BAB II

LANDASAN TEORI

2.1 Tinjauan Pustaka

Angin secara umum adalah setiap gerakan udara relatif terhadap permukaan bumi. Dalam pengertian teknis, yang dimaksud dengan angin adalah setiap gerakan udara yang mendatar atau hampir mendatar. Angin mempunyai arah dan kecepatan yang ditentukan oleh adanya perbedaan tekanan udara dipermukaan bumi.

Angin bertiup dari tempat bertekanan tinggi ke tempat bertekanan rendah. Semakin besar perbedaan tekanan udara semakin besar kecepatan angin, aplikasi ini hanyalah prototype kecil yang menggambarkan sebuah aplikasi untuk alat petunjuk arah angin dan kecepatan angin, dibandingkan dengan peralatan yang sesungguhnya aplikasi ini mungkin belum sempurna. Otak dari aplikasi alat petunjuk arah angin dan kecepatan angin adalah Mikrokontroler AT89C51, Alat yang digunakan untuk mengukur kecepatan angin menggunakan sensor optocoupler sebagai transducer. Alat ini dibuat sedemikian hingga dapat mengukur kecepatan angin minimal 0,1 Km/Jam. Sedangkan untuk menunjukan arah angin menggunakan sensor *rotary encoder* yaitu suatu sensor digital yang keluarannya berupa bit-bit digital sehingga mampu memenunjukan arah angin dari 0^{o} hingga 360^{o} dengan ketelitian $0{,}5^{o}$. (Rizal Bonodin,2007).

Pemantauan kecepatan dan arah angin biasanya dilakukan dari jarak dekat, oleh sebab itu diperlukan pemantauan dari jarak jauh untuk efisiensi waktu. Dalam alat ini mikrokontroler AT89S51 digunakan sebagai pengendali seluruh sistem data akan ditampilkan pada Borland Delphi 7 kemudian setelah itu dikirim melalui SMS. Serta digunakan sensor optocoupler untuk mencacah putaran kecepatan angin dan menentukan arah angin, Selain itu digunakan sebuah handphone untuk mengirimkan data yang telah diperoleh secara periodik.

Sistem pemrosesan data menggunakan IC Mikrokontroler AT89S51 yang diprogram dengan bahasa assembler melalui program Reads 51, Data diinterfacekan ke komputer dengan pemograman Delphi 7.0 melalui sistem komunikasi serial untuk ditampilkan ke komputer dan disimpan dalam media penyimpanan data (harddisk) kemudian dikirim melalui SMS Pada peralatan ini dihasilkan pemrograman interfacing melalui Borland Delphi 7.0 dengan tampilan data kecepatan angin dan arah angin. Setiap pembacaan data akan disimpan pada database dan data tersebut akan dikirim melalui SMS secara periodik. (Ari Nurwati, 2008).

2.2 Landasan Teori

2.2.1 Mikrokontroler AT89S51

Mikrokontroler tipe AT89S51 merupakan mikrokontroler keluaran MCS-51 dengan konfigurasi yang sama persis dengan AT89C51 yang cukup terkenal, hanya saja AT89S51 mempunyai fitur ISP (*In-Sistem Programmable Flash*

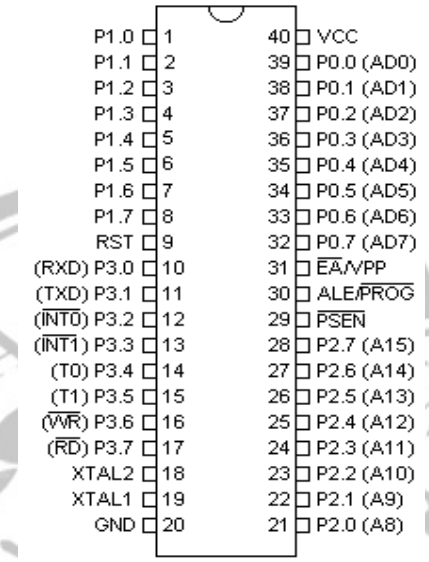
Memory). Fitur ini memungkinkan mikrokontroler dapat diprogram langsung dalam suatu sistem elektronik tanpa melalui *Programer board*. Mikrokontroler dapat diprogram langsung melalui kabel ISP yang dihubungkan dengan *paralel port* pada suatu *personal komputer*.

Adapun fitur yang dimiliki Mikrokontroler AT89S51 adalah sebagai berikut :

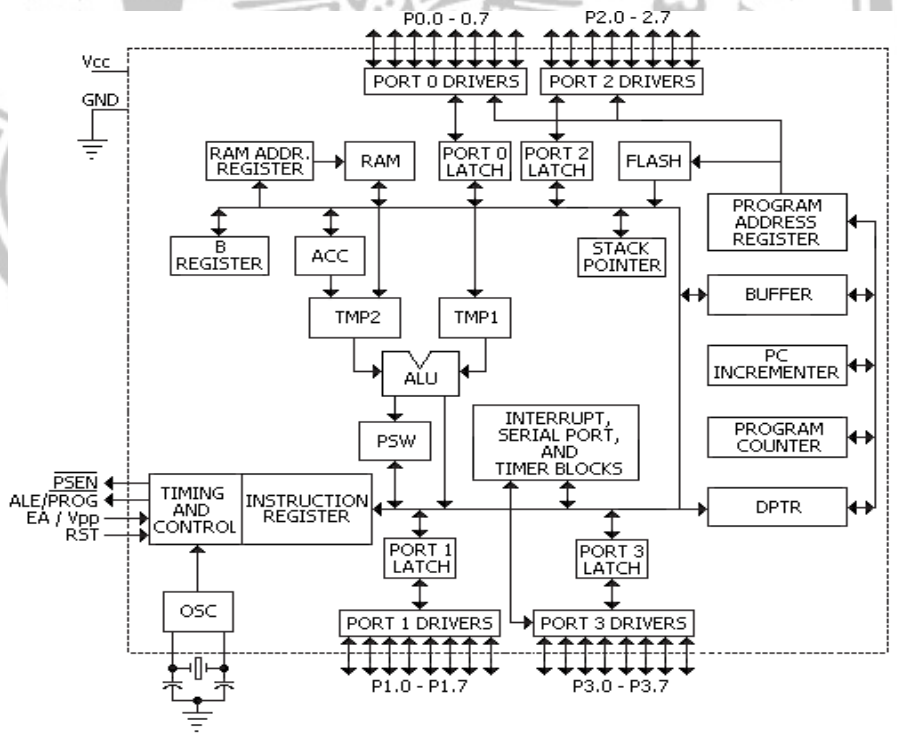
1. Sebuah CPU (*Central Processing Unit*) 8 bit yang termasuk keluarga MCS51.
2. Osilator internal dan rangkaian pewaktu, RAM internal 128 *byte* (*on chip*).
3. Empat buah *Programmable port* I/O, masing-masing terdiri atas 8 jalur I/O.
4. Dua buah *Timer counter* 16 *byte.*
5. Lima buah jalur interupsi (2 interupsi external dan 3 interupsi internal).
6. Sebuah port serial dengan kontrol *serial full duplex* UART.
7. Kemampuan melaksanakan operasi perkalian, pembagian dan operasi *Boolean* (bit).
8. Kecepatan pelaksanaan instruksi per siklus 1 mikrodetik pada frekuensi *clock* 12 MHz.
9. 4 *Kbytes Flash* ROM yang dapat diisi dan dihapus sampai 1000 kali.
10. *In-System Progammable Flash Memory.*

Dengan keistimewaan tersebut, pembuatan alat menggunakan AT89S51 menjadi lebih sederhana dan tidak memerlukan IC pendukung yang banyak. Sehingga mikrokontroler AT89S51 ini mempunyai keistimewaan dari segi perangkat keras.

Diagram pin dari AT89S51 ditunjukkan pada Gambar 2.1. Sedangkan pada Gambar 2.2 menunjukkan blok diagram AT89S51 secara detail.



Gambar 2.1 Diagram pin AT89S51 (*Data sheet* ATMEL)



Gambar 2.2 Diagram blok AT89S51 (*Data sheet* ATMEL)

Tabel 2.1 Diskripsi PIN AT89S51

No	Diskripsi PIN	Keterangan atau Fungsi
1	Vcc	Tegangan *supply* (+ 5V)
2	GND	*Ground*
3	Port 0	Port 0 merupakan jalur yang dapat digunakan sebagai *input/output* 8-bit *(bidirectional I/OPort)*. Port ini juga di multipleks untuk alamat rendah (A7..A0) dan data (D0..D7).
4	Port 1 (P1.0 – P1.7)	Merupakan *port* I/O 8 bit dengan *internal pull-up*. *Output* penyangga. *port* 1 dapat digunakan sebagai sumber untuk empat masukan TTL
5	Port 2	Port 2 adalah jalur yang dapat digunakan sebagai *input/output* dua arah (*bidirectional*) dengan *internal pullup*. *Port* ini dipakai untuk alamat tinggi (A15 – A8) ketika dipakai untuk ekternal memori, dapat dipakai untuk alamat tinggi selama pemprograman dan pengujian EEPROM internal.
6	Port 3 (P3.0 – P3.7)	Merupakan port I/O 8 bit *bidirectional*. *Port* ini juga mempunyai fungsi lain yaitu : RXD (P3.0) : masukan data *port* serial TXD (P3.1) : keluaran data *port* serial. INT0 (P3.2) : masukan interupsi 0 dari luar INT1 (P3.3) : masukan interupsi 1 dari luar.

Tabel 2.1 (lanjutan)

		T0 (P3.4) : masukan ke pencacah 0 T1 (P3.5) : masukan ke pencacah 1 WR (P3.6) : sinyal tulis untuk memori luar RD (P3.7) : sinyal baca untuk memori luar
7	RST(reset)	Transisi rendah ke tinggi (*Positive Going Triger*) dari pin ini akan mereset mikrokontroler.
8	ALE / PROG	Pin ini dipakai untuk menangkap atau me-*latch* alamat rendah ke memori eksternal selama operasi normal dan menerima masukan pulsa program selama pemrograman EPROM internal.
9	PSEN	Merupakan *Program Store Enable* dimana keluaran PSEN adalah sinyal kontrol yang mengijinkan atau mengaktifkan program memori eksternal (EPROM eksternal) ke bus data selama operasi normal
10	EA / Vpp	External *Access Enable*, EA harus dihubungkan ke *ground*, jika mikrokontroler akan mengeksekusi program dari memori eksternal lokasi 0000H hingga FFFFH. Selain itu, EA harus dihubungkan ke Vcc agar mikrokontroler mengakses program secara internal.
11	XTAL 1	Masukan ke osilator atau masukan sumber pulsa *(clock)* luar yang tersedia.
12	XTAL 2	Output dari osilator.

2.2.1.1 Sistem Interupsi AT89S51

Meskipun memerlukan pengertian yang lebih mendalam, pengetahuan mengenai interupsi sangat membantu mengatasi masalah pemrograman mikroprosesor/mikrokontroler dalam hal menangani banyak peralatan input/output. Pengetahuan mengenai interupsi tidak cukup hanya dibahas secara teori saja, diperlukan contoh program yang konkrit untuk memahami.

Saat kaki **RESET** pada IC mikroprosesor/mikrokontroler menerima sinyal reset (pada MCS51 sinyal tersebut berupa sinyal '**1**' sesaat, pada prosesor lain umumnya merupakan sinyal '**0**' sesaat), *Program Counter* diisi dengan sebuah nilai. Nilai tersebut dinamakan sebagai *vektor reset* (*reset vector*), merupakan nomor awal memori-program yang menampung program yang harus dijalankan.

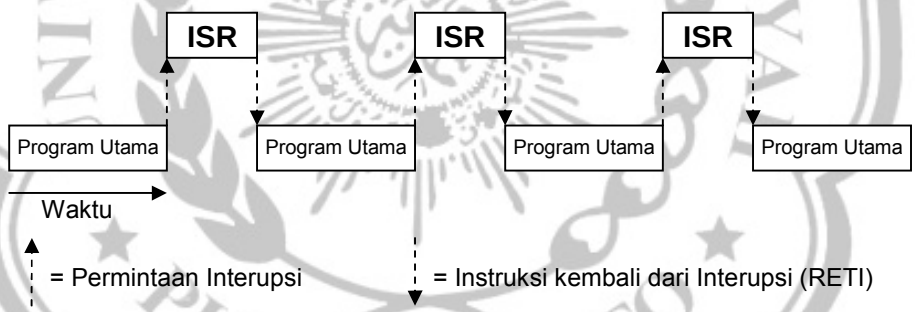
Pembahasan di atas memberi gambaran bahwa proses reset merupakan peristiwa perangkat keras (sinyal reset diumpankan ke kaki Reset) yang dipakai untuk mengatur kerja dari perangkat lunak, yakni menentukan aliran program prosesor (mengisi *Program Counter* dengan *vektor reset*).

Program yang dijalankan dengan cara reset, merupakan program utama bagi prosesor. Peristiwa perangkat keras yang dipakai untuk mengatur kerja dari perangkat lunak, tidak hanya terjadi pada proses reset, tapi terjadi pula dalam proses interupsi.

Dalam proses interupsi, terjadinya sesuatu pada perangkat keras tertentu dicatat dalam flip-flop khusus, flip-flop tersebut sering disebut sebagai 'penanda'

(*flag*), catatan dalam penanda tersebut diatur sedemikian rupa sehingga bisa merupakan sinyal permintaan interupsi pada prosesor. Jika permintaan interupsi ini dilayani prosesor, *Program Counter* akan diisi dengan sebuah nilai. Nilai tersebut dinamakan sebagai vektor interupsi(*interrupt vector*), yang merupakan nomor awal memori-program yang menampung program yang dipakai untuk melayani permintaan interupsi tersebut.

Program yang dijalankan dengan cara interupsi, dinamakan sebagai program layanan interupsi (**ISR** - *Interrupt Service Routine*). Saat prosesor menjalankan **ISR**, pekerjaan yang sedang dikerjakan pada program utama sementara ditinggalkan, selesai menjalankan **ISR** prosesor kembali menjalankan program utama, seperti yang digambarkan dalam Gambar 2.3



Gambar 2.3 Bagan kerja prosesor melayani interupsi

Sebuah prosesor bisa mempunyai beberapa perangkat keras yang merupakan sumber sinyal permintaan interupsi, masing-masing sumber interupsi dilayani dengan **ISR** berlainan, dengan demikian prosesor mempunyai beberapa *vektor interupsi* untuk memilih **ISR** mana yang dipakai melayani permintaan interupsi dari berbagai sumber. Kadang kala sebuah *vektor interupsi* dipakai oleh

lebih dari satu sumber interupsi yang sejenis, dalam hal semacam ini **ISR** bersangkutan harus menentukan sendiri sumber interupsi mana yang harus dilayani saat itu.

Jika pada saat yang sama terjadi lebih dari satu permintaan interupsi, prosesor akan melayani permintaan interupsi tersebut menurut perioritas yang sudah ditentukan, selesai melayani permintaan interupsi perioritas yang lebih tinggi, prosesor melayani permintaan interupsi berikutnya, baru setelah itu kembali mengerjakan program utama.

Saat prosesor sedang mengerjakan **ISR**, bisa jadi permintaan interupsi lain, jika permintaan interupsi yang datang belakangan ini mempunyai perioritas lebih tinggi, **ISR** yang sedang dikerjakan ditinggal dulu, prosesor melayani permintaan yang perioritas lebih tinggi, selesai melayani interupsi perioritas tinggi prosesor meneruskan **ISR** semula, baru setelah itu kembali mengerjakan program utama. Hal ini dikatakan sebagai interupsi bertingkat (*nested interrupt*), tapi tidak semua prosesor mempunyai kemampuan melayani interupsi secara ini.

2.2.1.2 Sumber interupsi AT89S51

Seperti terlihat dalam Gambar 2.4, AT89S51 mempunyai 6 sumber interupsi, yakni Interupsi External (*External Interrupt*) yang berasal dari kaki **INT0** dan **INT1**, *Interupsi Timer* (*Timer Interrupt*) yang berasal dari ***Timer*0** maupun ***Timer*1**, Interupsi Port Seri (*Serial Port Interrupt*) yang berasal dari bagian penerima dan bagian pengirim Port Seri.

Bit **IE0** (atau bit **IE1**) dalam **TCON** merupakan penanda (flag) yang menandakan adanya permintaan Interupsi Eksternal. Ada 2 keadaan yang bisa mengaktifkan penanda ini, yang pertama karena level tegangan '**0**' pada kaki **INT0** (atau **INT1**), yang kedua karena terjadi transisi sinyal '**1**' menjadi '**0**' pada kaki **INT0** (atau **INT1**). Pilihan bentuk sinyal ini ditentukan lewat bit **IT0** (atau bit **IT1**) yang terdapat dalam register **TCON**.

- Kalau bit **IT0** (atau **IT1**) ='**0**' maka bit **IE0** (atau **IE1**) dalam **TCON** menjadi '**1**' saat kaki **INT0**='**0**'.
- Kalau bit **IT0** (atau **IT1**) ='**1**' maka bit **IE0** (atau **IE1**) dalam **TCON** menjadi '**1**' saat terjadi transisi sinyal '**1**' menjadi '**0**' pada kaki **INT0**.

Menjelang prosesor menjalankan **ISR** dari *Interupsi Eksternal*, bit **IE0** (atau bit **IE1**) dikembalikan menjadi '**0**', menandakan permintaan *Interupsi Eksternal* sudah dilayani. Namun jika permintaan *Interupsi Eksternal* terjadi karena level tegangan '**0**' pada kaki **IT0** (atau **IT1**), dan level tegangan pada kaki tersebut saat itu masih ='**0**' maka bit **IE0** (atau bit **IE1**) akan segera menjadi '**1**' lagi.

Bit **TF0** (atau bit **TF1**) dalam **TCON** merupakan penanda (*flag*) yang menandakan adanya permintaan Interupsi Timer, bit **TF0** (atau bit **TF1**) menjadi '**1**' pada saat terjadi limpahan pada pencacah biner *timer0* (atau *timer1*).

Menjelang prosesor menjalankan **ISR** dari *Interupsi Timer*, bit **TF0** (atau bit **TF1**) dikembalikan menjadi '**0**', menandakan permintaan Interupsi *Timer* sudah dilayani.

Interupsi port seri terjadi karena dua hal, yang pertama terjadi setelah port seri selesai mengirim data 1 byte, permintaan interupsi semacam ini ditandai dengan penanda **TI**=’**1**’. Yang kedua terjadi saat port seri telah menerima data 1 byte secara lengkap, permintaan interupsi semacam ini ditandai dengan penanda **RI**=’**1**’.

Penanda di atas tidak dikembalikan menjadi ‘**0’** menjelang prosesor menjalankan **ISR** dari Interupsi port seri, karena penanda tersebut masih diperlukan **ISR** untuk menentukan sumber interupsi berasal dari **TI** atau **RI**. Agar port seri bisa dipakai kembali setelah mengirim atau menerima data, penanda-penanda tadi harus di-nol-kan lewat program.

Penanda permintaan interupsi semuanya bisa di-nol-kan atau di-satu-kan lewat instruksi, pengaruhnya sama kalau perubahan itu dilakukan oleh perangkat keras. Artinya permintaan interupsi bisa diajukan lewat pemrograman, misalnya permintaan interupsi eksternal **IT0** bisa diajukan dengan instruksi **IE0 = 1**.

2.2.1.3 Mengaktifkan Interupsi

Semua sumber permintaan interupsi yang dibahas di atas, masing-masing bisa diaktifkan atau dinonaktifkan secara tersendiri lewat bit-bit yang ada dalam register **IE** (*Interrupt Enable Register*).

Bit **EX0** dan **EX1** untuk mengatur interupsi eksternal **INT0** dan **INT1**, bit **ET0** dan **ET1** untuk mengatur interupsi *timer0* dan *timer1*, bit **ES** untuk mengatur interupsi port seri, seperti yang digambarkan dalam Gambar 2.20.

Di samping itu ada pula bit **EA** yang bisa dipakai untuk mengatur semua sumber interupsi sekaligus.

Setelah reset, semua bit dalam register **IE** bernilai '**0**', artinya sistem interupsi dalam keadaan non-aktif. Untuk mengaktifkan salah satu sistem interupsi, bit pengatur interupsi bersangkutan diaktifkan dan juga **EA** yang mengatur semua sumber interupsi. Misalnya instruksi yang dipakai untuk mengaktifkan interupsi ekternal **INT0** adalah **`EX0 = 1`** disusul dengan **`EA = 1`**.

2.2.1.4 Vektor Interupsi

Saat AT89S51 menanggapi permintaan interupsi, *Program Counter* diisi dengan sebuah nilai yang dinamakan sebagai vektor interupsi, yang merupakan nomor awal dari memori-program yang menampung **ISR** untuk melayani permintaan interupsi tersebut. Vektor interupsi itu dipakai untuk melaksanakan instruksi **LCALL** yang diaktifkan secara perangkat keras. Tabel 2.2 dibawah menujukkan alamat vektor dari tiap interupsi.

Tabel 2.2 Alamat vektor tiap interupsi

Interrupt #	Description	Vector Address
0	External 0	0x0003
1	Timer 0	0x000B
2	External 1	0x0013
3	Timer 1	0x001B
4	Serial	0x0023

Vektor interupsi untuk interupsi eksternal **INT0** adalah **0x03**, untuk interupsi *timer0* adalah **0x0B**, untuk interupsi ekternal **INT1** adalah **0x13**, untuk interupsi *timer1* adalah **0x1B** dan untuk interupsi port seri adalah **0x23**.

Jarak vektor interupsi satu dengan lainnya sebesar 8, atau hanya tersedia 8 byte untuk setiap **ISR**. Jika sebuah **ISR** memang hanya pendek saja, tidak lebih dari 8 byte, maka **ISR** tersebut bisa langsung ditulis pada memori-program yang disediakan untuknya. **ISR** yang lebih panjang dari 8 byte ditulis ditempat lain, tapi pada memori-program yang ditunjuk oleh *vektor interupsi* diisikan instruksi **JUMP** ke arah **ISR** bersangkutan.

2.2.1.5 Tingkatan Perioritas

Masing-masing sumber interupsi bisa ditempatkan pada dua tingkatan perioritas yang berbeda. Pengaturan tingkatan perioritas isi dilakukan dengan bit-bit yang ada dalam register **IP** (*Interrupt Priority*).

Bit **PX0** dan **PX1** untuk mengatur tingkatan perioritas interupsi eksternal **INT0** dan **INT1**, bit **PT0** dan **PT1** untuk mengatur interupsi *timer0* dan *timer1*, bit **PS** untuk mengatur interupsi port seri, seperti yang digambarkan dalam Gambar 2.4.

Setelah reset, semua bit dalam register **IP** bernilai '**0**', artinya semua sumber interupsi ditempatkan pada tingkatan tanpa perioritas. Masing-masing sumber interupsi bisa ditempatkan pada tingkatan perioritas utama dengan cara men-'satu'-kan bit pengaturnya. Misalnya interupsi *timer0* bisa ditempatkan pada tingkatan perioritas utama dengan instruksi **`PT1 = 1`**.

Sebuah ISR untuk interupsi tanpa perioritas bisa diinterupsi oleh sumber interupsi yang berada dalam tingkatan perioritas utama. Tapi interupsi yang berada pada tingkatan perioritas yang sama, tidak dapat saling meng-interupsi.

Jika 2 permintaan interupsi terjadi pada saat yang bersamaan, sedangkan kedua interupsi tersebut terletak pada tingkatan perioritas yang berlainan, maka interupsi yang berada pada tingkatan perioritas utama akan dilayani terlebih dulu, setelah itu baru melayani interupsi pada tingkatan tanpa perioritas.

Jika kedua permintaan tersebut bertempat pada tingkatan perioritas yang sama, perioritas akan ditentukan dengan urutan sebagai berikut: interupsi eksternal **INT0**, interupsi *timer0*, interupsi ekternal **INT1**, interupsi *timer1* dan terakhir adalah interupsi port seri.

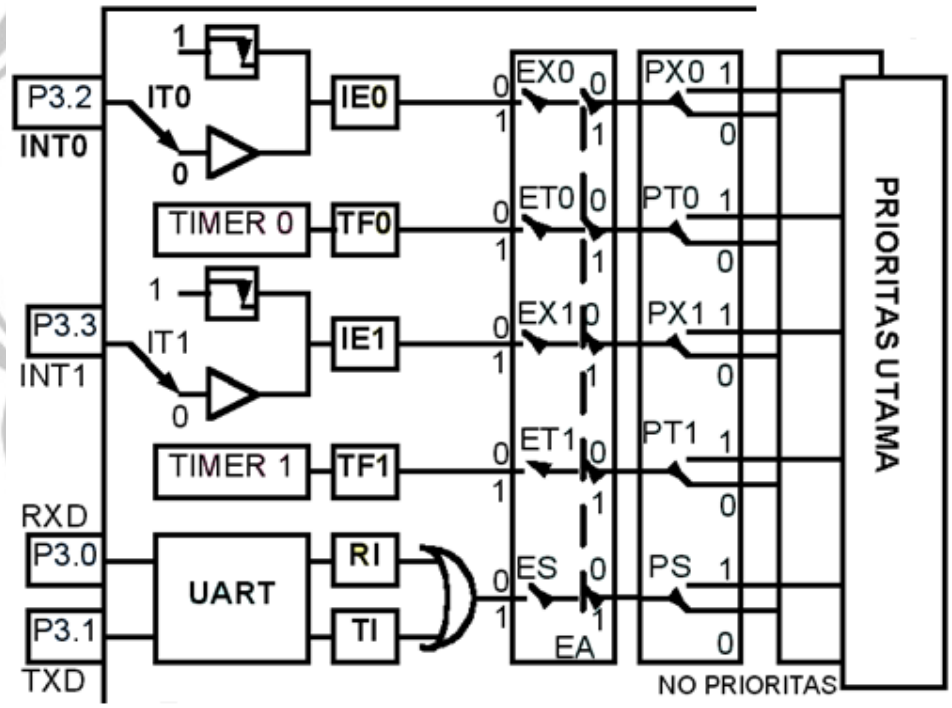
2.2.1.6 Bagan Lengkap Sistem Interupsi MCS51

Meskipun sistem interupsi AT89S51 termasuk sederhana dibandingkan dengan sistem interupsi MC68HC11 buatan Motorola, tapi karena menyangkut 5 sumber interupsi yang masing-masing harus diatur secara tersendiri, tidak mudah untuk mengingat semua masalah tersebut, terutama pada saat membuat program sering dirasakan sangat merepotkan membolak-balik buku untuk mengatur masing-masing sumber interupsi tersebut.

Gambar 2.4 menggambarkan sistem interupsi AT89S51 selangkapnya, berikut dengan masing-masing bit dalam register-register **SFR** (*Special Function Register*) yang dipakai untuk mengatur masing-masing sumber interupsi.

Saklar yang digambarkan dalam Gambar 2.4 mewakili bit dalam register yang harus diatur untuk mengendalikan sumber interupsi, kotak bergambar bendera kecil merupakan *flag* (petanda) dalam register yang mencatat adanya permintaan interupsi dari masing-masing sumber interupsi. Kedudukan saklar dalam gambar tersebut menggambarkan kedudukan awal setelah AT89S51 direset.

Gambar 2.4 ini sangat membantu saat penulisan program menyangkut interupsi AT89S51.



Gambar 2.4 Bagan Lengkap Sistem Interupsi AT89S51

2.2.1.7 Pedoman pembuatan ISR

Layanan interupsi oleh AT89S51 dilakukan dengan menjalankan program setara dengan **LCALL** mulai dari memori-program yang ditunjuk oleh *vektor interupsi*, sampai MCS51 menjumpai instruksi **RETI** (*Return from Interrupt*). Intstruksi **RETI** memberitahu AT89S51 bahwa **ISR** sudah selesai dikerjakan, kemudian mengambil kembali isi *Program Counter* yang sebelumnya disimpan ke dalam *Stack*. Dengan demikian MCS51 akan melanjutkan kembali pekerjaan di program utama yang ditinggal untuk melayani interupsi.

Instruksi **RET** memang bisa dipakai untuk meninggalkan **ISR** dan melanjutkan kerja program utama. Tapi **RET** tidak bisa menghentikan proses interupsi, akibatnya sebelum menjumpai instruksi **RETI**, AT89S51 tidak akan melayani permintaan interupsi yang lain.

2.2.1.8 Timer dan Counter dalam AT89S51

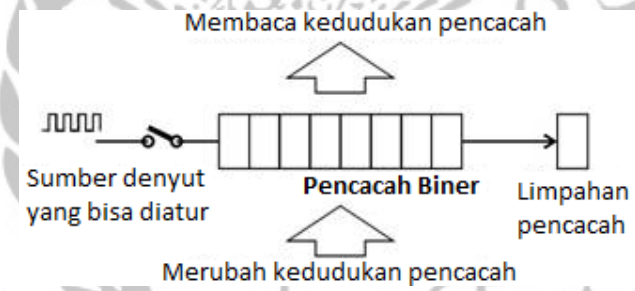
Timer dan Counter merupakan sarana input yang kurang dapat perhatian pemakai mikrokontroler, dengan sarana input ini mikrokontroler dengan mudah bisa dipakai untuk mengukur lebar pulsa, membangkitkan pulsa dengan lebar yang pasti, dipakai dalam pengendalian tegangan secara PWM (*Pulse Width Modulation*) dan sangat diperlukan untuk aplikasi *remote control* dengan infra merah.

Pada dasarnya sarana input yang satu ini merupakan seperangkat pencacah biner (*binary counter*) yang terhubung langsung ke saluran-data

mikrokontroler, sehingga mikrokontroler bisa membaca kedudukan pancacah, bila diperlukan mikrokontroler dapat pula merubah kedudukan pencacah tersebut.

Seperti layaknya pencacah biner, bilamana sinyal denyut (clock) yang diumpankan sudah melebihi kapasitas pencacah, maka pada bagian akhir untaian pencacah akan timbul sinyal limpahan, sinyal ini merupakan suatu hal yang penting sekali dalam pemakaian pencacah. Terjadinya limpahan pencacah ini dicatat dalam sebuah flip-flop tersendiri.

Di samping itu, sinyal denyut yang diumpankan ke pencacah harus pula bisa dikendalikan dengan mudah. Hal-hal yang dibicarakan di atas diringkas dalam Gambar 2.5.



Gambar 2.5 Konsep dasar *Timer/Counter* sebagai sarana input

Sinyal denyut yang diumpankan ke pencacah bisa dibedakan menjadi 2 macam, yang pertama yaitu sinyal denyut dengan frekuensi tetap yang sudah diketahui besarnya dan yang kedua adalah sinyal denyut dengan frekuensi tidak tetap.

Jika sebuah pencacah bekerja dengan frekuensi tetap yang sudah diketahui besarnya, dikatakan pencacah tersebut bekerja sebagai *timer*, karena

kedudukan pencacah tersebut setara dengan waktu yang bisa ditentukan dengan pasti.

Jika sebuah pencacah bekerja dengan frekuensi yang tidak tetap, dikatakan pencacah tersebut bekerja sebagai *counter*, kedudukan pencacah tersebut hanyalah menyatakan banyaknya pulsa yang sudah diterima pencacah.

Untaian pencacah biner yang dipakai, bisa merupakan pencacah biner menaik (*count up binary counter*) atau pencacah biner menurun(*count down binary counter*).

Timer/Counter sebagai sarana input banyak dijumpai dalam mikrokontroler, misalnya mikrokontroler keluarga MCS48, keluarga AT89S51 ataupun MC68HC11 semuanya memiliki *Timer/Counter* di dalam IC sebagai sarana *input*. Di samping itu bisa pula dijumpai IC *Timer/Counter* yang berdiri sendiri sebagai penunjang kerja mikroprosesor, misalnya 8253/8254 *Programmable Interval Timer* buatan Intel, atau MC6840 *Programmable Counter/Timer* buatan Motorola.

2.2.1.9 Sarana Timer/Counter dalam MCS51

Keluarga mikrokontroler MCS51, misalnya AT89S51 dan AT89Cx051, dilengkapi dengan dua perangkat *Timer/Counter*, masing-masing dinamakan sebagai *Timer0* dan *Timer1*. Sedangkan untuk jenis yang lebih *besar*, misalnya AT89C52, mempunyai tambahan satu perangkat *Timer/Counter* lagi yang dinamakan sebagai *Timer2*.

Perangkat *Timer/Counter* tersebut merupakan perangkat keras yang menjadi satu dalam IC mikrokontroler AT89S51, bagi pemakai mikrokontroler AT89S51 perangkat tersebut dikenal sebagai **SFR** (*Special Function Register*) yang berkedudukan sebagai *memori-data internal*.

Pencacah biner untuk *Timer0* dibentuk dengan register **TL0** (*Timer 0 Low Byte*, memori-data internal nomor 0x6A) dan register **TH0** (*Timer 0 High Byte*, memori-data internal nomor 0x6C).

Pencacah biner untuk *Timer 1* dibentuk dengan register **TL1** (*Timer 1 Low Byte*, memori-data internal nomor 0x6B) dan register **TH1** (*Timer 1 High Byte*, memori-data internal nomor 0x6D).

Pencacah biner pembentuk *Timer/Counter* AT89S51 merupakan pencacah biner menaik (*count up binary counter*) yang mencacah dari **0x0000** sampai **0xFFFF**, saat kedudukan pencacah berubah dari **0xFFFF** kembali ke **0x0000** akan timbul sinyal limpahan.

Untuk mengatur kerja *Timer/Counter* dipakai 2 register tambahan yang dipakai bersama oleh *Timer 0* dan *Timer 1*. Register tambahan tersebut adalah register **TCON** (*Timer Control Register*, memori-data internal nomor 0x88, bisa dialamat secara bit) dan register **TMOD** (*Timer Mode Register*, memori-data internal nomor 0x89).

2.2.1.10 Pencacah biner Timer 0 dan 1

TL0, **TH0**, **TL1** dan **TH1** merupakan **SFR** (*Special Function Register*) yang dipakai untuk membentuk pencacah biner perangkat *Timer 0* dan *Timer 1*.

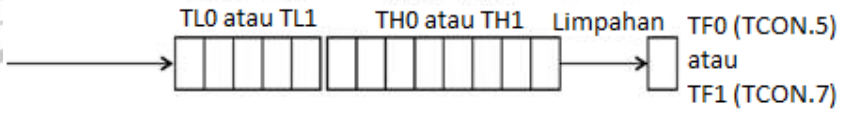
Kapasitas keempat register tersebut masing-masing 8 bit, bisa disusun menjadi 4 macam Mode pencacah biner seperti terlihat dalam Gambar 2.6 sampai Gambar 2.9.

Pada Mode 0, Mode 1 dan Mode 2 *Timer 0* dan *Timer 1* masing-masing bekerja sendiri, artinya bisa dibuat *Timer 0* bekerja pada Mode 1 dan *Timer 1* bekerja pada Mode 2, atau kombinasi mode lainnya sesuai dengan keperluan.

Pada Mode 3 **TL0**, **TH0**, **TL1** dan **TH1** dipakai bersama-sama untuk menyusun sistem *timer* yang tidak bisa di-kombinasi lain.

Susunan **TL0**, **TH0**, **TL1** dan **TH1** pada masing-masing mode adalah sebagai berikut:

Mode 0 – Pencacah Biner 13 bit

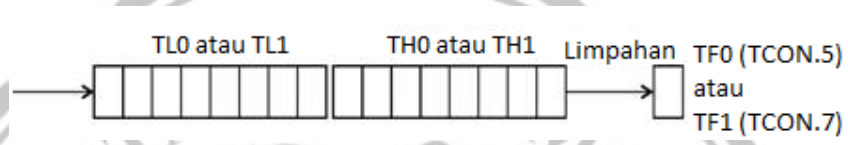


Gambar 2.6 Mode 0 - Pencacah Biner 13 Bit

Pencacah biner dibentuk dengan **TLx** (maksudnya bisa **TL0** atau **TL1**) sebagai pencacah biner 5 bit (meskipun kapasitas sesungguhnya 8 bit), limpahan dari pencacah biner 5 bit ini dihubungkan ke **THx** (maksudnya bisa **TH0** atau **TH1**) membentuk sebuah untaian pencacah biner 13 bit, limpahan dari pencacah 13 bit ini ditampung di flip-flop **TFx** (maksudnya bisa **TF0** atau **TF1**) yang berada di dalam register **TCON**.

Mode ini meneruskan sarana *Timer* yang ada pada mikrokontroler MCS48 (mikrokontroler pendahulu AT89S51), dengan maksud rancangan alat yang dibuat dengan MCS48 bisa dengan mudah diadaptasikan ke MCS51. Mode ini tidak banyak dipakai lagi.

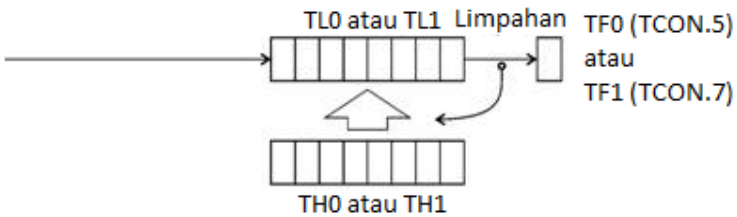
Mode 1 – Pencacah Biner 16 bit



Gambar 2.7 Mode 1 - Pencacah Biner 16 Bit

Mode ini sama dengan *Mode 0*, hanya saja register **TLx** dipakai sepenuhnya sebagai pencacah biner 8 bit, sehingga kapasitas pencacah biner yang tersebentuk adalah 16 bit. Seiring dengan sinyal denyut, kedudukan pencacah biner 16 bit ini akan bergerak dari 0x0000 (biner 0000 0000 0000 0000), 0x0001, 0x0002 … sampai 0xFFFF (biner 1111 1111 1111 1111), kemudian melimpah kembali menjadi 0x0000.

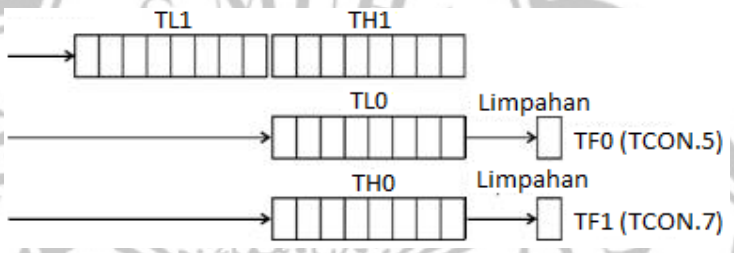
Mode 2 – Pencacah Biner 8 bit dengan Isi Ulang



Gambar 2.8 Mode 2 - Pencacah Biner 8 Bit dengan Isi Ulang

TLx dipakai sebagai pencacah biner 8 bit, sedangkan **THx** dipakai untuk menyimpan nilai yang diisikan ulang ke **TLx**, setiap kali kedudukan **TLx** melimpah (berubah dari 0xFF menjadi 0x00). Dengan cara ini bisa didapatkan sinyal limpahan yang frekuensinya ditentukan oleh nilai yang disimpan dalam **TH0**.

Mode 3 – Gabungan Pencacah Biner 16 bit dan 8 Bit

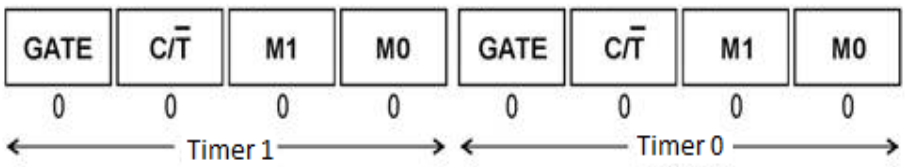


Gambar 2.9 Mode 3 – Gabungan Pencacah Biner 16 Bit dan 8 Bit

Pada Mode 3 **TL0**, **TH0**, **TL1** dan **TH1** dipakai untuk membentuk 3 untaian pencacah, yang pertama adalah untaian pencacah biner 16 bit tanpa fasilitas pemantau sinyal limpahan yang dibentuk dengan **TL1** dan **TH1**. Yang kedua adalah **TL0** yang dipakai sebagai pencacah biner 8 bit dengan **TF0** sebagai sarana pemantau limpahan. Pencacah biner ketiga adalah **TH0** yang dipakai sebagai pencacah biner 8 bit dengan **TF1** sebagai sarana pemantau limpahan.

2.2.1.11 Register Pengatur Timer

Register **TMOD** dan register **TCON** merupakan register pembantu untuk mengatur kerja *Timer0* dan *Timer1*, kedua register ini dipakai bersama oleh *Timer0* dan *Timer1*.

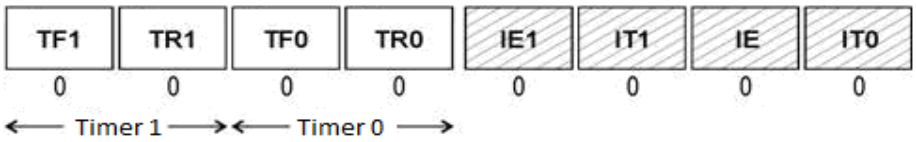


Gambar 2.10 Denah susunan bit dalam register TMOD

Register **TMOD** dibagi menjadi 2 bagian secara simitris, bit 0 sampai 3 register **TMOD** (**TMOD** bit 0 .. **TMOD** bit 3) dipakai untuk mengatur *Timer 0*, bit 4 sampai 7 register **TMODE** (**TMOD** bit 4 .. **TMOD** bit 7) dipakai untuk mengatur *Timer 1*, pemakaiannya sebagai berikut :

- Bit **M0/M1** dipakai untuk menentukan Mode *Timer*seperti yang terlihat dalam tabel di Gambar 2.11.
- Bit **C/T*** dipakai untuk mengatur sumber sinyal denyut yang diumpankan ke pencacah biner. Jika **C/T***=0 sinyal denyut diperoleh dari osilator kristal yang frekuensinya sudah dibagi 12, sedangkan jika **C/T***=1 maka sinyal denyut diperoleh dari kaki **T0** (untuk *Timer 0*) atau kaki **T1** (untuk *Timer 1*).
- Bit **GATE** merupakan bit pengatur saluran sinyal denyut. Bila bit **GATE=0** saluran sinyal denyut hanya diatur oleh bit **TRx** (maksudnya adalah **TR0** atau **TR1** pada register **TCON**). Bila bit **GATE=1** kaki

INT0 (untuk *Timer 0*) atau kaki **INT1** (untuk *Timer 1*) dipakai juga untuk mengatur saluran sinyal denyut (lihat Gambar 2.12).



Gambar 2.11 Denah susunan bit dalam register TCON

Register **TCON** dibagi menjadi 2 bagian, 4 bit pertama (bit 0 .. bit 3, bagian yang diarsir dalam Gambar 2.11) dipakai untuk keperluan mengatur kaki **INT0** dan **INT1**. Sisa 4 bit dari register **TCON** (bit 4..bit 7) dibagi menjadi 2 bagian secara simitris yang dipakai untuk mengatur *Timer0*/*Timer 1*, sebagai berikut:

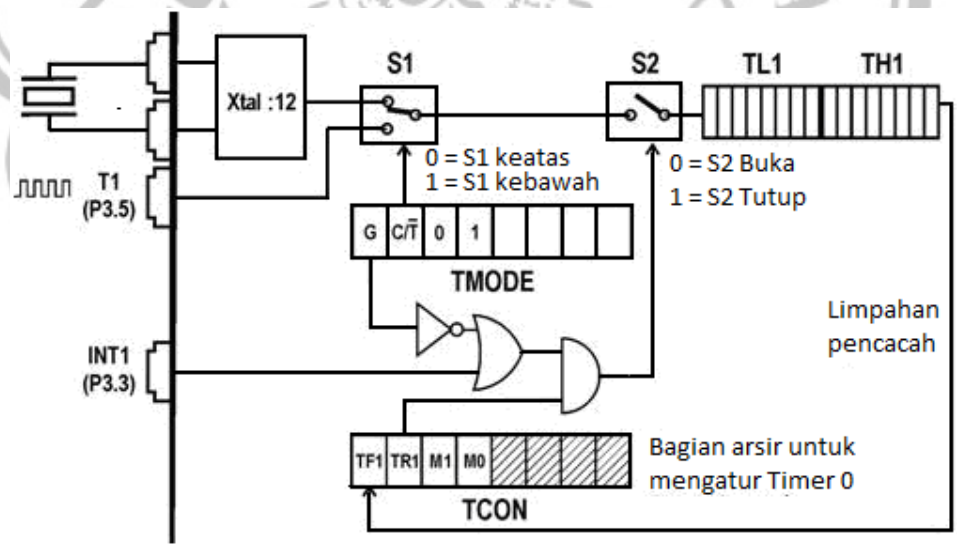
- Bit **TFx** (maksudnya adalah **TF0** atau **TF1)** merupakan bit penampung limpahan, **TFx** akan menjadi '**1**' setiap kali pencacah biner yang terhubung padanya melimpah (kedudukan pencacah berubah dari 0xFFFF kembali menjadi 0x0000). Bit **TFx** di-nol-kan dengan istruksi **CLRTF0** atau **CLRTF1**. Jika sarana interupsi dari *Timer 0*/*Timer 1* dipakai, **TRx** di-nol-kan saat AT89S51 menjalankan *rutin layanan interupsi* (**ISR**-*Interupt Service Routine*).
- Bit **TRx** (maksudnya adalah **TR0** atau **TR1)** merupakan bit pengatur saluran sinyal denyut, bila bit ini sama dengan **0** sinyal denyut tidak disalurkan ke pencacah biner sehingga pencacah berhenti mencacah. Bila bit **GATE** pada register **TMOD** =**1**, maka saluran sinyal denyut ini

diatur bersama oleh **TRx** dan sinyal pada kaki **INT0**/**INT1** (lihat Gambar 2.12).

2.2.1.12 Mengatur Timer

Gambar 2.12 merupakan bagan susunan rangkaian yang bisa terjadi pada *Timer 1* secara lengkap, digambarkan pula hubungan-hubungan semua register pembentuk dan pengatur *Timer 1*. Gambar ini berlaku pula untuk *Timer 0*.

Dalam pemakaian sesungguhnya, rangkaian yang dipakai hanya sebagian dari rangkaian lengkap tersebut, sesuai dengan keperluan sistem yang dibangun. Rangkaian yang dikehendaki dibentuk dengan mengatur register **TMODE**, sedangkan kerja dari *Timer* dikendalikan lewat register **TCON**.



Gambar 2.12 Skema lengkap *Timer* 1 dalam Mode 1

Setelah AT89S51 di-reset register **TMOD** bernilai 0x00, hal ini berarti:

- bit **C/T*** =’**0**’, menurut Gambar 2.12 keadaan ini membuat saklar **S1** ke posisi atas, sumber sinyal denyut berasal dari osilator kristal yang frekuensinya sudah dibagi 12, pencacah biner yang dibentuk dengan **TL1** dan **TH1** berfungsi sebagai ***timer***. Jika sistem yang dirancang memang menghendaki *Timer 1* bekerja sebagai ***timer*** maka bit **C/T*** tidak perlu diatur lagi. Tapi jika sistem yang dirancang menghendaki agar *Timer 1* bekerja sebagai ***counter*** untuk menghitung pulsa yang masuk lewat kakai **T1** (**P3.5**), maka posisi saklar **S1** harus dikebawahkan dengan membuat bit **C/T*** menjadi ‘**1**’.
- bit **GATE**=’**0**’, hal ini membuat output gerbang OR selalu ‘**1**’ tidak dipengaruhi keadaan ‘**0**’ atau ‘**1**’ pada kaki **INT1** (**P3.3**). Dalam keadaan semacam ini, saklar **S2** hanya dikendalikan lewat bit **TR1** dalam register **TCON**. Jika **TR1**=**1** saklar **S2** tertutup sehingga sinyal denyut dari **S1** disalurkan ke sistem pencacah biner, aliran sinyal denyut akan dihentikan jika **TR**=**0**. Sebaliknya jika bit **GATE**=**1**, output gerbang OR akan mengikuti keadaan kaki **INT1**, saat **INT1**=**0** apa pun keadaan bit **TR1** output gerbang AND selalu =**0** dan saklar **S1** selalu terbuka, agar saklar **S1** bisa tertutup kaki **INT1** dan bit **TR1** harus =**1** secara bersamaan. Jika sistem yang dirancang menghendaki kerja dari *timer/counter* dikendalikan dari sinyal yang berasal dari luar IC, maka bit **GATE** harus dibuat menjadi **1**

- bit **M1** dan **M0**=**0**, berarti **TL1** dan **TH1** disusun menjadi pencacah biner 13 bit (Mode 0), jika dikehendaki *Timer 1* bekerja pada mode 1 seperti terlihat dalam Gambar 2.12, maka bit **M1** harus dibuat menjadi '**0**' dan bit **M0** menjadi '**1'**.

Setelah reset **TMOD** bernilai **0x00**, berarti *Timer 1* bekerja sebagai pencacah biner 13 bit, sumber sinyal denyut dari osilator kristal atau *Timer 1* bekerja sebagai '***timer***', bit **GATE** ='**0**' berarti kaki **INT1** tidak berpengaruh pada rangkaian sehingga *Timer 1* hanya dikendalikan dari bit **TR1**.

Dalam pemakaian biasanya dipakai pencacah biner 16 bit, untuk keperluan itu instruksi yang diperlukan untuk mengatur **TMOD** adalah :

```
TMOD=0x10;
```

Bilangan **0x00** atau biner **%00010000** diisikan ke **TMOD**, berakibat bit 7 **TMOD** (bit **GATE**) bernilai '**0**', bit 6 (bit **C/T***) bernilai '**0**', bit 5 dan 4 (bit **M1** dan **M0**) bernilai '**01'**, ke-empat bit ini dipakai untuk mengatur *Timer 1*, sehingga *Timer* 1 bekerja sebagai ***timer*** dengan pencacah biner 16 bit yang dikendalikan hanya dengan TR1.

Jika dikehendaki pencacah biner dipakai sebagai counter untuk mencacah jumlah pulsa yang masuk lewat kaki **T1** (**P3.5**), instruksinya menjadi :

```
TMOD = 0x50;
```

Perbedaannya dengan instruksi di atas adalah dalam instruksi ini bit 6 (bit **C/T***) bernilai '**1**'. Selanjutnya jika diinginkan sinyal dari perangkat keras di luar

IC MCS51 bisa ikut mengendalikan *Timer 1*, instruksi pengatur *Timer 1* akan menjadi :

```
TMOD = 0xD0;
```

Dalam hal ini bit 7 (bit **GATE**) bernilai '**1**'.

Setelah mengatur konfigurasi *Timer 0* seperti di atas, pencacah biner belum mulai mencacah sebelum diperintah dengan instruksi :

```
TR1  = 1;
```

Perlu diingatkan jika bit **GATE** = '**1**', selama kaki **INT1** bernilai '**0**' pencacah biner belum akan mencacah. Untuk menghentikan proses pencacahan, dipakai instruksi

```
TR1 = 0;
```

Di atas hanya dibahas *Timer1* saja, tata cara untuk *Timer0* persis sama. Yang perlu diperhatikan adalah register **TMOD** dipakai untuk mengatur *Timer0* dan juga *Timer1*, sedangkan **TMOD** tidak bisa dialamati secara bit (*non bit addressable*) sehingga jika jika kedua *timer* dipakai, pengisian bit-bit dalam register **TMOD** harus dipikirkan sekaligus untuk *Timer0* dan *Timer1*.

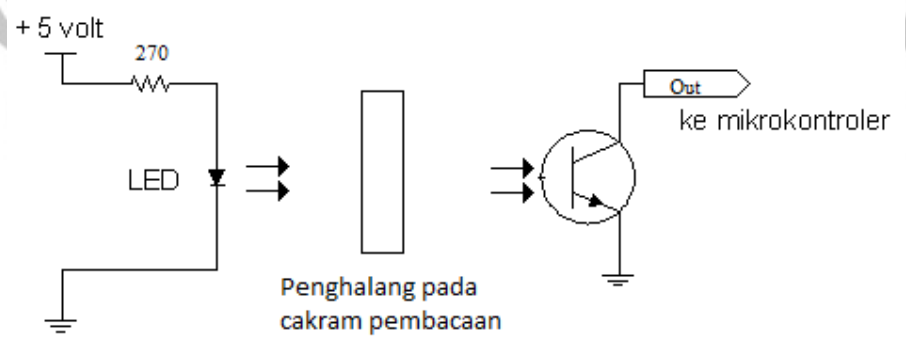
Bit **TR1** dan **TR0** yang dipakai untuk mengendalikan proses pencacahan, terletak di dalam register **TCON** (memori-data internal nomor 0x88) yang bisa dialamati secara bit (*bit addressable*). Sehingga **TR0** dan **TR1** bisa diatur secara terpisah (dengan perintah **SETB** atau **CLR**), tidak seperti mengatur **TMOD** yang harus dilakukan secara bersamaan.

Demikian pula bit penampung limpahan pencacah biner **TF0** dan **TF1**, juga terletak dalam register **TCON** yang masing-masing bisa di-monitor sendiri.

2.2.2 Sensor Cahaya

Sensor cahaya adalah sensor yang terdiri dari susunan LED dan Foto transistor yang dipasang berhadapan. Ada kalanya LED yang digunakan adalah LED inframerah, dapat juga LED merah biasa, namun bagian penerima biasanya berupa Fototransisitor.

Fototransistor pada sensor cahaya ini akan menghantar setiap kali cahaya LED pasangan yang ada dihadapannya tepat jatuh mengenainya. Oleh karena itu jika terhalang, fototransistor ini tidak menghantar dan berada dalam status *cut off* dan tidak ada arus listrik yang mengalir padanya.



Gambar 2.13 Bentuk Sensor Cahaya

Sensor cahaya ini dikenal dengan nama *opto coupler* dan biasanya berbentuk kotak dengan warna hitam dan berukuran sebesar ibu jari dengan 2 atau 3 kaki yang sejajar. Antara penerima dan pengirim terdapat celah yang dapat

dipergunakan untuk sisipan perangkat lain seperti cakram dekoder dan kertas berlubang.

2.2.3 Catu Daya

Perangkat elektronika mestinya dicatu oleh suplai arus searah DC (*direct current*) yang stabil agar dapat bekerja dengan baik. Baterai atau *accu* adalah sumber catu daya DC yang paling baik. Namun untuk aplikasi yang membutuhkan catu daya lebih besar, sumber dari baterai tidak cukup. Sumber catu daya yang besar adalah sumber bolak-balik AC (*Alternating Current*) dari pembangkit tenaga listrik. Untuk itu diperlukan suatu perangkat catu daya yang dapat mengubah daya masukan AC menjadi daya DC. Tegangan keluaran dapat berubah oleh perubahan tegangan saluran AC dan arus beban.

Parameter-parameter yang menentukan kelayakan adalah:

a) Kerut, komponen AC di tegangan keluaran yang ditumpangkan pada komponen DC.
b) Regulasi saluran, perubahan harga konstan (*steady state*) pada tegangan keluaran yang menyertai ubahan di tegangan saluran, sementara kondisi lain konstan.
c) Regulasi beban, perubahan harga konstan pada tegangan keluaran yang menyertai perubahan arus beban, sementara kondisi lain konstan.

2.2.3.1 Karakteristik Catu Daya

Mutu catu daya tergantung dari tegangan beban, arus beban, pengaturan tegangan, dan faktor-faktor lainnya. Karakteristik catu daya yang diatur adalah.

2.2.3.1.1 Regulasi Beban

Regulasi beban atau efek beban ditentukan sebagai perubahan tegangan keluar yang diatur bila arus beban berubah dari harga minimum ke harga maksimum.

$$LR = V_{NL} - V_{FL} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2.1)$$

Keterangan

LR = regulasi beban (*load regulation*)

V_{NL} = tegangan beban tanpa arus beban

V_{FL} = tegangan beban dengan arus beban penuh

Regulasi beban sering diungkapkan dalam persen dengan membagi perubahan pada tegangan beban dengan tegangan tanpa beban:

$$\%LR = \frac{V_{NL} - V_{FL}}{V_{NL}} x100\% \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2.2)$$

Keterangan

$\%LR$ = persen regulasi beban

V_{NL} = tegangan beban tanpa arus beban

V_{FL} = tegangan beban dengan arus beban penuh

2.2.3.1.2 Regulasi Sumber

Regulasi sumber disebut juga efek sumber atau regulasi jala-jala adalah perubahan pada tegangan beban yang diatur untuk jangkauan tegangan jala-jala tertentu, khasnya 115 V ± 10%, yaitu jangkauan sekitar 103 V sampai 127 V.

$$SR = V_{Awal} - V_{Akhir} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2.3)$$

Catu daya yang bermutu baik seperti Hewlett Packard 6214 A mempunyai regulasi sumber 4 mV. Persen regulasi sumber adalah

$$\%SR = \frac{SR}{V_{nom}} x100\% \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2.4)$$

Keterangan

%SR = persen regulasi sumber

SR = perubahan tegangan beban pada perubahan penuh jala-jala

V_{nom} = tegangan beban nominal

2.2.3.1.3 Impedansi Keluar

Catu daya yang diatur adalah sumber tegangan DC yang amat kaku. Ini berarti bahwa impedansi keluar pada frekuensi rendah amat kecil. Penggunaan umpan balik tegangan mengurangi impedansi keluar lebih jauh lagi karena

$$r_{out(t)} = \frac{r_{out}}{1 + AB} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2.5)$$

Catu daya yang diatur mempunyai impedansi keluar yang khas dalam besaran miliohm ($m\Omega$).

2.2.3.1.4 Penolakan Riak

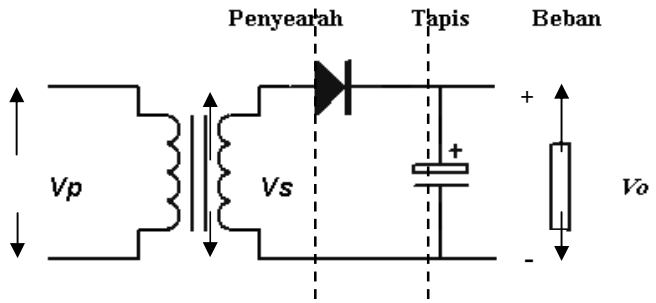
Pengatur tegangan memantapkan tegangan keluar terhadap perubahan tegangan masuk. Riak ekivalen dengan perubahan tegangan masuk, jadi pengatur tegangan meredam riak yang dating bersama-sama dengan tegangan masuk. Penolakan riak (*ripple rejection* = RR) biasanya disebutkan dalam desibel. Misalnya, RR = 80 dB berarti bahwa riak keluarnya 80 dB lebih kecil daripada riak masuknya. Pada bilangan biasa, ini berarti bahwa riak keluarnya 10.000 kali lebih kecil daripada riak masuknya.

2.2.3.2 Jenis-jenis Catu Daya

Catu daya yang banyak digunakan dalam praktek di lapangan adalah

2.2.3.2.1 Penyearah Tunggal atau Penyearah Setengah Gelombang

Catu daya dengan jenis penyearah setengah gelombang ditunjukkan pada Gambar 2.14.



Gambar 2.14 Catu daya dengan penyearah setengah gelombang.

Pada tiap denyut positif (hasil penyearah dioda), kapasitor diisi muatan, kemudian membuang muatannya ke beban. Penyusutan tegangan pada kapasitor (*Vc*) bergantung pada kapasitas C dan besarnya beban (Ω). Semakin besar beban, tegangan keluaran (*Vo*) akan semakin konstan. Semakin besar kapasitas C, maka penyearahan akan semakin baik. Besarnya frekuensi keluaran sama dengan frekuensi masukan. Besarnya arus maksimum ditunjukkan pada persamaan berikut

$$I_{maks} = \frac{V_{R-maks}}{R} \quad \text{.....} \quad (2.6)$$

$$V_R = \frac{V_p}{\pi} \quad \text{.....} \quad (2.7)$$

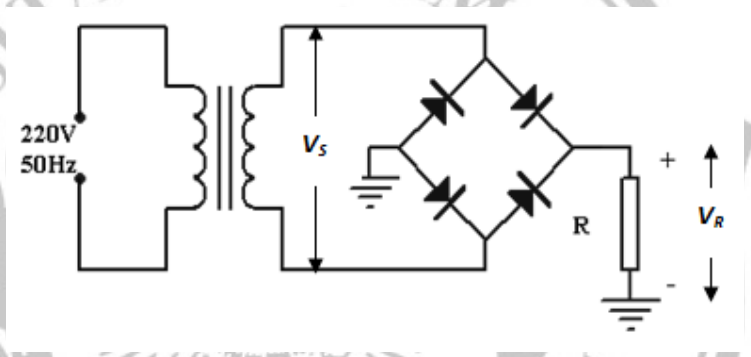
2.2.3.2.2 Penyearah gelombang penuh (full-wave rectifier)

Saat digunakan sebagai penyearah gelombang penuh, dioda secara bergantian menyearahkan tegangan AC pada saat siklus positif dan negatif. Penyearah gelombang penuh ada 2 macam dan penggunaannya disesuaikan dengan transformator yang dipakai. Untuk transformator biasa digunakan jembatan dioda (dioda bridge) sementara untuk transformator CT digunakan 2 dioda saja sebagai penyearahnya.

2.2.3.2.2.1 Penyearah gelombang penuh dengan jembatan dioda

Pada dioda bridge, hanya ada 2 dioda saja yang menghantarkan arus untuk setiap siklus tegangan AC sedangkan 2 dioda lainnya bersifat sebagai isolator pada saat siklus yang sama.

Penyearah gelombang penuh dengan jembatan dioda ditunjukkan pada Gambar 2.15.



Gambar 2.20 Catu daya penyearah gelombang penuh dengan jembatan dioda.

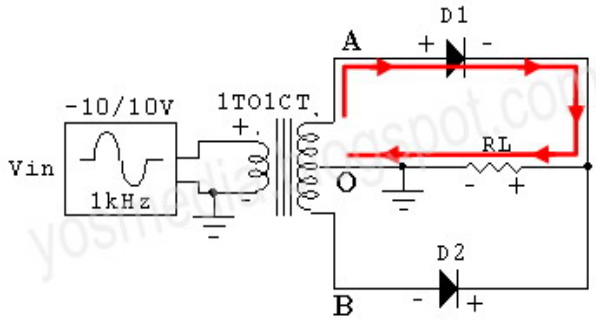
Berdasarkan Gambar 2.20 di atas, nilai V_R dapat diketahui menggunakan persamaan sebagai berikut:

$$V_R = \frac{2V_{S-maks}}{\pi} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (2.8)$$

2.2.3.2.2.2 Penyearah Gelombang penuh menggunakan 2 dioda

penyearah gelombang penuh menggunakan 2 dioda ini hanya bisa

digunakan pada transformator CT. Penyearah gelombang penuh menggunakan 2 dioda ditunjukkan pada Gambar 2.21

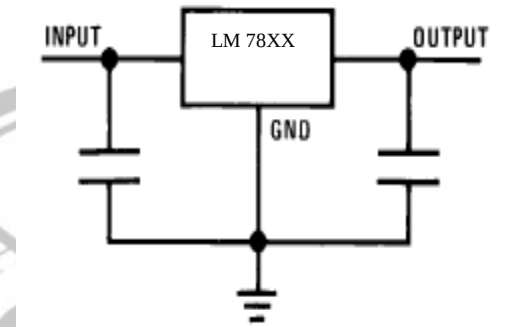


Gambar 2.16 Catu daya penyearah gelombang penuh dengan 2 dioda.

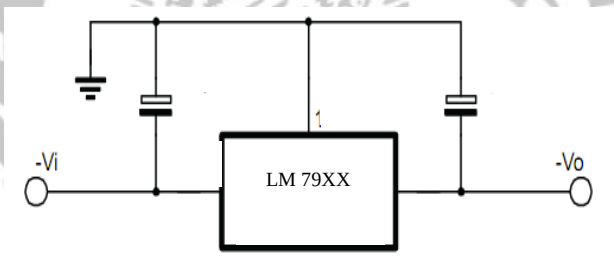
Pada trafo diketahui bahwa pada bagian sekunder trafo CT terdapat 2 sinyal output yang terjadi secara bersamaan, mempunyai amplitudo yang sama namun berlawanan fasa. Saat tegangan input (tegangan primer) berada pada siklus positif, pada titik AO akan terjadi siklus positif sementara pada titik OB akan terjadi siklus negatif. Akibatnya D1 akan mengalami panjaran maju (forward bias) sedangkan D2 mengalami panjaran balik (reverse bias) sehingga arus akan mengalir melalui D1 menuju ke beban dan kembali ke titik center tap.

2.2.3.2.3 Regulator Tegangan 3 Terminal bentuk IC

Salah satu metode agar dapat menghasilkan tegangan output DC stabil adalah dengan menggunakan IC 78XX untuk tegangan positif dan IC 79XX untuk tegangan negatif dalam sistem Regulator Tegangan. Tipe positif untuk meregulasi tegangan positif ditunjukkan pada Gambar 2.17 dan Tipe negatif untuk meregulasi tegangan negative ditunjukkan pada Gambar 2.18



Gambar 2.17 Regulator tegangan 3-terminal positif



Gambar 2.18 Regulator tegangan 3-terminal negatif

2.3 .4 SDCC

2.2.4.1 Pengenalan (*Identifier*)

Pengenalan adalah suatu nama yang digunakan untuk menyatakan variabel, kostanta bernama, tipe data, fungsi dan label. Suatu pengenalan merupakan satu atau beberapa karakter antara lain: huruf, angka dan garis bawah (_), yang harus dimulai dengan huruf atau garis bawah, contohnya adalah

tunda_pendek, _pencacah. Huruf kecil dengan huruf besar (kapital) pada suatu pengenalan adalah berbeda (*case sensitive*).

2.2 .4.2 Tipe Data

SDCC mendukung beberapa tipe data seperti pada Tabel 2.3.

Tabel.2.3 Tipe data SDCC

Tipe Data	Lebar Data	Jangkauan
Char (character)	8 bit (1 byte)	-128 s/d 127
Int (integer)	16 bit (2 byte)	-32768 s/d 32767
Short (short integer)	16 bit (2 byte)	-32768 s/d 32767
Long (long integer)	32 bit (4 byte)	-2.147.438.648 s/d 2.147.438.648
Float (floating point)	32 bit (4 byte)	$3.4.10^{-38}$ s/d $3.4.10^{+38}$
Void	tidak bertipe	-

SDCC tidak mendukung tipe data ***double.*** Tipe data dapat ditambahkan awalan ***signed*** untuk bilangan bertanda dan ***unsigned*** untuk bilangan tak bertanda. Awalan ini hanya dapat ditambahkan pada tipe data yang berhubungan dengan bilangan bulat yaitu ***char, int, short,***dan ***long***. Bilangan negatif dinyatakan dalam bentuk komplemen ke-2.

2.2.4.3 Bentuk Umum Program C

Untuk memberikan keterangan atau penjelasan program dapat dilakukan dengan dua cara, yaitu dengan tanda // atau /* ….*/. Tanda // digunakan untuk keterangan satu baris dan tanda /*…*/ untuk beberapa baris.

Contoh : // keterangan

*/ keterangan lebih dari dua baris */

2.2.4.4 Pengarah Preprosesor (#include)

#include <at89x51.h> merupakan pengarah preprosesor C (dikerjakan oleh sdcpp) yang memerintahkan untuk menyisisipkan *file* lain, dalam hal ini adalah *file* at89x51.h yang berisi deklarasi register dari mikrokontroler AT89S51. Secara lengkap *file header* untuk mikrokontroler buatan ATMEL seperti pada Tabel 2.4.

Tabel 2.4 File header untuk mikrokontroler ATMEL

Versi Mikrokontroler	File Header
AT89S51	at89x51.h
AT89S52	at89x52.h
AT891051/ AT892051/ AT894051	at89x051.h

Pengarah preprosesor lain yang sering digunakan adalah #define. Pengarah *preprosesor* #define digunakan untuk membuat konstanta bersimbol, alias, atau *macro.*

2.2.4.5 Fungsi Main( )

Program C minimal harus mempunyai satu fungsi, yaitu fungsi ***main*( )**. Void di depan main menandakan bahwa fungsi ***main*( )** tidak mempunyai nilai balik (*return value*). ***void*** didalam tanda kurung setelah kata main menandakan bahwa fungsi tidak mempunyai argumen atau parameter. Fungsi yang tidak mempunyai argumen, kata kunci ***void*** dapat dihilangkan, tetapi fungsi ***main*( )** tidak mempunyai nilai balik maka kata ***void*** tidak boleh dihilangkan.

2.2.4.6 Operasi Aritmatika

Sebagaimana kompiler bahasa C yang lainnya SDCC juga mendukung operator aritmetika, seperti pada Tabel 2.5.

Tabel 2.5 Operator aritmatika yang digunakan dalam bahasa C

No	Simbol	Keterangan
1	+	Operasi Penjumlahan
2	-	Operasi Pengurangan
3	*	Operasi Perkalian
3	/	Operasi Pembagian
5	%	Operasi Persen / sisa pembagian

2.2.4.7 Operator Penugasan (=)

Operator (=) di dalam bahasa C disebut juga operator tugas. Untuk menggunakan operator tugas adalah *left-hand-operand = right-hand-operand;*

Di sini operator penugasan menyebabkan nilai dari *riht-hand-operand* untuk diperintahkan penempatan dari memori *left-hand-operand*. Nilai sebelum operator penugasan adalah sama dengan nilai yang diperintahkan kepada *left-hand-operand*. Sebagai contoh a = 5; nilai sesudah operator penugasan (5) ke dalam penempatan memori dari bilangan bulat variabel a .

2.2.4.8 Operator *Bitwise* (manipulasi per bit)

Operasi *bitwise* banyak digunakan dalam aplikasi mikrokontroler, misalnya untuk menguji kondisi bit pada *port* 1 atau pergeseran bit .

Tabel 2.6 Operator manipulasi bit

No	Operator	Keterangan
1	&	Operator *bitwise* DAN/ AND
2	|	Operator *bitwise* ATAU/OR
3	^	Operator XOR
4	~	Operator *bitwise* NOT/komplemen
5	>>	Operator geser kanan
6	<<	Operator geser kiri

2.2.4.9 Operator Majemuk

Operator majemuk terdiri atas dua operator yang digunakan untuk menyingkat tulisan. Operator majemuk seperti pada Tabel 2.7.

Tabel 2.7 Operator majemuk

No	Operator	Contoh	Kependekan Dari
1	+=	*Conter* += 1;	*Conter* = *Conter* +1;
2	-=	*Conter* -= 1;	*Conter* = *Conter* -1;
3	*=	*Conter* *= 1;	*Conter* = *Conter* *1;
4	/=	*Conter* /= 1;	*Conter* = *Conter* /1;
5	%=	*Conter* %= 1;	*Conter* = *Conter* %1;
6	<<=	*Conter* <<=1;	*Conter* = *Conter* <<1;
7	>>=	*Conter* >>=1;	*Conter* = *Conter* >>1;
8	&=	*Conter* &= 1;	*Conter* = *Conter* &1;
9	|=	*Conter* |= 1;	*Conter* = *Conter* |1;
10	^=	*Conter* ^= 1;	*Conter* = *Conter* ^1;
11	~=	*Conter* ~= 1;	*Conter* = ~ *Conter*

2.2.4.10 Operator Logik

Operator logika digunakan untuk menghubungkan dua buah ekspresi kondisi menjadi satu buah ekspresi kondisi. Ekspresi kondisi biasanya bernilai benar atau salah. Ada tiga buah operator logika seperti pada Tabel 2.7.

Tabel 2.8 Operator logika

Operator logika	Makna	Keterangan
&&	DAN/AND	Mengembalikan benar hanya jika kedua sisi benar && bernilai benar.
||	ATAU/ OR	Mengembalikan benar jika ada sisi pada || yang bernilai benar (salah satu sisi atau kedua sisi benar)
!	TIDAK/NOT	Mengubah relasi benar kesalah atau relasi salah ke benar.

Operator AND akan bernilai benar jika dan hanya jika dua ekspresi bernilai benar. Operasi OR akan bernilai benar jika dan hanya jika salah satu ekspresi bernilai benar. Sedangkan operasi NOT menghasilkan nilai benar jika ekspresinya bernilai salah dan bernilai salah jika ekspresi bernilai benar. Secara keseluruhan tabel kebenaran operasi logika seperti pada Tabel 2.8 dan Tabel 2.9.

Tabel 2.9 Tabel kebenaran logika AND, OR

Ekspresi 1	Ekepresi 2	Operasi &&	Operasi ||
Salah	Salah	Salah	Salah
Salah	Benar	Salah	Benar
Benar	Salah	Salah	Benar
Benar	Benar	Benar	Benar

Tabel 2.10 Tabel kebenaran logika NOT

Ekspresi	Operasi
Salah	Benar
Benar	Salah

2.2.4.11 Operator Relasi

Operator relasi merupakan hubungan antara dua ungkapan yang biasa digunakan untuk membandingkan dua buah nilai atau kondisi.

Tabel 2.11 Operator relasi

Operator	Contoh	Keterangan
==	a == b	Bernilai **benar** jika a dan b mempunyai nilai yang sama dan bernilai **salah** jika a dan b bernilai berbeda.
!=	a != b	Bernilai **benar** jika a dan b mempunyai nilai yang tidak sama dan bernilai **salah** jika a dan b bernilai sama.
>	a > b	Bernilai **benar** jika a bernilai lebih dari b dan bernilai **salah** jika a bernilai kurang dari atau sama dengan b.
<	a < b	Bernilai **benar** jika a bernilai kurang dari b dan bernilai **salah** jika a bernilai lebih dari atau sama dengan b.
>=	a >= b	Bernilai **benar** jika a bernilai lebih dari atau sama dengan b dan bernilai **salah** jika a bernilai kurang dari b.

Tabel 2.11 (lanjutan)

<=	a <= b	Bernilai **benar** jika a bernilai kurang dari atau sama dengan b dan bernilai **salah** jika a bernilai lebih dari b.

2.2.4.12 Operator Penurunan dan Penaikan

Bahasa C mempunyai operator untuk penurunan (*decrement*) dan penaikan (*increment*) nilai variabel. Operator penaikan dan penurunan seperti pada Tabel 2.15.

Format penulisannya :

Operator nama_variable atau nama_variable operator.

Contoh : **++counter atau counter ++**

--hasil atau hasil--

penulisan operator ++ atau – sebelum atau sesudah memberikan hasil akhir yang sama namun dalam beberapa pernyataan dapat menghasilkan perbedaan.

Tabel 2.12 Operator penurunan dan penaikan

No	Operator	Keterangan
1	++	Penaikan
2	- -	Penurunan

2.2.4.13 Pernyataan Kondisional (*if*)

Setiap bahasa pemrograman mempunyai beberapa bentuk pernyataan *if*. Pernyataan *if* menguji operator relasi dan memutuskan bagian program mana yang akan dijalankan dan yang diabaikan. Pernyataan *if* yang menentukan program harus memutar atau berjalan langsung. Pernyataan *if* juga membiarkan program anda mengambil keputusan pada saat berjalan berdasarkan nilai data.

2.2.4.14 Pernyataan pengulangan (*for*)

Pada saat bahasa C (SDCC) menjumpai pernyataan *for* akan mengikuti langkah-langkah berikut untuk melaksanakan kalang;

a. Melaksanakan ekspresi awal yang biasanya hanya berupa pernyataan penugasan.
b. Menguji ekspresi kondisi dengan hasil salah atau benar.
c. Melaksanakan tubuh kalang jika kondisi benar.
d. Melaksanakan ekspresi pencacah, yang biasanya berupa operasi penaikan atau penurunan.
e. Kembali ke langkah 2 .

Sewaktu kondisi diuji dan didapatkan bernilai salah bahasa C (SDCC) menghentikan kalang dan program berlanjut ke pernyataan sesudah kalang *for*. Titik koma selalu mengakhiri pernyataan yang tereksekusi dan ekspresi awal, dan

kondisi adalah pernyataan tersendiri di dalam *for*. Dua tanda titik koma pada *for* membantu memisahkan ketiga pernyataan yang terdapat di dalam tanda kurung. Bentuk perulangan dengan *for*, ekspresi1 merupakan bentuk inisialisasi awal, sedangkan ekspresi2 merupakan kondisi dan ekspresi3 adalah pernyataan jika kondisi ekspresi 2 bernilai benar. Sebagai contoh pemakaian *for*

```
1. void tunda(int j)
2. {
3.     int i;
4.     for (i = 0; i < j; i++) ;
5. }
```

2.2.4.15 Pernyataan *While*

Pada pemograman bahasa C instruksi *while* digunakan untuk memanggil program berulang-ulang kali sehingga program menjadi sangat panjang. Disini kita tinggal memutuskan kode yang digunakan berulang-ulang dan menempatkannya di dalam sebuah kalang. Didalam kalang tersebut kita mungkin hanya menanyakan sebuah pernyataan dan menempatkan masukan baru berulang-ulang. Setiap berputar pada kalang bahasa C dapat menanyakan pernyataan. Saat mengulang kembali pernyataan, ketika nilai tidak terpenuhi maka kalang akan berhenti melakukan pengulangan.