω, yang merupakan hambatan diantara kaki C dan B yang sama dengan hasil pengurangan antara hambatan dikaki utama (1KΩ) dan hambatan antara kaki A dan C (250Ω).

2.6 *Analog To Digital Converter* (ADC)

Pada tugas akhir ini penggunaan ADC 0808 dipilih untuk pengiriman data delapan sensor karena dalam ADC 0808 sudah terdapat *multiplexer* dan juga terdapat ADC itu sendiri. Dengan delapan *input* yang dimiliki oleh ADC 0808, kita dapat mengambil contoh data sesuai dengan jalur *input* data, kanal yang diambil akan diubah ke delapan *bit digital* dengan waktu konversi sekitar 100 µs, kemudian dengan *buffer latch* yang tersedia, memungkinkan melakukan hubungan langsung dengan data *bus microcomputer*, besar tegangan referensi *external* sebesar 5 volt DC dan besar sinyal *clock* 10 sampai dengan 1280 kHZ, secara *internal* ADC 0808 menggunakan teknik *successive aproximation* untuk konversi *analog* to *digital*. *Register successive aproximation*, R-2R *resistor ladder* dan

komparator *analog* termasuk dalam *chip* dengan adanya tegangan referensi 5v maka tegangan *input* dibatasi antara 0v sampai dengan 5v.

Proses konversi diawali dengan memberi pulsa ALE (*Address Latch Enable atau* sinyal penahan alamat) dengan *input* aktif *High*. Proses ini adalah mengirim sinyal alamat yang dipilih ke *multiplexer* dan melewatkan sinyal dari salah satu delapan kanal *input* menuju komparator, pemberian pulsa *start* digunakan untuk mengawali konversi *output* EOC (*end of convertion*) berada pada keadaan *low* saat *register successive aproximation internal* mengumpulkan kode *biner* delapan *bit rising edge* (transisi '0' (*low*) ke '1'(*high*)) dari EOC menandakan bahwa konversi berakhir dan *Byte* (8 *bit*) data siap dibaca, pemberian pulsa aktif (*high*) pada OE (*output enable*) menempatkan data tersebut pada delapan jalur data *bus* yang dalam keadaan normal adalah *tri-state*, sekali konversi diawal, maka *output* EOC dapat dideteksi melalui metode *polling* (*scanning* secara terus menerus) maupun menggunakan sinyal interupsi ke *microcontroller*.



Gambar 2.22. Pin Diagram ADC0808