BAB II
TINJAUAN PUSTAKA

2.1 Braille

Braille adalah sejenis sistem tulisan sentuh yang digunakan oleh penyandang cacat tuna netra. Sistem ini awalnya dirancang oleh Louis Braille yang berkewarganegaraan Perancis yang buta sejak kecil. Ketika usia 15 tahun, Louis Braille mengubah bentuk tulisan latin biasa menjadi tulisan sentuh yang dapat digunakan tentara untuk membaca dalam gelap. Sistem ini dinamakan sistem Braille dengan tujuan mendapatkan kemudahan dalam membaca. Namun saat itu Braille tidak mempunyai huruf W, tetapi sekarang Braille mempunyai huruf W. Ada beberapa versi tulisan Braille dibeberapa Negara, yaitu Standar Braille, American Modified Braille, US Computer Braille, US Extended Computer Braille, Euro Computer Braille, ISO 8859-I Braille, Russian Braille, Greek Braille, Hebrew Braille, Arab Braille, Japanese Braille, Korean Braille, Chinese Braille, Braille ASCII, Unicode.

Selain standar Braille, umumnya semua versi diatas memiliki perbedaan dalam hal, yaitu membedakan antara huruf besar dan huruf kecil dalam satu blok, membedakan antara huruf besar dan angka dalam satu blok, banyaknya jenis karakter yang dapat diterjemaahkan kedalam kode Braille dan jumlah dot yang digunakan (Standar Braille menggunakan 6 dot sedangkan versi lama Braille menggunakan 8 dot) [6].

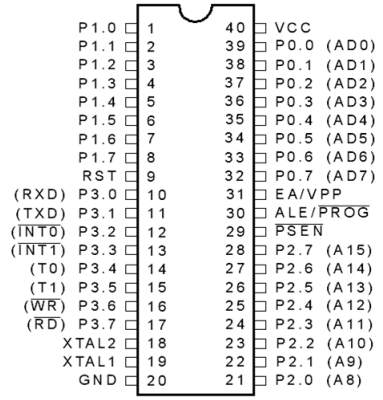
2.2 Mikrokontroler AT89C51

Mikrokontroler AT89C51 adalah sebuah mikrokontroler dengan 4 Kbyte Flash PEROM (*Programmable and Erasable Read Only memory*). Mikrokontroler AT89C51 memiliki memori dengan teknologi *non volatile* (memori dapat di isi ulang atau dapat di hapus berkali-kali). Memori ini dapat digunakan untuk menyimpan instruksi berstandar MCS-51 sehingga memungkinkan mikrokontroler

ini untuk bekerja dalam mode single chip operation yang tidak memerlukan memori eksternal untuk menyimpan *source code* tersebut. Program dapat dimasukan dan dihapus sampai dengan 1000 kali[4].

2.2.1 Deskripsi Pin

Mikrokontroler AT89C51 memiliki 40 kaki dan 32 kaki diantaranya merupakan port paralel yang terdiri dari port P0, P1, P2 dan P3 yang masing-masing memiliki 8 port.



Gambar 2.1. Konfigurasi Pin ATMEL AT89C51

Adapun nama dan fungsi dari setiap pin pada mikrokontroler AT89C51:

a. Port P0 (pin 39 sampai dengan pin 32)

Port P0 berfungsi sebagai I/O (Input/ Output) biasa atau *low order multiplex address data* (menerima kode byte pada saat *flash programming*). Pada fungsi sebagai I/O biasa port ini dapat memberikan *output sink* ke delapan buah TTL input atau dapat diubah sebagai input dengan memberikan logika 1 pada port tersebut. Pada fungsi sebagai *low order multiplex address data*, port ini akan mempunyai internal pull up terutama pada saat verifikasi program.

b. Port P1 (pin 1 sampai dengan pin 8)

Port P1 berfungsi sebagai I/O biasa atau menerima *low order address bytes* pada saat *flash programming*. Port ini mempunyai internal pull up dan berfungsi sebagai input dengan memberikan output sink keempat buah input TTL.

c. Port P2 (pin 21 sampai dengan pin 28)

Port P2 berfungsi sebagai I/O biasa atau high order address, pada saat mengakses memori secara 16 bit (Movx @DPTR). Pada saat mengakses memori secara delapan bit, (Mov @Rn), port ini akan mengeluarkan isi dari P2 *Special Function Register* (SFR). Port ini mempunyai internal pull up dan berfugsi sebagai input dengan memberikan logika 1. Sebagai output, port ini dapat memberikan output sink keempat buah input TTL.

d. Port P3 (pin 10 sampai dengan pin 17)

Sebagai I/O biasa, Port P3 mempunyai sifat yang sama dengan port P1 dan port P2, sedangkan fungsi spesial dari port-port P2 adalah:

Tabel 2.1. Fungsi-fungsi alternatif pada *Port* 3

Pin	Port	Nama	Fungsi
Pin 10	P3.0	RXD	Port serial input
Pin 11	P3.1	TXD	Port serial output
Pin 12	P3.2	INT0	Port External Interupt 0
Pin 13	P3.3	INT1	Port External Interupt 1
Pin 14	P3.4	T0	Port external timer 0 input
Pin 15	P3.5	T1	Port external timer 1 input
Pin 16	P3.6	WR	External Data Memori write strobe
Pin 17	P3.7	RD	External Data Memori write strobe

e. Pin 9

Pin reset (RST) akan aktif dengan memberikan input high selama 2 *cycle.*

f. Pin 18

Pin 18 atau Pin XTAL2 untuk output oscillator.

g. Pin 19

Pin 19 atau Pin XTAL1 untuk input oscillator.

h. Pin 20

Pin 20 berfungsi sebagai ground dari mikrokontroler AT89C51.

i. Pin 29

Pin 29 atau PSEN berfungsi pada saat mengeksekusi program yang terletak pada memori eksternal. PSEN akan aktif dua kali setiap *cycle*.

j. Pin 30

Pin 30 atau ALE dapat berfungsi sebagai *Address Latch Enable* yang me-latch low byte address pada saat mengakses memori eksternal, sedangkan pada saat flash programming (PROG) berfungsi sebagai pulsa input untuk operasi normal, ALE akan mengeluarkan sinyal clock sebanyak 1/16 frekuensi oscillator kecuali pada saat mengakses memori eksternal sinyal clock pada pin ini dapat pula di *disable* dengan mengeset bit 0 dari *special function register* di alamat 8EH. ALE hanya akan aktif pada saat mengakses memori eksternal (MOVX dan MOVC).

k. Pin 31

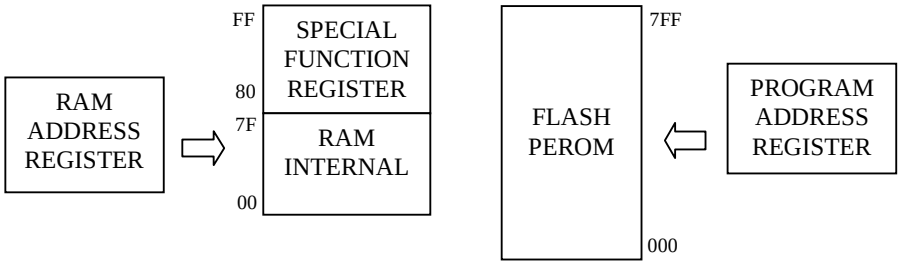
Pin 31 atau EA pada kondisi low, pin ini akan berfungsi sebagai EA yaitu mikrokontroler akan menjalankan program yang ada pada memori eksternal setelah sistem direset. Jika berkondisi high, pin ini akan berfungsi untuk menjalankan program yang ada pada memori internal. Pada saat *flash programming* pin akan mendapat tegangan 12 volt (VP).

l. Pin 40

Pin 40 berfungsi sebagai VCC pada mikrokontroler AT89C51.

2.2.2 Struktur Memori

Struktur memori pada mikrokontroler AT89C51 dapat dilihat pada gambar dibawah ini:



Gambar 2.2. Alamat RAM Interal dan Flash PEROM

1. RAM internal, memori sebesar 128 byte yang biasanya digunakan untuk menyimpan variabel atau data yang bersifat sementara.

   AT89C51 mempunyai struktur memori yang terpisah antara RAM internal dan Flash PEROM. RAM internal dialamati oleh RAM *Address Register* sedangkan Flash PEROM yang menyimpan instruksi-instruksi MCS51 yang dialamati oleh *Program Address Register*. RAM internal terdiri atas:

   a. *Register Banks*

      AT89C51 mempunyai 8 buah register dari R_0 hingga R_7. Register register tersebut selalu terletak pada alamat 00H hingga 07H pada setiap kali direset. Posisi R_0 hingga R_7 dapat dipindah ke Bank 1 (08H hingga 0FH), Bank 2 (10H hingga 17H), Bank 3 (18H hingga 1FH) dengan mengatur bit RS0 dan RS1.

   b. *Bit Addressable* RAM

      RAM pada alamat 20H hingga 2FH dapat diakses secara pengalamatan bit, sehingga hanya sebuah instruksi setiap bit dapat di *set, clear,* AND dan OR.

   c. RAM keperluan umum

      RAM pada alamat 30H hingga 7FH dapat diakses dengan pengalamatan langsung maupun tak langsung.

2. *Special Function Register (SFR)*

   Memori yang berisi register-register yang mempunyai fungsi-fungsi khusus yang disediakan oleh mikrokontroler seperti *timer, serial* dan lain-lain. *Special Function Register (SFR)* yang dimiliki oleh AT89C51 sebanyak 21 SFR yang terletak pada alamat 80H hingga FFH. Beberapa dari SFR mampu dialamati dengan pengalamatan bit. Berikut ini beberapa register pada SFR, yaitu:

Tabel 2.2. Alamat *Special Function Register*

Register	Mnemonic	Alamat
P0	*Port* 0 *Latch*	80H
SP	*Stack Pointer*	81H

Tabel 2.2. Lanjutan

DPTR	*Data Pointer*	82H-83H
DPL	*Data Pointer Low Byte*	82H
DPH	Data *Pointer High Byte*	83H
PCON	*Power Control*	87H
TCON	*Timer/Counter Control*	88H
TMOD	*Timer/Counter Mode Control*	89H
TL0	*Timer/Counter 0 Low Byte*	8AH
TL1	*Timer/Counter 1 Low Byte*	8BH
TH0	*Timer/Counter 0 High Byte*	8CH
TH1	*Timer/Counter 1 High Byte*	8DH
P1	*Port* 1 *Latch*	90H
SCON	*Serial Port Control*	98H
SBUF	*Serial Data Port*	99H
P2	*Port* 2 *Latch*	A0H
IE	*Interrupt Enable*	A8H
P3	*Port* 3 *Latch*	B0H
IP	*Interrupt Priority Control*	B8H
PSW	*Program Status Word*	D0H
ACC	*Accumulator*	E0H
B	*Register* B	F0H

3. *Flash* PEROM

Memori yang digunakan untuk menyimpan instruksi-instruksi MCS-51 dialamati oleh *Program Address Register* (*Register* Alamat Program). AT89C51 memiliki 4 Kb *Flash* PEROM yang menggunakan *Atmel's High-Density Non Volatile Technology*.

Program yang ada pada *Flash* PEROM akan dijalankan jika pada saat sistem di-*reset*, pin EA/VP berlogika satu maka mikrokontroler aktif berdasarkan program yang ada pada *Flash* PEROM-nya. Namun jika pin EA/VP berlogika nol, mikrokontroler aktif berdasarkan program yang ada pada memori eksternal.

2.2.3 *Timer* Mikrokontroler AT89C51

AT89C51 mempunyai dua buah *timer*, yaitu *Timer* 0 dan *Timer* 1, setiap *timer* terdiri dari 16 bit *timer* yang tersimpan dalam dua buah *register* yaitu THx untuk *Timer High Byte* dan TLx untuk *Timer Low Byte* yang keduanya dapat berfungsi sebagai *counter* maupun sebagai *timer*. Perbedaan terletak pada sumber *clock* dan aplikasinya.

Jika *timer* mempunyai sumber *clock* dengan frekuensi tertentu yang sudah pasti sedangkan *counter* mendapat sumber *clock* dari pulsa yang hendak dihitung jumlahnya. Aplikasi dari *counter* atau penghitung biasa digunakan untuk aplikasi menghitung jumlah kejadian yang terjadi dalam periode tertentu sedangkan *timer* atau pewaktu biasa digunakan untuk aplikasi menghitung lamanya suatu kejadian yang terjadi. Perilaku dari *register* THx dan TLx diatur oleh *register* TMOD dan *register* TCON. *Timer* dapat diaktifkan melalui perangkat keras maupun perangkat lunak. Register-register yang digunakan untuk pengaturan *timer*, yaitu:

1. *Timer Mode Register* (TMOD) di alamat 89H

   Pada register TMOD terdapat 4 bit pertama untuk *setting Timer* 0 dan 4 bit berikutnya untuk *setting Timer* 1.

MSB *LSB*

Gate(1)	C/T(1)	M1(1)	M0(1)	Gate(0)	C/T(0)	M1(0)	M0(0)
TIMER 1				*TIMER 0*			

Gambar 2.3. Register TMOD

Keterangan:

- *Gate* : *Timer* akan berjalan bila bit ini set dan INT0 (untuk *Timer* 0) atau INT1 (untuk Timer 1) berkondisi high.
- C/T : 1 = *Counter* dan 0 = *Timer*
- M1 & M0 : Untuk memilih *mode timer*.

Untuk pemilihan *mode timer* yang akan digunakan dapat dilihat pada tabel berikut:

Tabel 2.3. *Mode Timer*

M1	M0	*Mode*	Operasi
0	0	0	*Timer* 13 bit
0	1	1	*Timer/counter* 16 bit
1	0	2	*Timer* 8 bit di mana nilai *timer* tersimpan pada TLx. *Register* THx berisi nilai isi ulang yang akan dikirim ke TLx setiap *overflow*.
1	1	3	Pada *mode* ini, AT89S51 bagaikan memiliki 3 buah *timer*. *Timer* 0 terpisah menjadi 2 buah *timer* 8 bit (TL0 – TF0 dan TH0 – TF1) dan *timer* 1 tetap 16 bit.

1. THx dan TLx

   THx dan TLx adalah *register* yang menunjukkan nilai dari *Timer* (x adalah nomor Timer). Masing-masing *Timer* mempunyai dua buah *register* yaitu THx untuk *high* byte dan TLx untuk *low* byte.

   TH0 : Timer 0 *High* Byte terletak pada alamat 8AH

   TL0 : Timer 0 *Low* Byte terletak pada alamat 8BH

   TH1 : Timer 1 *High* Byte terletak pada alamat 8CH

   TL1 : Timer 1 *Low* Byte terletak pada alamat 8DH

2. *Timer Control Register* (TCON)

   Pengontrolan kerja *timer/counter* diatur oleh *register* TCON. Pada *register* TCON hanya ada 4 bit yang mempunyai fungsi hubungan dengan *timer*, yaitu TCON.4, TCON.5, TCON.6 dan TCON.7

TCON.7	**TCON.6**	**TCON.5**	**TCON.4**	**TCON.3**	**TCON.2**	**TCON.1**	**TCON.0**
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Gambar 2.4 *Register* TCON

Keterangan:

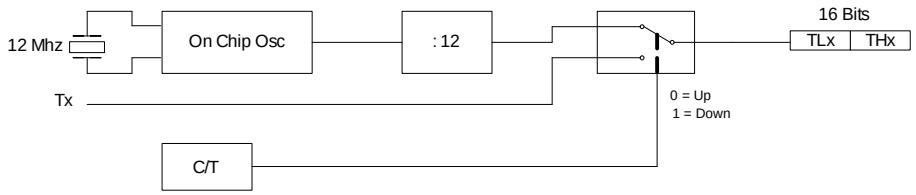
TCON.7 atau TF1: *Timer* 1 *Overflow Flag* yang akan set bila *timer* *overflow*. Bit ini dapat di *clear* oleh *software* atau *hardware* pada saat program menuju ke alamat yang ditunjuk oleh *interrupt vector.*

TCON.6 atau TR1: 1 = *Timer* 1 aktif, 0 = *Timer* 1 non aktif

TCON.5 atau TF0: *Timer* 1 *Overflow Flag* yang akan set bila *timer* *overflow*. Bit ini dapat di *clear* oleh *software* atau *hardware* pada saat program menuju ke alamat yang ditunjuk oleh *interrupt vector.*

TCON.4 atau TR0: 1 = *Timer* 1 aktif, 0 = *Timer* 1 non aktif

2.2.3.1. Prinsip Kerja *Timer* Mikrokontroler AT89C51



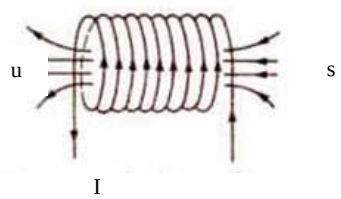
Gambar 2.5. Operasi *Timer*

Seperti yang telah disebutkan di atas, *timer* mempunyai dua sumber clock untuk beroperasi, yaitu sumber *clock* internal dan sumber *clock* eksternal. Jika *timer* menggunakan sumber *clock* eksternal, maka bit C/T harus di-*set* atau berkondisi *high*, saklar akan menghubungkan sumber clock *timer* ke pin Tx (To untuk *Timer* 0, Tı untuk *Timer* 1). Apabila sumber clock internal digunakan, input *clock* berasal dari osilator yang telah dibagi 12, maka bit C/T harus di-*clear* atau berkondisi *low* sehingga saklar akan menghubungkan sumber clock *timer* ke osilator yang telah dibagi 12[9].

2.3. Solenoid

Solenoid adalah alat yang dapat mengkonversi sinyal elektrik atau arus listrik menjadi gerak mekanik. Solenoid dibuat dari kumparan dan inti besi yang dapat digerakkan yang berfungsi sebagai aktuator pada alat bantu kode braille.

Secara skematik bentuk dari solenoida dapat dilihat pada gambar dibawah ini, di mana solenoid terdiri dari n buah lilitan kawat berarus listrik I, medan magnet yang dihasilkan memiliki arah seperti pada gambar, di mana kutub utara magnet mengikuti aturan tangan kanan 1.



Gambar 2.6.a. Solenoid dengan banyak lilitan n



Gambar 2.6.b Solenoid

Kuat medan magnet pada solenoid dengan jumlah lilitan persatuan panjang n adalah dapat dihitung dengan menggunakan persamaan dibawah ini:

$B = \mu_0 n.I$ ……... (2.1)

dimana:

$n = \frac{N}{l}$ …….. (2.2)

Keterangan:

B = Medan magnet

μ_0 = Kostanta permeabilitas udara

n = Jumlah lilitan persatuan panjang

N = Jumlah lilitan

l = Panjang lilitan

I = Arus

2.3.1. Cara Kerja Solenoid

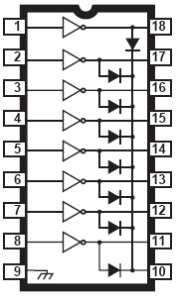
Prinsip kerja dari *Solenoid* berdasarkan pada penghantar yang membawa arus kedalam kumparan sehingga kumparan akan menimbulkan medan magnet. Medan magnet ini dibuat sedemikian rupa sehingga keadaannya selalu tolak menolak antara medan magnet.Sedangkan untuk mengubah arah putaran s*olenoid* dengan cara membalikkan polaritas sumber tegangannya.

2.3.2. Driver Solenoid IC ULN2803

IC ULN2803 merupakan *driver* solenoid dengan desain *dual in-lines packages*. IC ULN2803 mampu beroperasi pada tegangan 4,5 V sampai 36 V dan *output* yang keluar dari IC ULN2803 ini akan bertentangan dengan *input* yang diberikan. Satu buah Transistor Darlington digunakan untuk mengontrol 6 buah *solenoid*. Arus keluaran dari IC ULN2803 ini sebesar 500mA pada setiap *chanel*-nya[5].



Gambar 2.7. IC ULN2803



Gambar 2.8. Skema dalam IC ULN2803

2.4. Komponen Instrumentasi

Komponen instrumentasi yang digunakan untuk membangun sistem alat bantu baca tunanetra mencakup IC *regulator*, kapasitor, dioda, kristal.

2.4.1. LM78xx (IC *Regulator* Tegangan)

Seperti diperlihatkan gambar 2.9, IC ini mempunyai tiga kaki yang digunakan sebagai komponen pendukung dari Vcc untuk menghasilkan tegangan 5, 6, 8, 9, 10, 12, 15, 18, 24Volt. Simbol ‘xx’ pada gambar 2.8 menandakan besar tegangan yang dihasilkan seperti untuk menghasilkan tegangan keluaran 5Volt maka nilai untuk menandakan simbol ‘xx’ tersebut adalah 05, yang berarti IC yang digunakan adalah LM7805. IC *regulator* ini berfungsi untuk menstabilkan tegangan. Penerapan IC ini mengharuskan Vi > Vo. IC *regulator* yang digunakan yaitu LM7805 untuk menghasilkan tegangan keluaran 5Volt dan LM78012 untuk menghasilkan tegangan keluaran 12Volt.

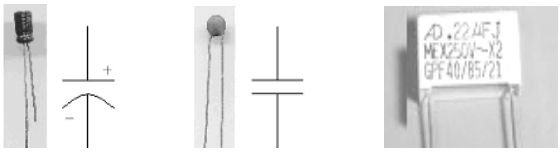


Gambar 2.9. IC LM78xx

2.4.2. Kapasitor

Kapasitor disebut juga *Kondensator*, yaitu komponen yang berfungsi untuk menyimpan muatan/tegangan listrik atau menahan arus searah. Kapasitor ELCO (*Electrolit Capasitor*) terbuat dari keping aluminium dan elektrolit yang dikandung dalam lembaran kertas berpori. Plat aluminium bersifat sebagai isolator dan elektrolit berfungsi sebagai konduktor. Kapasitor ELCO memiliki kekutuban atau polaritas yaitu tanda positif dan tanda negatif. Jika dalam pemasangan kutub-kutub ELCO terbalik maka kapasitor akan rusak. Karena tidak terlalu akurat dan bersifat *elektronik marginal properties*, maka kapasitor jenis ini tidak baik digunakan dalam rangkaian yang berhubungan dengan transmisi sinyal HF. Jadi, kapasitor ELCO ini lebih baik digunakan untuk *filter ripple, timing circuit.* Kapasitor keramik secara internal tidak dibangun sebagai koil, sehingga cocok untuk penggunaan aplikasi tinggi. Kapasitor ini bersifat non-polaritas atau tidak memiliki tanda positif dan tanda negatif sehingga dapat dipasang bolak-balik. Biasanya digunakan untuk *by-pass* sinyal frekuensi tinggi ke *ground.* Kapasitor keramik bergantung pada suhu lingkungan.

Metalized polyester capasitor dibuat dari *film dielectric* dan biasa disebut dengan Kapasitor Milar. Mempunyai kualitas yang baik, *low drift*, temperaturnya stabil. Secara fungsional, kapasitor milar ini sama dengan kapasitor non polaritas lain. Untuk satuan dari ELCO adalah mikro, kapasitor keramik adalah piko dan kapasitor milar adalah nano. Simbol dan contoh dari kapasitor diperlihatkan oleh gambar 2.10.



(a) (b) (c)

Gambar 2.10. (a) ELCO, (b) Kapasitor Keramik dan (c) Kapasitor Milar

2.4.3. Dioda

Dioda ialah suatu komponen semikonduktor yang memiliki sifat yang unik. Dioda hanya mengizinkan arus mengalir dalam satu arah saja, jika

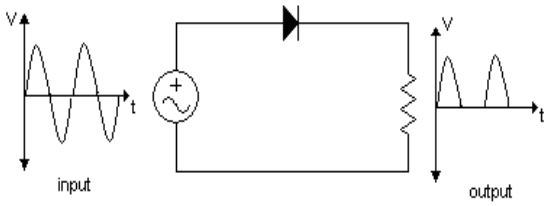
dipakai sebagai penyearah dengan kata lain dioda dapat mengubah sinyal AC menjadi sinyal DC.



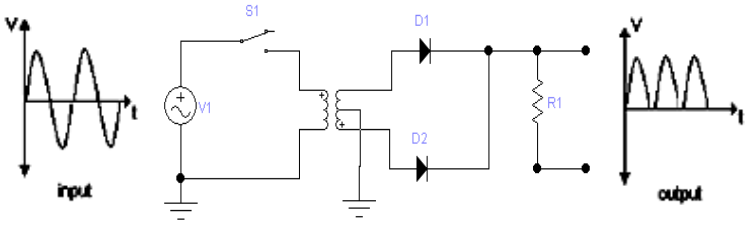
(a) (b)

Gambar 2.11. Dioda (a).Fisik, (b).Simbol

Gambar 2.11 menunjukkan sebuah dioda dan simbolnya. Bagian dioda terdiri dari 2 bagian yaitu anoda (positif) dan katoda (negatif). Seperti telah dijelaskan di atas, bahwa dioda hanya mengalirkan arus satu arah saja. ini berarti selama siklus negatif dari tegangan masukan, tidak akan ada arus yang melewati dioda.



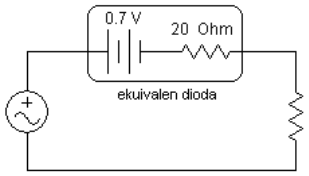
Gambar 2.12.a Karakteristik dioda



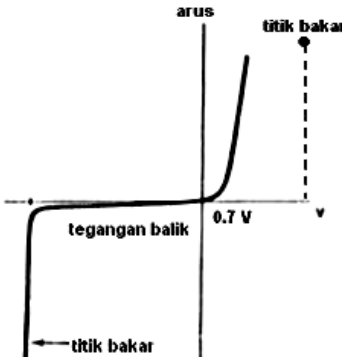
Gambar 2.12.b Rangkaian Dioda Gelombang Penuh

Pada gambar 2.12.b, selama setengah siklus positif tegangan sekunder, dioda atas dibias forward dan dioda bawah dibias reverse, maka arus akan melalui dioda atas, resistor beban, setengah lilitan atas. Setengah siklus negative, arus melalui dioda bawah, resistor beban, dan stengah lilitan bawah. Arus beban adalah dalam arah yang sama, Sehingga tegangan beban merupakan sinyal gelombang penuh. Hal-hal lain yang perlu diperhatikan ialah, pada saat tegangan sumber melewati dioda, terjadi penurunan tegangan sekitar 0.7 Volt. jadi bila tegangan input ialah 5 volt, tegangan

keluarannya menjadi 4.3 Volt. Selama tegangan input kurang dari 0.7 Volt, tidak akan ada arus yang dapat mengalir, dan setelah tegangan masukan melebihi 0.7 Volt, arus akan naik dengan cepat. Rangkaian *ekuivalen* dan *grafik* arusnya tampak seperti gambar 2.13.



(a)



(b)

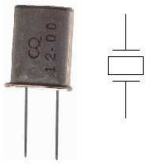
Gambar 2.13. Rangkaian Ekivalen Dioda dan Grafiknya

Sesaat setelah tegangan input melewati 0.7 Volt, arus akan mulai mengalir, tapi yang perlu diperhatikan ialah tegangan dan arus yang diberikan ke dioda tidak boleh terlalu tinggi karena akan menyebabkan kerusakan pada dioda atau terbakar, umumnya dioda dapat bertahan hingga 50 Volt dan arus 1 Ampere, tentunya tergantung pada jenis dioda yang kita pakai, itulah sebabnya mengapa dioda banyak digunakan sebagai penyearah tegangan.

2.4.4. Kristal

Kristal merupakan pembangkit *clock* internal yang menentukan rentetan kondisi-kondisi (*state*) yang membentuk sebuah siklus mesin mikrokontroler. Siklus mesin tersebut diberi nomor S1 hingga S6, masing-masing kondisi panjangnya 2 *periode* osilator, dengan demikian satu siklus mesin paling lama dikerjakan dalam 12 *periode* osilator. Satuan kristal biasanya dalam skala mega yaitu antara 4MHz sampai 24MHz dengan

bentuk dan simbol seperti yang diperlihatkan oleh gambar 2.14. Pada pembuatan alat bantu baca tuna netra ini menggunakan kristal dengan frekuensi 12MHz untuk mempermudah dalam perhitungan *timer*.



Gambar 2.14. Kristal

2.5. Mode Transmisi Port Serial

Ada 2 macam cara komunikasi data serial yaitu Sinkron dan Asinkron. Pada komunikasi data serial sinkron, clock dikirimkan bersama sama dengan data serial, tetapi clock tersebut dibangkitkan sendiri-sendiri baik pada sisi pengirim maupun penerima. Sedangkan pada komunikasi serial asinkron tidak diperlukan clock karena data dikirimkan dengan kecepatan tertentu yang sama baik pada pengirim/penerima[10].

Pada IBM PC kompatibel port serialnya termasuk jenis asinkron. Komunikasi data serial ini dikerjakan oleh UART (*Universal Asynchronous Receiver Transmitter*). Pada UART, kecepatan pengiriman data dan fase *clock* pada sisi *transmitter* dan sisi *receiver* harus sinkron. Untuk itu diperlukan sinkronisasi antara *transmitter* dan *receiver*. Hal ini dilakukan oleh bit "*Start*" dan bit "*Stop*". Ketika saluran transmisi dalam keadaan idle, output UART adalah dalam keadaan logika "1".

Ketika *transmitter* ingin mengirimkan data, output UART akan diset dulu ke logika "0" untuk waktu satu bit. Sinyal ini pada *receiver* akan dikenali sebagai sinyal "*Start*" yang digunakan untuk mensinkronkan fase *clock*-nya sehingga sinkron dengan fase *clock transmitter*. Selanjutnya data akan dikirimkan secara serial dari bit yang paling rendah (bit0) sampai bit tertinggi. Selanjutnya akan dikirimkan sinyal "*Stop*" sebagai akhir dari pengiriman data serial. Sebagai contoh misalnya akan dikirimkan data huruf "A" dalam format ASCII atau sama dengan 41 hexa.



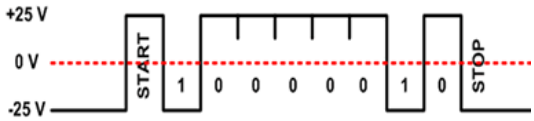
2.15. Pengiriman huruf A tanpa bit paritas

Kecepatan transmisi (*baud rate*) dapat dipilih bebas dalam rentang tertentu. *Baud rate* yang umum dipakai adalah 110, 135, 150, 300, 600, 1200, 2400, dan 9600 (bit/perdertik). Dalam komunikasi data serial, *baud rate* dari kedua alat yang berhubungan harus diatur pada kecepatan yang sama. Selanjutnya harus ditentukan panjang data (6,7 atau 8 bit), paritas (genap, ganjil, atau tanpa paritas) dan jumlah bit "Stop" (1, 1 ½ , atau 2 bit). Berikut ini adalah karakteristik sinyal *port* serial, *flow control* dan konfigurasi *port* serial.

Karakteristik Sinyal Port Serial

Standar sinyal komunikasi serial yang banyak digunakan adalah Standar RS232 yang dikembangkan oleh *Electronic Industri Association* (EIA/TIA) yang pertama kali dipublikasikan pada tahun 1962. Ini terjadi jauh sebelum IC TTL populer sehingga sinyal ini tidak ada hubungan sama sekali dengan level tegangan IC TTL. Standar ini hanya menyangkut komunikasi antara *Data Terminal Equipment* (DTE) dengan alat-alat pelengkap komputer *Data Circuit Terminating Equipment* (DCE). Standar sinyal RS232 memiliki ketentuan level tegangan sebagai berikut :

- Logika 1 disebut '*Mark*' terletak antara -3 Volt sampai -25 Volt
- Logika '0' disebut '*space*' terletak antara +3 Volt sampai +25 Volt.
- Daerah tegangan antara -3 Volt sampai +3 Volt adalah *invalid level*, yaitu daerah tegangan yang tidak memiliki *level* logika pasti sehingga harus dihindari. Demikian juga *level* tegangan dibawah -25 Volt dan diatas +25 Volt juga harus dihindari karena bisa merusak line driver pada saluran RS232.



Gambar 2.16. Level Tegangan RS232 pada pengiriman huruf "A" Tanpa Bit Paritas.

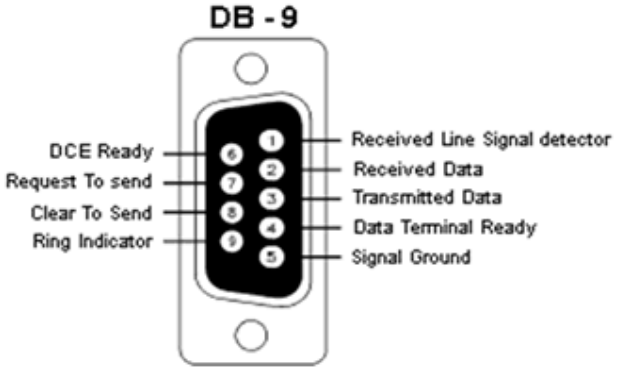
Flow Control

Jika kecepatan transfer data dari DTE ke DCE (misal dari komputer / modem) lebih cepat dari pada transfer data dari DCE ke DCE (modem ke modem) maka cepat atau lambat kehilangan data akan terjadi karena *buffer* pada DCE akan mengalami overflow. Untuk itu diperlukan sistem flow control untuk mengatasi masalah tersebut.

Terdapat 2 macam *flow control* yaitu secara *hardware* dan secara *software*.*Flow control* secara *software* atau yang sering disebut dengan *Xon* (karakter ASCII 17) dan *Xoff* (karakter ASCII 19). DCE akan mengirimkan Xoff ke komputer untuk memberitahukan agar komputer menghentikan pengiriman data jika buffer pada DCE telah penuh. Jika buffer telah kembali siap menerima data DCE akan mengirimkan karakter *Xon* ke komputer dan komputer akan melanjutkan pengiriman data sampai data terkirim semua. Keuntungan *flow control* ini adalah hanya diperlukan kabel sedikit, karena karakter kontrol dikirim lewat saluran TxRx.*Flow Control* secara *hardware* atau sering disebut RTS/CTS menggunakan dua kabel untuk melakukan pengontrolan. Komputer akan men-set saluran *Request to Send* (RTS) jika akan mengirimkan data ke DCE. Jika *buffer* di DCE siap menerima data, maka DCE akan membalas dengan men-set saluran *Clear to Send* (CTS) dan komputer akan mulai mengirimkan data.Jika buffer telah penuh, maka saluran akan di reset dan komputer akan menghentikan pengiriman data sampai saluran ini di-set kembali.

Konfigurasi Port Serial

Konektor DB-9 pada bagian belakang komputer adalah port serial RS232 yang biasa dinamai dengan COM1 dan COM2.



Gambar 2.17. Port Serial

Tabel 2.4. Konfigurasi port serial

Pin	Nama Sinyal	*Direction*	Keterangan
1	DCD	*In*	*Data Carrier Detect/Receive Line Signal Detect*
2	RxD	*In*	*Receive Data*
3	TxD	*Out*	*Transmit Data*
4	DTR	*Out*	*Data Terminal Ready*
5	GND	-	*Ground*
6	DSR	*In*	*Data Set Ready*
7	RTS	*Out*	*Request to Send*
8	CTS	*In*	*Clear to Send*
9	RI	*In*	*Ring Indicator*

Berikut ini keterangan mengenai fungsi saluran RS232 pada konektor DB-9:

1. *Received Line Signal Detect*, dengan saluran ini DCE memberitahukan ke DTE bahwa pada terminal masukan ada data masuk.
2. *Receive Data*, digunakan DTE untuk menerima data dari DCE.
3. *Transmit Data*, digunakan DTE untuk mengirimkan data ke DCE.
4. *Data Terminal Ready*, pada saluran ini DTE memberitahukan kesiapan terminalnya.
5. *Signal Ground*, saluran ground
6. *DCE ready* adalah sinyal aktif pada saluran ini menunjukkan bahwa DCE sudah siap.
7. *Request to Send*, dengan saluran ini DCE diminta mengirim data oleh DTE.
8. *Clear to Send*, dengan saluran ini DCE memberitahukan bahwa DTE boleh mulai mengirim data.
9. *Ring Indicator*, pada saluran ini DCE memberitahukan ke DTE bahwa sebuah stasiun menghendaki hubungan dengannya.

Dibawah ini adalah program inisialisasi untuk mikrokontroler:

```
Init_serial:  MOV SCON,#50H
              MOV TMOD,#20H
              MOV TH1,#0A0H
              MOV TCON,#40H
              MOV PCON,#00H
```

Penjelasan dari contoh program Init_serial

1. SCON, #50H

SCON (Serial Control) adalah register yang digunakan untuk mengatur komunikasi serial.

MSB							*LSB*
SM0	**SM1**	**SM2**	**REN**	**TB8**	**RB8**	**TI**	**RI**

Gambar 2.18. Alokasi Bit SCON

Tabel 2.5. *Serial Port Control*

Bit	**Alamat *Bit***	**Simbol**	**Deskripsi**
SCON.7	9FH	SM0	Pemilih Mode Komunikasi Serial
SCON.6	9EH	SM1	Pemilih Mode Komunikasi Serial
SCON.5	9DH	SM2	Pemilih Mode Komunikasi Multiprosesor
SCON.4	9CH	REN	*Reception Enable*
SCON.3	9BH	TB8	Bit ke-9 yang dikirim
SCON.2	9AH	RB8	Bit ke-9 yang diterima
SCON.1	99H	TI	Transmit Interupt Flag
SCON.0	98H	RI	Receive Interupt Flag

MOV SCON,#50H → 50H = 01010000b

Berikut ini adalah penjelasan masing-masing bit SCON yang berkaitan dengan serial port:

a. SM0 & SM1

Pemilihan mode komunikasi serial. Baud rate pada mode 1, 2 dan 3 dapat dilipatgandakan dengan memberi nilai ‘1’ pada SMOD. Baud

rate variabel adalah baud rate yang dapat diatur.

Tabel 2.6. Mode Komunikasi Serial

SM0	**SM1**	**Mode**	**Deskripsi**	***Baud Rate***
0	0	0	8-bit *Shift Register*	Frek. Osilator/12
0	1	1	8-bit UART	Variabel
1	0	2	9-bit UART	Frek. Osilator/64
1	1	3	9-bit UART	Variabel

b. SM2

Jika SM2 bernilai ‘1’ maka komunikasi multiprosesor diaktifkan dengan kondisi terdapat pada tabel 2.6.

Tabel 2.7. Mode Komunikasi Multiprosesor

Mode	Jika SM2 =1
2 atau 3	RI tidak akan diaktifkan jika bit ke-9 yang diterima bernilai ‘0’.
1	RI tidak akan diaktifkan jika stop bit yang valid (bernilai ‘1’) tidak diterima.

Pada mode 0, nilai SM2 harus ‘0’.

c. REN

REN harus diberi nilai ‘1’ untuk mengaktifkan penerimaan data. Jika REN diberi nilai ‘0’, maka tidak akan ada penerimaan data.

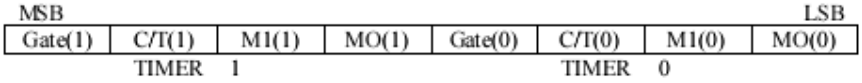
d. TB8

TB8 adalah bit ke-9 yang dikirimkan dalam mode 2 atau 3. Nilai bit ini diatur oleh program user

e. RB8

RB8 adalah bit ke-9 yang diterima dalam mode 2 atau 3. pada mode 1, RB8 adalah stop bit yang diterima. Pada mode 0, RB8 tidak digunakan.

2. TMOD, #20H → 20H = 00100000b



Gambar 2.19. Bit Register TMOD

SFR TMOD digunakan untuk mengontrol mode operasi dari kedua timer. Setiap bit dari SFR ini menyediakan informasi bagi mikrokontroler bagaimana menjalankan timer. Empat bit MSB (bit 4 hingga bit 7) berhubungan dengan TIMER1, sedangkan empat bit LSB (bit 0 hingga bit 3) mempunyai fungsi sama yang diperuntukan bagi TIMER0.

Tabel 2.8. Bit Register TMOD

Bit	Nama	Fungsi	*Timer*
7	Gate1	Jika bit ini diset, *timer* hanya akan bekerja jika INT1 (P3.3) berlogika 1. Jika bit ini dinolkan, *timer* akan bekerja tanpa dipengaruhi kondisi INT1	1

Tabel 2.8. Lanjutan			
6	C/T1	Jika bit ini diset, *timer* akan menghitung kondisi pada T1 (P3.5). Jika bit ini dinolkan, *timer* akan bertambah satu setiap siklus mesin	1
5	T1M1	Bit mode *timer*	1
4	T1M0	Bit mode *timer*	1
3	Gate0	Jika bit ini diset, *timer* hanya akan bekerja jika INT0 (P3.2) berlogika 1. Jika bit ini dinolkan, *timer* akan bekerja tanpa dipengaruhi kondisi INT1	0
2	C/T0	Jika bit ini diset, *timer* akan menghitung kondisi pada T0 (P3.4). Jika bit ini dinolkan, *timer* akan bertambah satu setiap siklus mesin	0
1	T0M1	Bit mode *timer*	0
0	T0M0	Bit mode *timer*	0

Seperti terlihat pada tabel diatas, ada 4 bit yang menyatakan mode untuk kedua timer. Masing-masing dua bit untuk satu timer. Adapun mode operasi yang dimaksud disini tercantum dalam tabel di bawah ini.

Tabel 2.9 Mode Operasi Timer

TxM1	Tx0M0	Mode Timer	Keterangan
0	0	0	*Timer* 13 bit
0	1	1	*Timer* 16 bit
1	0	2	8 bit *auto reload*
1	1	3	Mode timer *split*

Mode Timer 13 Bit (Mode 0)

Dalam mode 0, timer yang dibentuk adalah timer 13 bit. Mode ini digunakan untuk menjaga kompatibilitas pendahulu keluarga 8051, yaitu generasi 8048. Pada saat ini timer 13 bit sudah jarang digunakan. Saat timer diset sebagai timer 13 bit, TLx akan mencacah dari 0 hingga 31. jika TLx melebihi 31, maka akan direset ke harga awal 0 dan kemudian menambah harga THx. Dengan demikian, hanya 13 bit dari dua byte yang digunakan, yaitu bit 0-4 dari TLx dan bit 0-7 dari THx. Sehingga maksimum harga yang bisa dicapai adalah 8.192. sehingga jika timer diset dalam mode ini, maka akan menjadi nol setelah 8.192 siklus.

Mode Timer 16 Bit (Mode 1)

Timer mode 1 adalah timer 16 bit. Mode ini adalah mode yang paling umum digunakan. Fungsinya sama dengan timer 13 bit, namun yang didayagunakan adalah 16 bit. TLx akan mencacah dari 0 hingga 255. Jika TLx melebihi 255, maka akan kembali reset ke 0 dan menambah THx dengan 1. Karena kemampuan 16 bit, maka mode ini memiliki batas maksimum harga 65.535. Sehingga jika timer diset dalam mode ini, maka akan menjadi 0 setelah 65.535 siklus mesin.

Mode Timer 8 Bit Auto Reload (Mode 2)

Timer mode 2 adalah timer 8 bit dengan kemampuan pengisian ulang (auto reload). Dalam mode ini, THx akan menyimpan harga awal counter dan TLx berfungsi sebagai timer 8 bit. TLx akan memulai mencacah dengan harga yang tersimpan pada THx, dan jika telah melampaui harga 255, maka akan reset dan kembali ke harga awal yang tersimpan di THx. Sebagai contoh, dimisalkan TH0 menyimpan nilai FDh dan TL0 dengan nilai Feh, maka untuk beberapa siklus mesin didapatkan urutan perubahan harga tercantum dalam tabel dibawah ini.

Tabel 2.10. Contoh Perubahan Nilai TL0

Siklus Mesin	Harga TH0	Harga TL0
1	FDH	FEH
2	FDH	FFH
3	FDH	FDH
4	FDH	FEH
5	FDH	FFH
6	FDH	FDH
7	FDH	FEH

Seperti terlihat diatas, nilai TH0 tidak pernah berubah. Dengan demikian dalam mode 2, THx merupakan variabel yang menentukan waktu sedangkan TLx adalah timer yang selalu mencacah secara konstan setiap siklus mesin. TLx akan overflow dan reset ke nilai yang tersimpan dalam TH0. keuntungan yang didapatkan karena fleksibelitas penentuan tenggang waktu dengan mengatur nilai pada THx.

Contohnya jika diinginkan timer yang selalu menghitung dari 200 hingga 255. jika digunakan mode 0 atau 1, diperlukan pengecekan terus-menerus apakah timer mengalami overflow atau tidak. Jika terjadi overflow, maka diperlukan kode untuk mereset timer ke harga 200. Hal ini akan memerlukan banyak instruksi dan memakan waktu sehingga tidak efisien. Namun, jika digunakan mode 2, tidak diperlukan monitor terus-menerus terhadap kondisi timer. Cukup masukkan nilai 200 pada THx dan biarkan mikrokontroler yang mengatur agar timer selalu mencacah dari 200 hingga 255.

Mode Timer Split (Mode 3)

Timer mode 3 adalah mode timer split. Jika TIMER0 diset dalam mode 3, maka akan menjadi 2 timer 8 bit yang berbeda. Timer 0 adalah TL0 dan timer 1 adalah TH0. kedua-duanya akan mencacah dari 0 hingga 255 dan jika menemui kondisi overflow akan reset ke nol. Saat TIMER0 dalam mode split, TIMER1 bisa diset pada mode 0, 1 atau bahkan 2 secara normal. Mode ini hanya dipakai jika diperlukan dua timer 8 bit yang terpisah.

3. Nilai *baud rate* yang digunakan. Cara penghitungan TH1 ditunjukkan pada persamaan dibawah ini:

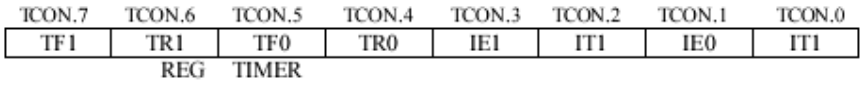
$$\text{TH1} = 256 - \frac{\text{k} * \text{Fosc}}{32 * 12 * \text{Baud Rate}} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.5.1)$$

Hal-hal yang harus diperhatikan dalam menentukan baud rate:

- Nilai k tergantung dari SM0 yang terdapat pada register SCON, yaitu:
    1. Jika nilai SM0 pada register SCON bernilai nol, maka nilai k yang diberikan adalah 1.
    2. Jika nilai SM0 pada register SCON bernilai 1, maka nilai k yang diberikan adalah dua.

Persamaan diatas digunakan untuk menghitung TH1 pada dengan baud rate yang diinginkan untuk mode komunikasi serial 1.

4. TCON, #40H → 40H= 01000000



Gambar 2.20. Bit register TCON

Register ini mengontrol kedua timer dan menyediakan informasi yang sangat berguna berkaitan dengan timer-timer tersebut. Struktur register TCON dapat dilihat pada tabel di bawah ini.

Tabel 2.11. Bit register TCON

Bit	Nama	Alamat	Fungsi	Timer
7	TF1	8FH	Timer 1 *Overflow*. Bit ini diset oleh mikrokontroler jika Timer 1 *overflow.*	1
6	TR1	8EH	Timer 1 Run. Jika Bit ini diset, maka Timer 1 akan bekerja. Sebaliknya jika direset, maka Timer 1 akan mati.	1
5	TF0	8DH	Timer 0 *Overflow*. Bit ini diset oleh mikrokontroler jika Timer 0 *overflow.*	0
4	TR0	8CH	Timer 0 Run. Jika Bit ini diset, maka Timer 0 akan bekerja. Sebaliknya jika direset, maka Timer 0 akan mati.	0

Dalam tabel hanya dicantumkan 4 bit dari 8 bit yang ada pada register TCON. Hal ini karena hanya 4 bit (bit 4 hingga bit 7) yang berkaitan dengan timer, sedangkan bit sisanya berkaitan dengan interupsi.

5. PCON ,#00h

Register power control digunakan untuk pemakaian daya pada IC.

2.6. Bahasa *Assembly*

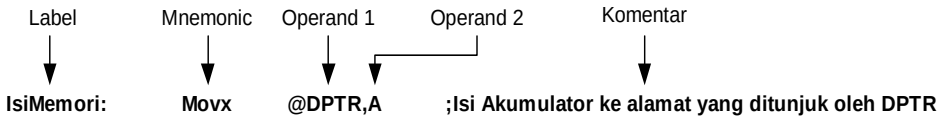
Assembler adalah program komputer yang men-*translitrasi* program dari bahasa *assembly* ke bahasa mesin. Sedangkan bahasa *assembly* adalah ekuivalensi bahasa mesin dalam bentuk alpanumerik. *Mnemonics* alpanumerik digunakan sebagai alat

bantu bagi programer untuk memprogram mesin komputer daripada menggunakan serangkaian 0 dan 1 (bahasa mesin) yang panjang dan rumit.

2.6.1. Konstruksi Program *Assembly*

Program sumber *assembly* terdiri dari kumpulan baris-baris perintah dan biasanya disimpan dengan *extension* .ASM dengan 1 baris untuk satu perintah, setiap baris perintah tersebut bisa terdiri atas beberapa bagian, yakni bagian label, bagian *mnemonic*, bagian *operand* yang bisa lebih dari satu dan terakhir bagian komentar seperti yang terlihat pada gambar 2.21. Program sumber (*source code*) dibuat dengan program editor seperti *notepad* atau Editor DOS, selanjutnya program sumber diterjemahkan ke bahasa mesin dengan menggunakan program *assembler*. Hasil kerja program *assembler* adalah "*program objek*" dan juga "*assembly listing*". Ketentuan penulisan *source code* adalah sebagai berikut:

1. Masing-masing bagian dipisahkan dengan spasi atau TAB, khusus untuk *operand* yang lebih dari satu masing-masing *operand* dipisahkan dengan koma.
2. Bagian-bagian tersebut tidak harus semuanya ada dalam sebuah baris, jika ada satu bagian yang tidak ada maka spasi atau TAB sebagai pemisah bagian tetap harus ditulis.
3. Bagian label ditulis mulai huruf pertama dari baris, jika baris bersangkutan tidak mengandung label maka label tersebut digantikan dengan spasi atau TAB, yakni sebagai tanda pemisah antara bagian label dan bagian *mnemonic*.



Gambar 2.21. Bentuk Program Sumber *Assembly*

2.6.2. Instruksi MCS-51 yang Digunakan

Beberapa instruksi yang digunakan dalam penyusunan program alat bantu baca tunanetra, adalah:

1. EQU

   Konstanta Data yang dideklarasikan dengan nilai tertentu.

2. ORG (*Origin)*

   Digunakan untuk menunjukkan lokasi memori tempat instruksi atau perintah yang ada dibawahnya disimpan.

3. END

   END biasanya diletakkan di akhir baris dari *file* program sumber *assembler* sebagai tanda akhir pernyataan (*statement*) bagi program *assembler* dalam melakukan proses *assembly*.

4. MOV

   Instruksi ini melakukan pemindahan data dari variabel pada kode operasi kedua dan disimpan di variabel pada kode operasi pertama.

5. INC

   Menambahkan nilai variabel yang ditunjuk dengan 1 dan hasilnya disimpan di variabel tersebut.

6. ACALL

   Instruksi melakukan lompatan dengan area sebesar 2 Kbyte.

7. JMP

   Melakukan lompatan dan menjalankan program yang berada di alamat yang ditentukan oleh label tertentu.

8. DJNZ

   Melakukan pengurangan pada Rn (R0…R7) dengan 1 dan lompat ke alamat yang ditentukan apabila hasilnya bukan 00. Apabila hasilnya telah mencapai 00, maka program akan terus menjalankan instruksi dibawahnya.

9. CJNE

   Instruksi ini melakukan perbandingan antara data tujuan dan data sumber serta melakukan lompatan ke alamat yang ditentukan jika hasil perbadingan tidak sama.

10. PUSH

    Melakukan penyimpanan data dari suatu *register* atau memori ke dalam *stack*. Lokasi data dalam *stack* tersebut ditunjuk oleh *stack pointer*. Pada saat instruksi ini dijalankan, nilai dari *stack pointer* akan bertambah satu

dan *register* atau memori yang di PUSH akan masuk ke dalam alamat yang ditunjuk oleh *stack pointer* tersebut.

11. POP

    Melakukan pengambilan data dari dalam *stack* ke suatu *register* atau memori. Lokasi data dalam *stack* tersebut ditunjuk oleh *stack pointer*. Pada saat instruksi ini dijalankan, data di alamat yang ditunjuk oleh *stack pointer* akan dipindah ke *register* atau memori dilanjutkan dengan pengurangan nilai *stack pointer* dengan nilai 1.

12. RET

    Instruksi ini melakukan lompatan ke alamat yang disimpan dalam SP dan SP-1. Instruksi ini biasa digunakan pada saat kembali dari subrutin yang dipanggil dengan instruksi ACALL atau LCALL.

2.7. Pengenalan Suara *(Speech Recognition)*

Speech Recognition adalah sebuah proses pengenalan suara manusia dan mengubahnya menjadi sesuatu yang dapat dimengerti oleh komputer. Oleh karena itu, komputer membutuhkan penerjemaah. Penelitian untuk membuat algoritma pengenalan suara dan membuat model-model yang efektif sudah dilakukan sejak komputer ditemukan[2].

2.7.1. Kunci Operasi Sistem Pengenalan Suara:

Setiap sistem pengenalan suara menggunakan empat kunci operasi untuk mendengarkan dan mengerti tentang apa yang diucapkan, yaitu:

1. Pemisahan kata ialah proses untuk memisahkan suara yang diucapkan menjadi beberapa bagian. Masing-masing bagian bisa berupa kalimat ataupun hanya sebuah kata. Terdapat tiga dasar metode yang dapat digunakan pada proses pemisahan kata ini, yaitu *discrete speech*, *word spotting* dan *continuous speech*. Pada *discrete speech*, mengucapkan kalimat secara terpenggal dan ada jeda sejenak diantara kata yang diucapkan. Jeda tersebut digunakan oleh sistem untuk mendeteksi awal dan akhir sebuah kata. *Discrete speech* ini mempunyai kelebihan yaitu sedikit *resource* (memori komputer, waktu proses) yang digunakan oleh sistem untuk mendeteksi suara, tetapi mempunyai kelemahan yaitu ketidaknyamanan dalam pengucapan sebuah kalimat. Pada *word spotting*,

dalam sebuah kalimat yang diucapkan, sistem hanya mendeteksi kata yang terdapat di dalam perbendaharaan yang dimilikinya, dan mengabaikan kata-kata lain yang tidak dimilikinya. Sehingga walau kalimat yang diucapkan berbeda, tetapi di dalam kalimat tersebut terdapat sebuah kata yang sama dan terdapat di perbendaharaan kata, maka hasil pengenalan akan sama. Kelemahan metode ini ialah besar kemungkinan sistem akan melakukan kesalahan arti pengenalan dalam bentuk kalimat. Tetapi metode ini mempunyai kelebihan yaitu pengguna dapat mengucapkan kalimat secara normal tanpa harus berhenti diantara kata. Pada metode *continuous speech*, sistem akan mengenali dan memproses setiap kata yang diucapkan. Metode ini akan menghasilkan keakuratan dalam mengenali ucapan. Tetapi di samping itu metode ini memerlukan *resource* yang besar dalam prosesnya. Pada metode ini, sistem harus dapat mendeteksi awal dan akhir setiap kata dalam kalimat tanpa adanya jeda diantara kata-kata tersebut, dan setelah berhasil memisahkan kata, langkah selanjutnya adalah mencocokkan dengan perbendaharaan kata yang dimilikinya.

2. Perbendaharaan kata memiliki dua hal yang sangat penting untuk diperhatikan, yaitu ukuran dan keakuratan. Jika perbendaharaan kata berjumlah banyak maka sebuah sistem akan mudah dalam melakukan pencocokan kata, tetapi dengan makin meningkatnya jumlah perbendaharaan kata, maka jumlah kata yang mempunyai ucapan hampir sama juga meningkat yang pada akhirnya akan menurunkan keakuratan pengenalan. Sebaliknya, jika sebuah sistem mempunyai perbendaharaan kata sedikit, maka keakuratan pengenalan akan meningkat karena sedikitnya kata yang hampir sama, tetapi kata yang tidak dikenali akan semakin banyak. Untuk sistem pengenalan pembicaraan dengan mode *command and control*, akan lebih baik jika menggunakan jumlah perbendaharaan kata sedikit (kurang dari 100 kata), tetapi untuk mode diktasi akan membutuhkan jumlah perbendaharaan kata yang banyak.

3. Pencocokan kata adalah proses untuk mencocokkan kata ucapan yang berhasil diidentifikasi dengan basis data yang dimiliki oleh sistem. Terdapat dua metode yang dapat dipakai pada proses pencocokan kata

ini, yaitu : *whole-word matching* dan *phoneme matching*. Pada *whole-word matching*, sistem akan mencari kata yang sama persis di basis data dengan kata yang diucapkan. Sedangkan pada *phoneme matching*, sistem mempunyai kamus fonem. Fonem ialah bagian terkecil dan unik dari suara yang membentuk sebuah kata.

4. Sistem pengenalan pembicaraan mempunyai beberapa sifat dilihat dari ketergantungan terhadap pengguna yaitu *speaker dependent, speaker independent* dan *speaker adaptive*. Pada *speaker dependent*, sistem membutuhkan pelatihan untuk setiap pengguna yang akan menggunakan sistem tersebut. Sistem tidak akan bisa mengenali pengguna yang belum pernah melakukan pelatihan. Pada *speaker independent*, pengguna tidak perlu melakukan pelatihan sebelum dapat menggunakan sistem, karena sistem mampu mengenali suara semua pengguna tidak tergantung warna suara dan dialek yang digunakan. *Speaker adaptive* merupakan perpaduan dari *speaker dependent* dan *speaker independent*, dimana pengguna tidak perlu melakukan pelatihan dan keakuratan pengenalan sistem akan makin meningkat jika pengguna yang sama bekerja terus menerus selama beberapa waktu tertentu.

2.7.2. *Grammar*

Elemen terakhir dari *speech engine* adalah *grammar rules.* G*rammar rules* biasanya digunakan perangkat lunak pengenalan suara untuk menganalisa input suara, memprosesnya, mengenali dan mengerti tentang apa yang telah diucapkan. *Grammars* dapat dibagi kedalam tiga tipe, setiap tipe memiliki kelemahan dan kelebihannya masing-masing. Berikut ini tipe-tipe dari *grammars*:

1. *Context-free grammars*
2. *Dictation grammars*
3. *Limited domain grammars*

2.7.2.1.*Context-Free Grammars*

Context-free grammars menawarkan fleksibilitas yang besar ketika mengiterpretasikan suara manusia. *Context-free grammars*

bekerja dengan prinsip yang mengikuti aturan yang yang telah dibentuk untuk menentukan kata apa yang akan muncul selanjutnya pada sebuah kalimat. *Context-free grammars* tidak bekerja dengan cara bahwa setiap kata harus dimengerti dalam sebuah *context*, melainkan *Context-free grammars* harus mengevaluasi hubungan antara setiap kata dan bagian kata untuk mengetahui aturan tentang kata yang mungkin akan diberikan saat itu. Elemen utama dari *context-free grammar* adalah:

1. Kata-kata

   Sebuah daftar dari kata yang sah untuk diucapkan.
2. Aturan-aturan

   Sebuah aturan dari struktur suara pada kata yang digunakan.
3. Daftar-daftar.

   Satu atau lebih aturan kata yang telah ditetapkan.

Context-free grammars merupakan sistem yang sangat baik untuk menghadapi variasi *input* yang sangat besar. *Context-free systems* juga dapat digunakan untuk menangani variabel-variabel kosakata, karena kebanyakan aturan yang dibangun untuk *Context-free grammars* berada disekitar daftar-daftar yang dideklarasikan dan kumpulan kata yang cocok. Saat *engine* pengenalan suara dapat mengerti aturan-aturan, maka akan sangat mudah untuk memperluas kosakata dengan cara mengembangkan daftar-daftarnya.

Hal yang paling penting pada *context-free* grammars adalah mengijinkan untuk memperluas kosakata pada saat sistem sedang berjalan. Saat operasi *context-free grammars* fokus pada *lists* (daftar-daftar), pada saat itu *context-free grammars* mengijinkan pengguna untuk menambahkan anggota-anggota *lists* untuk meningkatkan nilai dari sistem pengenalan suara dengan cepat. Ini memudahkan pengguna untuk meng-*install* sebuah sistem hanya dengan komponen-komponen dasar. Sistem dasar dapat dikembangkan untuk menangani kebutuhan dari para pengguna.

Dengan langkah ini *context-free grammars* menawarkan fleksibilitas yang sangat tinggi dengan biaya pengembangan dan kesulitan yang sedikit.

2.7.2.2. *Dictation Grammars*

Tidak seperti *context-free grammars* yang beroperasi menggunakan aturan-aturan, *dictation grammars* memfokuskan evaluasinya menggunakan kosakatanya sendiri. Fungsi utama dari sebuah *dictation grammars* adalah untuk mengubah suara manusia kedalam bentuk tulisan seakurat mungkin dengan sangat teliti. Untuk melakukan hal ini *dictation grammars* tidak hanya membutuhkan kekayaan kosakata, tetapi sebuah contoh output untuk dipergunakan sebagai sebuah model ketika menganalisa input suara. Terdapat empat elemen *dictation grammars*, yatiu:

1. *Topic*

   Mengidentifikasi topik diktasi, seperti tentang medis atau Undang-undang.

2. *Common*

   Kumpulan kata-kata yang biasanya digunakan pada diktasi. Biasanya pada sebuah kelompok berisi tentang teknis atau mengkhususkan kata-kata yang mungkin muncul pada saat diktasi..

3. *Group*

   Kumpulan kata yang berhubungan tapi tidak memiliki hubungan langsung dengan topik diktasi. *Group* biasanya memiliki kumpulan kata yang memiliki harapan keluar selama diktasi. Model *grammar* dapat memiliki lebih dari satu grup.

4. *Sample*

   Sebuah contoh *text* yang menampilkan gaya penulisan dari pembicara atau format umum diktasi. *Text* ini digunakan untuk membantu e*ngine* pengenalan suara untuk menganalisa input suara.

Keberhasilan dari sebuah *dictation grammar* tergantung dari kualitas kosakata. Lebih banyak kosakata pada *lists*, semakin besar juga peluang untuk melakukan kesalahan, seperti salah menempatkan antara satu *item* dengan *item* yang lainnya. Tetapi semakin kosakata dibatasi maka semakin banyak kata-kata yang tidak dikenali yang terjadi pada saat diktasi. Kebanyakan sistem diktasi yang berhasil adalah dengan cara menyeimbangkan banyak kosakata dan kata-kata unik yang terdapat didalam basis data. Untuk itu, sistem diktasi biasanya hanya diperuntukkan pada satu topik saja, seperti tentang Undang-undang atau diktasi medis. Dengan membatasi kosakata yang memiliki kemiripan selama pelatihan diktasi, maka akan meningkatkan akurasi dari diktasi.

2.7.2.3. *Limited domain grammars*

Limited domain grammars menawarkan sebuah kompromi diantara fleksibilitas *context-free grammars* dan akurasi dari *dictation grammars*. *Limited domain grammars* mempunyai elemen-elemen sebagai berikut:

1. Kata-kata

   Sebuah *list* yang mengkhususkan kata-kata yang memiliki kemiripan yang terjadi saat sebuah sesi berjalan.

2. *Group*

   Sekumpulan kata yang berhubungan. *Grammar* dapat berisi beberapa kumpulan kata-kata. Satu ungkapan tunggal diharapkan meliputi salah satu kata di dalam group.

3. *Sample*

   Sebuah contoh *text* yang menampilkan gaya penulisan dari pembicara atau format umum diktasi. *Text* ini digunakan untuk membantu e*ngine* pengenalan suara untuk menganalisa *input* suara.

Limited domain grammars berguna pada situasi dimana sistem tidak membutuhkan kosakata yang besar. Contoh-contoh yang terdapat pada sistem menggunakan bahasa alami untuk menerima perintah.

2.7.3. Pengenalan Suara *(Speech Recognition) dengan SAPI (Speech Application Programming Interface)*

SAPI 5.1 automation object dapat digunakan baik untuk diktasi maupun CnC *(Command and Control) speech recognition*. *SAPI 5.1* memiliki 19 komponen yang di dalamnya terdapat empat bagian pusat untuk pengenalan suara, yaitu:

1. *SpInprocRecognizer*

   Merepresentasikan sebuah engine pengenalan suara yang digunakan hanya pada satu aplikasi saja.

2. *SpSharedRecognizer*

   Merepresentasikan sebuah engine pengenalan suara yang dapat berbagi banyak aplikasi.

3. *SpInprocRecoContext*

   Adalah sebuah konteks pengenalan yang digunakan oleh SpInprocRecognizer.

4. *SpSharedRecoContext*

   Adalah sebuah konteks pengenalan yang digunakan oleh SpSharedRecognizer.

2.7.4. Arsitektur *Speech Application Programming Interface*

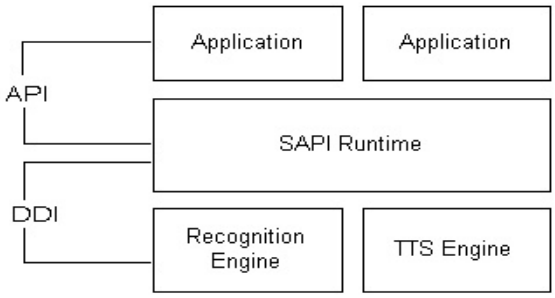
SAPI 5.1 terdiri dari 2 antar muka yaitu *application programming interface* (API) dan d*evice driver interface* (DDI). SAPI menyediakan sebuah *high-level interface* antara aplikasi dengan speech engine. SAPI juga mengimplementasikan semua detail *level* bawah yang dibutuhkan untuk mengontrol dan mengatur operasi *real-time* dari berbagai speech engines. Dua tipe dasar dari SAPI *engine* adalah *text-to-speech (TTS)* dan *speech recognition.* TTS mengubah bentuk tulisan (string) dan *files* kedalam bentuk audio suara menggunakan *synthetic voice*, sedangkan pengenalan suara mengubah suara yang diucapkan oleh manusia kedalam bentuk teks string dan *files*. Model SAPI dibagi ke dalam dua tingkatan:

1. High-level SAPI

   Tingkatan ini menyediakan dasar speech service dalam *form command* dan control (CnC) dan text-to-speech sederhana.

2. Low-level SAPI

Tingkatan ini menyediakan semua akses yang terperinci untuk semua speech service, direct interface untuk mengontrol dialog, manipulasi atribut untuk pengenalan suara dan text-to-speech.



Gambar 2.22. Arsitektur SAPI

2.7.4.1. *Application programming interface* (API)

Pada aplikasi pengenalan suara, ISpRecoContext adalah interface utama dan IspEventSource menjadi kendaraan untuk menerima notifikasi permintaan *speech recognition event*. Ketika aplikasi menerima notifikasi, ada kemungkinan terdapat informasi yang sama pada beberapa even. Dengan memanggil ISpEventSource::GetInfo, maka variable anggota ulCount akan mengembalikan nilai yang berupa struktur SPEVENT SOURCEINFO yang didalamnya terdapat jumlah even yang memiliki informasi yang sama. Dengan menggunakan IspEvent Source::GetEvents, aplikasi akan mengeluarkan sejumlah struktur SPEVENT, di mana masing-masing mempunyai informasi tentang even tertentu.

2.7.4.2. *Device Driver Interface* (*DDI*)

DDI menyediakan fungsi untuk menerima data suara dari SAPI dan mengembalikan pengenalan frasa ke level SAPI paling dasar. Terdapat dua antar muka yang digunakan oleh DDI yaitu ISpSREngine, yang diimplementasikan oleh *engine* dan ISpSREngineSite yang diimplementasikan oleh SAPI. Antar muka SAPI yaitu ISpSREngineSite juga menyediakan metode untuk memberikan informasi lebih detail mengenai apa yang

dikenali oleh *engine*. *Grammars* dan *speakers* menyediakan informasi ke *engine* yang dapat membantu *engine* untuk melakukan pengenalan pembicaraan lebih baik, disamping juga merupakan bagian penting komunikasi yang menghubungkan SAPI dan *speech engine*. Terdapat 2 aspek terakhir yang berhubungan dengan komunikasi antara *engine* dan SAPI yaitu urutan pemanggilan yang terjadi serta masalah *threading.* Salah satu keuntungan menggunakan SAPI 5.1 ini adalah penyederhanaan masalah *threading*. *Engine* menyediakan layanan ke SAPI melalui antar muka ISpSREngine. Semua fungsi pengenalan terjadi melalui IspSREngine::RecognizeStream. Ketika SAPI memanggil ISpSREngine::SetSite, maka SAPI memberikan pointer ke antar muka ISpSREngineSite, kemudian *engine* dapat berkomunikasi dengan SAPI selama ISpSREngine::RecognizeStream dieksekusi. SAPI membuat sebuah *thread* ke obyek ISpSREngine dan *engine* tidak boleh meninggalkan ISpSREngine::Recognize Stream sampai terjadi kesalahan atau SAPI sudah terindikasi dengan menggunakan ISpSREngineSite::Read, yang artinya tidak ada lagi data yang dapat diproses dan *engine* telah selesai melakukan tugasnya. SAPI memisahkan *engine* dari kerumitan untuk mengatur peralatan suara secara detail. SAPI menjaga *logical stream* dari *raw audio data* dengan membuat indeks posisi *stream*. Dengan menggunakan indeks posisi *stream*, *engine* dapat melakukan pemanggilan terhadap ISpSREngineSite::Read untuk menerima *buffer* dari *raw audio data* selama ISpSREngine::Recognize Stream dieksekusi. Pemanggilan ini akan terjadi sampai semua data yang dibutuhkan tersedia.

DDI memungkinkan *engine* hanya mempunyai satu buah thread yang dieksekusi antara SAPI dan *engine*. Satu-satunya metode yang tidak mengijinkan IspSREngine untuk masuk dan keluar secara cepat ialah ISpSREngine::Recognize Stream. *Engine* akan selalu tereksekusi dan terpisah di dalam

::RecognizeStream sampai terjadi kegagalan atau tidak ada lagi data yang diterima dari ISpSREngine::Read. Ketika *engine* mempunyai kesempatan untuk memberikan SAPI giliran melakukan pemanggilan kembali ke ISpSREngine, maka *engine* harus memanggil IspSREngine::Synchronize dan memberikan indeks posisi *stream*, pada saat *engine* telah selesai melakukan pengenalan data. Pada saat sinkroninisasi, SAPI dapat melakukan pemanggilan kembali ke metode ISpSREngine yang lain kecuali Recognize. Sebagai contoh, *speaker* dapat berubah, *grammars* dapat di-*unload*, diaktifkan ataupun dinon-aktifkan secara dinamis.

2.7.5. Perangkat yang Dibutuhkan Untuk Pengenalan Suara menggunakan SAPI 5.1

Sistem suara membutuhkan sumber daya yang intensif. *Engine* pengenalan suara membutuhkan *RAM (Random Access Memory)* yang cukup dan *disk space* untuk merespon permintaan pengguna dengan cepat. Maka dari itu, sangat penting untuk menetapkan perangkat keras dan perangkat lunak yang dibutuhkan ketika membuat model dan mengimplementasikannya. Tidak semua komputer memiliki *memory, disk space* dan perangkat keras yang dibutuhkan untuk mengimplementasikan pengenalan suara dan *text-to speech*. Dibawah ini ada tiga kategori umum dari sumber daya yang dibutuhkan:

1. Perangkat keras umum seperti kecepatan *processor* dan *RAM*.

   Sistem suara membutuhkan sumber daya processor dan RAM. *SR (speech recognition) services* membutuhkan variasi tingkatan sumber daya yang tergantung pada jenis *engine* pengenalan suara yang sudah terinstall dan tingkatan dari *services* yang diimplementasikan, sedangkan kebutuhan *text-to speech (TTS) engine* lebih stabil karena hanya tergantung pada *engine text-to speech engine* yang diinstal. SR dan TTS *engine* yang sekarang tersedia untuk SAPI yang biasanya dapat berhasil diimpelementasikan menggunakan 486/33 *processor chip* dan tambahan 1MB RAM. Tetapi konfigurasi seperti diatas tidak direkomendasikan. Disarankan *processor y*ang digunakan adalah Pentium *processor* (P60

atau lebih) dan sedikitnya memiliki 16MB dari total RAM. Sistem yang mendukung diktasi pengenalan suara membutuhkan sumber daya yang lebih besar. Pada umumnya yang menerapkan *commad and control (CnC) services* hanya akan membutuhkan 1MB dari RAM (tidak termasuk RAM yang diperlukan oleh aplikasi). *Dictation services* akan membutuhkan sedikitnya 8MB dari RAM. Tipe dari sampling suara, analisis dan ukuran dari pengenalan kosakata semuanya mempengaruhi sumber daya yang dibutuhkan. Tabel di bawah ini menunjukkan kebutuhan sumber daya yang dibutuhkan untuk *speech recognition services.*

Tabel 2.12. Minimal *processor* dan *RAM* yang dibutuhkan oleh SR *services.*

Level-level dari *speech recognition services*	***Processor* (Minimal)**	***Additional RAM* (Minimal)**
Discrete, speaker-depenent, whole word, small vocabulary	386/16	64K
Discrete, speaker-indepenent, whole word, small vocabulary	386/33	256K
Countinous, speaker-independent, sub-word, small vocabulary	486/33	1MB
Countinous, speaker-depenent, whole word, large vocabulary	Pentium	8MB
Countinous, speaker-indepenent, whole word, large vocabulary	RISC processor	8MB

Pada Windows 95, memori minimal yang dibutuhkan adalah sebesar 12MB, tetapi disarankan lebih besar dari itu yaitu 16MB dan 24MB. Pada Windows NT memori minimal yang dibutuhkan adalah sebesar 16MB atau 24MB, tetapi disarankan lebih besar dari itu yaitu 32MB.

2. Perangkat lunak seperti sistem operasi dan *engine* pengenalan suara dan *engine text-to speech.*

   Microsoft Speech API hanya dapat diterapkan pada sistem operasi *Windows 32 bit.* Oleh karena itu kita membutuhkan *Windows 95,*

Windows NT atau versi yang lebih besar lagi pada *workstation.* Perangkat lunak yang sangat penting untuk mengimplementasikan *speech services* adalah *SR engine* dan *TTS engine*. SR dan TTS *engine* adalah *back-end processing module* dalam SAPI model, sedangkan aplikasi kita berperan sebagai front end dan *SPEEch.dll* berperan sebagai perantara di antara dua proses. Selain perangkat lunak diatas, kita juga harus memiliki sebuah produk dari *Microsoft* yaitu *Microsoft Speech SDK 5.1*.

3. Perangkat keras khusus seperti *sound cards, microphone, speaker* dan *headphones.*

2.7.6. Batasan Pengenalan Suara

Terdapat tiga batasan yang sangat vital dari teknologi pengenalan suara, yaitu:

1. Identifikasi pembicara

   Kesulitan dalam membedakan pembicara dan juga pada sistem diktasi *engine* pengenalan suara tidak dapat membedakan jika ada beberapa pembicara yang berbicara secara bersamaan.

2. Pengenalan input

   Engine pengenalan suara tidak dapat mengenali bahasa alami karena *engine* pengenalan suara hanya dapat mengenali kata yang terdapat pada kosakata yang dimilikinya lalu memprosesnya berdasarkan aturan *grammar* yang *speech engine* ketahui. SR *engine* tiddak mampu berhadapan dengan variasi pengucapan kata-kata yang sangat besar, contohnya seperti pengucapan kata *"either" (ee-ther atau I-ter)* dan kata *"potato" (po-tay-toe atau po-tah-toe).* Hal seperti itu akan membuat sistem menjadi bingung. Oleh karena itu, banyaknya variasi pengucapan kata-kata sangat besar pengaruhnya dan dapat mengurangi akurasi sistem pengenalan suara.

3. Akurasi pengenalan

   Akurasi pengenalan dapat dipengaruhi oleh dialek, kualitas dari microphone dan level noise yang ada selama sesi pengucapan. Sama

seperti pada masalah pengenalan suara, variasi dialek dapat menghambat performa dari *engine* pengenalan suara.

2.7.7. Diktasi

Diktasi merupakan salah satu jenis pengenalan suara di mana suatu mesin mendengarkan apa yang kita ucapkan dan menterjemaahkannya ke dalam bentuk teks[2].

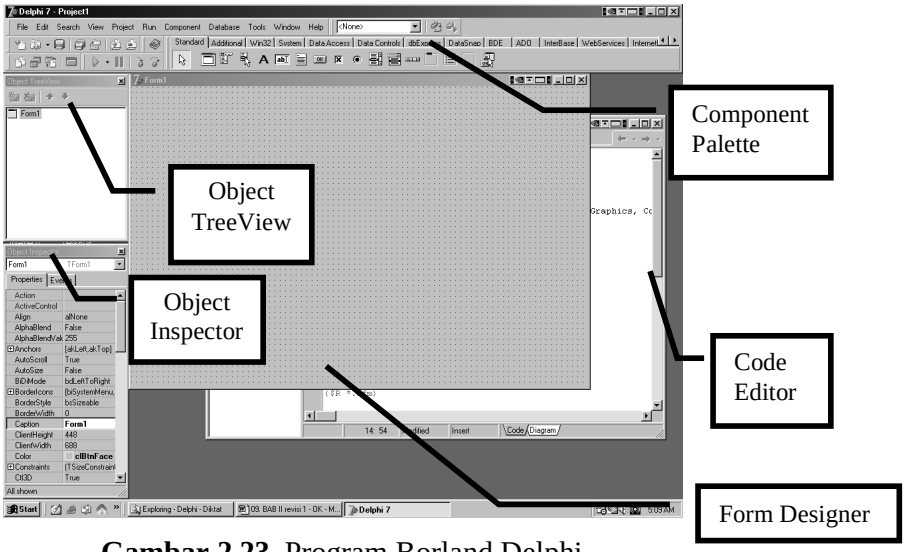
Semua itu terjadi di dalam *speech engine*. Kebanyakan dari *dictation engines* yang modern menggunakan suatu langkah, di mana *engine* tersebut mendengarkan kata yang diucapkan dan memecahnya menjadi suatu rangkaian hipotesis kata. Setiap hipotesis kata berisi sebuah daftar kata-kata yang mungkin dengan memberikan beberapa kemungkinan dan menjadi tepat. Sebagai contoh, pada kalimat *"The quick red fox"*, komputer akan memecahnya menjadi 4 hipoteis kata terpisah. *The "fox"*, hipotesisnya mungkin berisi beberapa kemungkinan seperti *"fax", "box", "fix"* dan lain-lain. Setiap hipotesis kata kemudian disimpan dalam bentuk "*context"*. Jadi, setiap kata akan dipertimbangkan hubungannya dengan kata-kata sebelum dan sesudahnya. Berdasarkan aturan-aturan pada konteks *speech engine*, maka hasil akhir yang didapat adalah yang terbaik dari kata yang telah diucapkan. Ketepatan dari diktasi tergantung dari kecepatan CPU dan sistem memori yang tersedia. Semakin banyak sumber daya semakin banyak juga konteks yang dapat dipertimbangkan pada suatu waktu dan dapat menghasilkan pengenalan yang akurat.

Yang terpenting untuk keakuratan dari pengenalan diktasi adalah *engine* dapat mengerti suara setiap pembicara. S*peech engines* dikhususkan untuk suatu bahasa tertentu bahkan mungkin hanya untuk suatu daerah. Inilah alasan mengapa ada *English engine, French engine, Chinese engine* dan lain-lain. Pada setiap bahasa tersebut terdapat perbedaan-perbedaan (kadang perbedaan yang ekstrim). Seorang anak perempuan berumur 5 tahun di komputer suaranya sangat berbeda dengan seorang laki-laki berumur 47 tahun. Inilah alasannya mengapa kebanyakan mesin diktasi memerlukan pelatihan suara karena pengenalan mode diktasi merupakan *speaker dependent*, yaitu keakuratan pengenalan mode ini bergantung pada

pola suara dan aksen pembicara serta pelatihan yang telah dilakukan. Semakin banyak sesi pelatihan yang diselesaikan maka keakuratan pengenalan diktasi yang didapat akan semakin baik.

2.8. Bahasa Pemrograman Delphi .

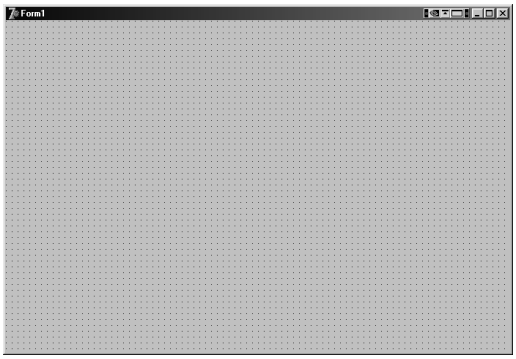
Borland Delphi atau yang biasa disebut *Delphi* saja, merupakan sarana pemrograman aplikasi *visual*. Bahasa pemrograman yang digunakan adalah bahasa pemrograman *pascal* atau yang kemudian juga disebut bahasa pemrograman *Delphi*. *Delphi* telah memanfaatkan suatu teknik pemrograman yang disebut RAD sehingga membuat pemrograman menjadi lebih mudah. *Delphi* adalah suatu bahasa pemrograman yang telah memanfaatkan metoda pemrograman *Object Oriented Programming* (OOP). Adapun tampilan program *delphi* dapat dilihat pada gambar 2.23.



Gambar 2.23. Program Borland Delphi

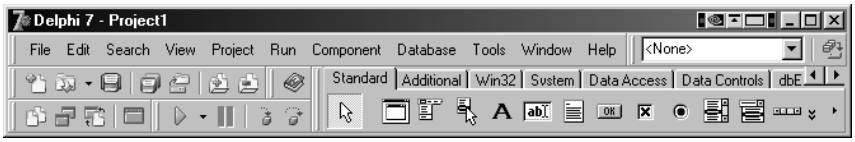
Tampilan sarana pengembangan aplikasi yang terdapat pada lingkungan kerja *Delphi* dapat dilihat pada gambar 2.23. Berikut penjelasan masing-masing bagian tersebut:

- *Form Designer* atau *form* adalah *windows* kosong tempat merancang antarmuka pemakai (*user interface*) aplikasi. Tampilan awalnya seperti pada gambar 2.24. pada form inilah ditempatkan komponen-konponen sehingga aplikasi dapat berinteraksi dengan pemakainya.



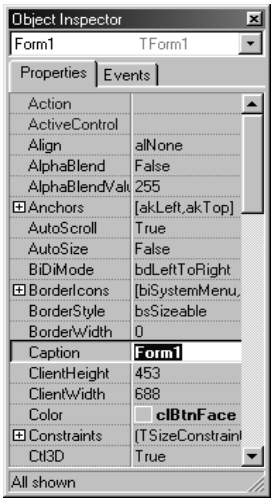
Gambar 2.24. *Form Designer* pada *Delphi*

- *Componen Palette*, berisi ikon-ikon komponen *visual* dan *nonvisual* yang dapat digunakan untuk merancang antarmuka bagi pemakai aplikasi. Komponen *palette* terdiri atas beberapa *page* yang dipakai sebagai pengelompok jenis komponen, misalnya yang tampak pada gambar 2.25 adalah *page standard*.



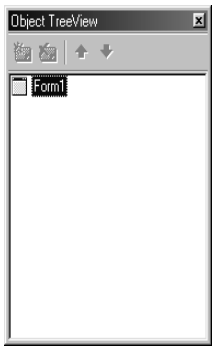
Gambar 2.25. *Component Palette*

- *Object Inspector*, untuk menentukan dan mengubah *property* (atribut) dan *event object*. Selain itu dapat juga dipilih komponen melalui *object inspector*. Tampilan *object inspector* adalah seperti yang terlihat pada gambar 2.26.



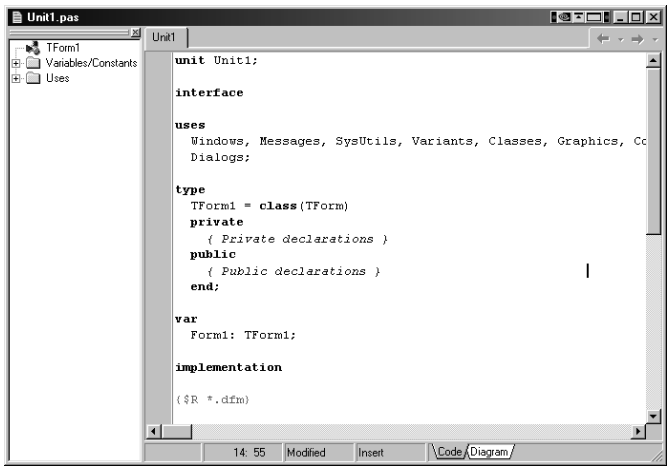
Gambar 2.26. *Windows Object Inspector*

- *Object TreeView* untuk menampilkan dan mengubah hubungan logis antar komponen di dalam projek. Contoh tampilan *object treeview* dapat dilihat pada gambar 2.27.



Gambar 2.27. *Windows Object TreeView*

- *Code Editor*, berfungsi untuk menulis dan menyunting kode program. Lokasi kode editor ada di belakang form. Pada gambar 2.28 adalah contoh tampilan kode editor.



Gambar 2.28. *Windows Code Editor*