

MIKROKONTROLLER AT89S51

Mikrokontroller 8 bit dengan 4K byte ISP
(In System Programmable)

(dari berbagai sumber)

DISKRIPSI

AT89S51 mempunyai konsumsi daya rendah, mikrokontroller 8-bit CMOS dengan 4K byte memori Flash ISP (in system programmable/ dapat diprogram didalam sistem).Divais ini dibuat dengan teknologi memori nonvolatile kerapatan tinggi dan kompatibel dengan standart industri 8051, set instruksi dan pin keluaran. Flash yang berada didalam chip memungkinkan memori program untuk diprogram ulang pada saat chip didalam sistem atau dengan menggunakan Programmer memori nonvolatile konvensional. Dengan mengkombinasikan CPU 8 bit yang serbaguna dengan flash ISP pada chip, AT89S51 merupakan mikrokontroller yang luarbiasa yang memberikan fleksibilitas yang tinggi dan penyelesaian biaya yang efektif untuk beberapa aplikasi kontrol.

AT89S51 memberikan fitur-fitur standar sebagai berikut: 4K byte Flash, 128 byte RAM, 32 jalur I/O, Timer Watchdog, dua data pointer, dua 16 bit timer/ counter, lima vektor interupsi dua level, sebuah port serial full duplex, oscilator internal, dan rangkaian clock. Selain itu AT89S51 didisain dengan logika statis untuk operasi dengan frekuensi sampai 0 Hz dan didukung dengan mode penghematan daya. Pada mode idle akan menghentikan CPU sementara RAM, timer/ counter, serial port dan sistem interupsi tetap berfungsi. Mode Power Down akan tetap menyimpan isi dari RAM tetapi akan membekukan osilator, menggagalkan semua fungsi chip sampai interupsi eksternal atau reset hardware ditemui.

DISKRIPSI PIN

VCC Tegangan Supply

GND Ground

Port 0

Port 0, merupakan port I/O 8 bit open drain dua arah. Sebagai sebuah port, setiap pin dapat mengendalikan 8 input TTL. Ketika logika "1" dituliskan ke port 0, maka port dapat digunakan sebagai input dengan high impedansi. Port 0 dapat juga dikonfigurasi untuk multipleksing dengan address/ data bus selama mengakses memori program atau data eksternal. Pada mode ini P0 harus mempunyai pull up

Port 1

Port 1 merupakan port I/O 8 bit dua arah dengan internal pull up. Buffer output

port 1 dapat mengendalikan empat TTL input. Ketika logika “1” dituliskan ke port 1, maka port ini akan mendapatkan internal pull up dan dapat digunakan sebagai input.

Port 1 juga menerima alamat byte rendah selama pemrograman dan verifikasi Flash

Port Pin Fungsi Alternatif

P1.5 MOSI (digunakan untu In System Programming)

P1.6 MISO (digunakan untu In System Programming)

P1.7 SCK (digunakan untu In System Programming)

Port 2

Port 2 merupakan port I/O 8 bit dua arah dengan internal pull up. Buffer output port 2 dapat mengendalikan empat TTL input. Ketika logika “1” dituliskan ke port 2, maka port ini akan mendapatkan internal pull up dan dapat digunakan sebagai input.

Port 3

Port 3 merupakan port I/O 8 bit dua arah dengan internal pull up. Buffer output port 3 dapat mengendalikan empat TTL input. Ketika logika “1” dituliskan ke port 3, maka port ini akan mendapatkan internal pull up dan dapat digunakan sebagai input.

Port 3 juga melayani berbagai macam fitur khusus, sebagaimana yang ditunjukkan pada tabel berikut:

Port Pin	Fungsi Alternatif
P3.0	RXD (port serial input)
P3.1	TXD (port serial output)
P3.2	INT0 (interupsi eksternal 0)
P3.3	INT1 (interupsi eksternal 1)
P3.4	T0 (input eksternal timer 0)
P3.5	T1 (input eksternal timer 1)
P3.6	WR (write strobe memori data eksternal)
P3.7	WR (read strobe memori program eksternal)

RST

Input Reset. Logika high “1” pada pin ini untuk dua siklus mesin sementara oscilator bekerja maka akan mereset devais.

ALE/ PROG

Address Latch Enale (ALE) merupakan suatu pulsa output untuk mengunci byte low dari alamat selama mengakses memori eksternal. Pin ini juga merupakan input pulsa pemrograman selama pemrograman flash (paralel)

Pada operasi normal, ALE mengeluarkan suatu laju konstan $1/6$ dari frekuensi oscilator dan dapat digunakan untuk pewaktu eksternal.

PSEN

Program Store Enable merupakan strobe read untuk memori program eksternal.

EA/ VPP

Eksternal Access Enable. EA harus di hubungkan ke GND untuk enable devais, untuk memasuki memori program eksternal mulai alamat 0000H s/d FFFFH.

EA harus di hubungkan ke VCC untuk akses memori program internal

Pin ini juga menerima tegangan pemrograman (VPP) selama pemrograman Flash

XTAL1

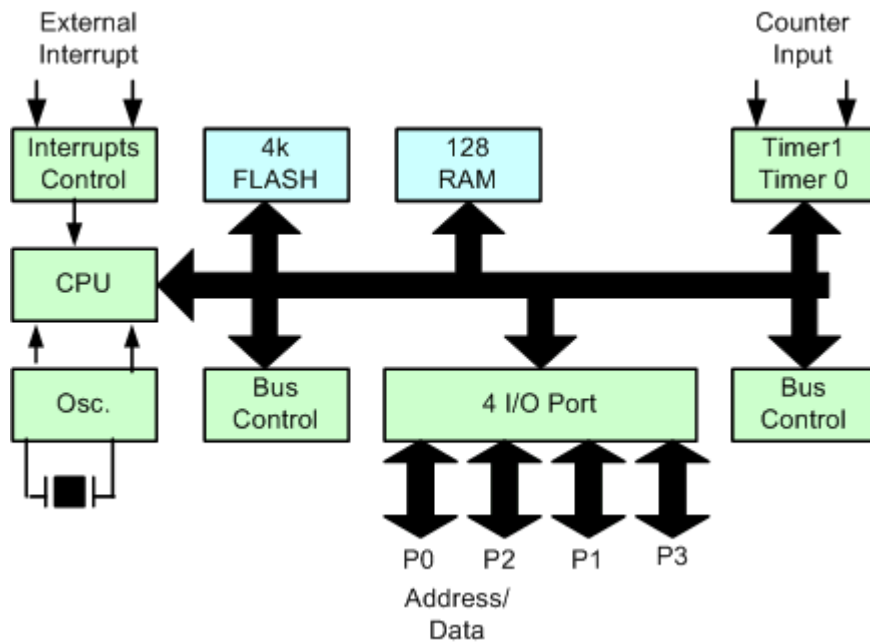
Input untuk penguat oscilator inverting dan input untuk rangkaian internal clock

XTAL2

Output dari penguat oscilator inverting.

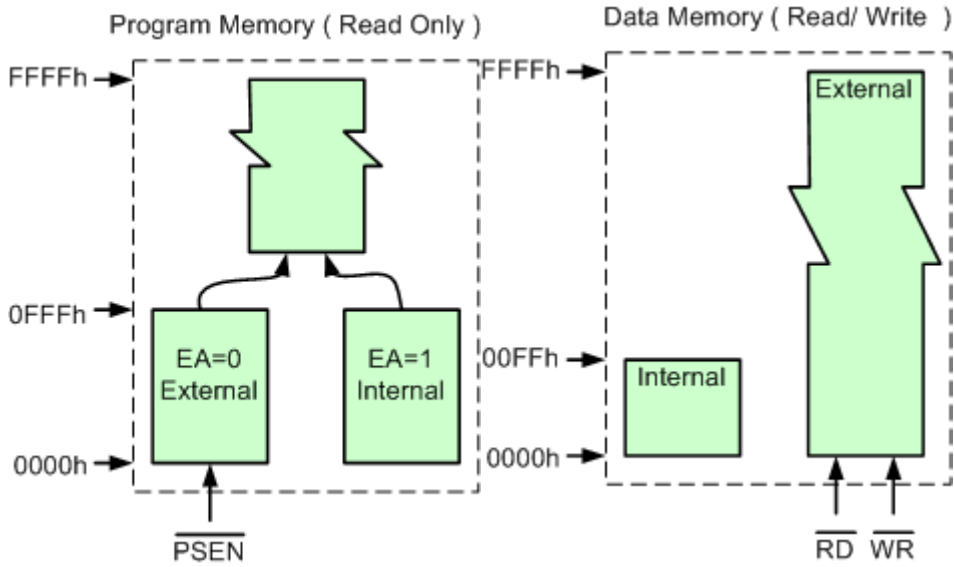
Organisasi Memori

Semua divais 8051 mempunyai ruang alamat yang terpisah untuk memori program dan memori data, seperti yang ditunjukkan pada gambar 1.1. dan gambar 1.2. Pemisahan secara logika dari memori program dan data, memungkinkan memori data untuk diakses dengan pengalamatan 8 bit, yang dengan cepat dapat disimpan dan dimanipulasi dengan CPU 8 bit. Selain itu, pengalamatan memori data 16 bit dapat juga dibangkitkan melalui register DPTR. Memori program (ROM, EPROM dan FLASH) hanya dapat dibaca, tidak ditulis. Memori program dapat mencapai sampai 64K byte. Pada 89S51, 4K byte memori program terdapat didalam chip. Untuk membaca memori program eksternal mikrokontroller mengirim sinyal PSEN (program store enable)



Gambar 1.1. Diagram blok mikrokontroller 8051

Memori data (RAM) menempati ruang alamat yang terpisah dari memori program. Pada keluarga 8051, 128 byte terendah dari memori data, berada didalam chip. RAM eksternal (maksimal 64K byte). Dalam pengaksesan RAM Eksternal, mikrokontroller mengirim sinyal RD (baca) dan WR (tulis).

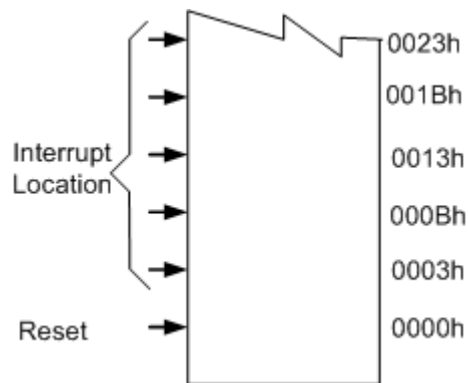


Gambar 1.2. Arsitektur Memori Mikrokontroller 8051

Program Memory

Gambar 1.2. menunjukkan suatu peta bagian bawah dari memori program. Setelah reset CPU mulai melakukan eksekusi dari lokasi 0000H. Sebagaimana yang ditunjukkan pada gambar 1.3, setiap interupsi ditempatkan pada suatu lokasi tertentu pada memori program. Interupsi menyebabkan CPU untuk melompat ke lokasi dimana harus dilakukan suatu layanan tertentu.

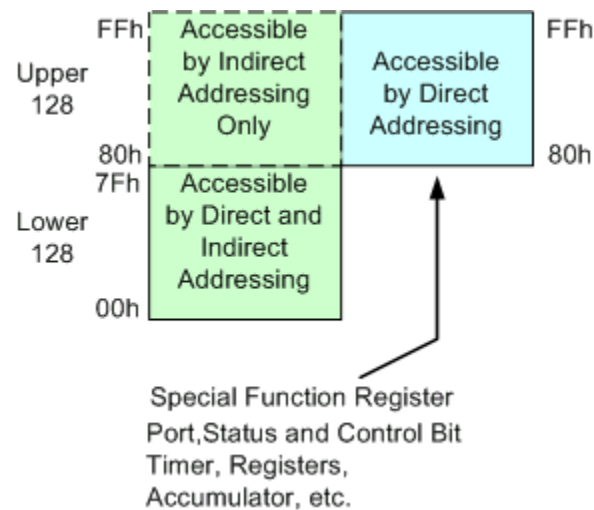
Interupsi Eksternal 0, sebagai contoh, menempatai lokasi 0003H. Jika Interupsi Eksternal 0 akan digunakan, maka layanan rutin harus dimulai pada lokasi 0003H. Jika interupsi ini tidak digunakan, lokasi layanan ini dapat digunakan untuk berbagai keperluan sebagai Memori Program.



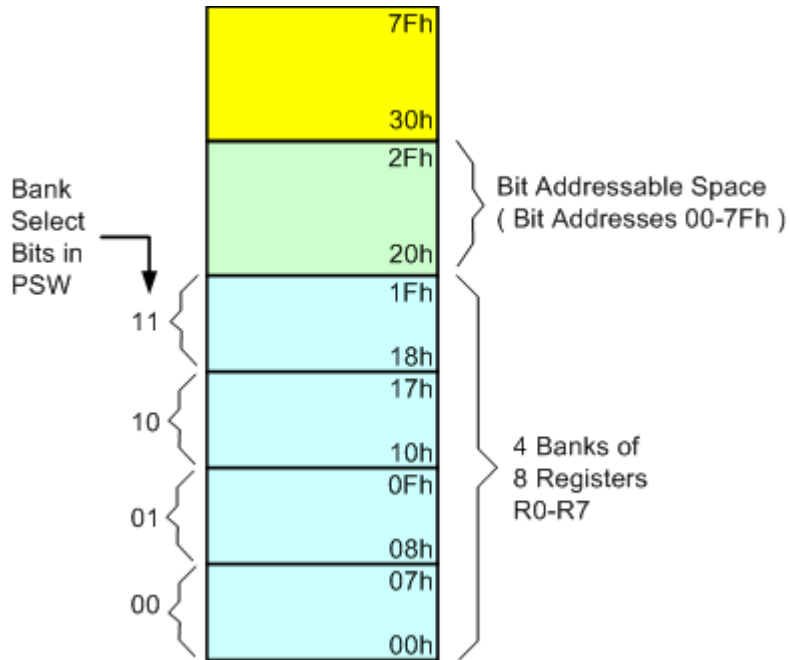
Gambar 1.3. Peta Interupsi mikrokontroller 8051

Memory Data

Pada gambar 1.2. menunjukkan ruang memori data internal dan eksternal pada keluarga 8051. CPU membangkitkan sinyal RD dan WR yang diperlukan selama akses RAM eksternal. Memori data internal terpetakan seperti pada gambar 1.2. Ruang memori dibagi menjadi tiga blok, yang diacukan sebagai 128 byte lower, 128 byte upper dan ruang SFR. Alamat memori data internal selalu mempunyai lebar data satu byte. Pengalamatan langsung diatas 7Fh akan mengakses satu alamat memori, dan pengalamatan tak langsung diatas 7Fh akan mengakses satu alamat yang berbeda. Demikianlah pada gambar 1.4 menunjukkan 128 byte bagian atas dan ruang SFR menempati blok alamat yang sama, yaitu 80h sampai dengan FFh, yang sebenarnya mereka terpisah secara fisik 128 byte RAM bagian bawah dikelompokkan lagi menjadi beberapa blok, seperti yang ditunjukkan pada gambar 1.5. 32 byte RAM paling bawah, dikelompokkan menjadi 4 bank yang masing-masing terdiri dari 8 register. Instruksi program untuk memanggil register-register ini dinamai sebagai R0 sampai dengan R7. Dua bit pada Program Status Word (PSW) dapat memilih register bank mana yang akan digunakan. Penggunaan register R0 sampai dengan R7 ini akan membuat pemrograman lebih efisien dan singkat, bila dibandingkan pengalamatan secara langsung.

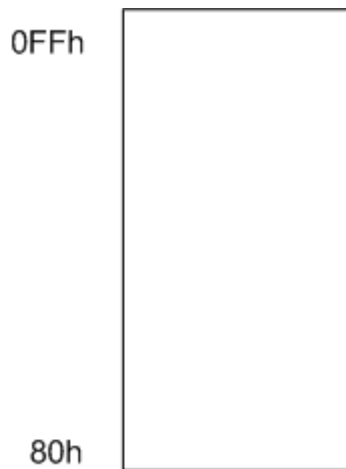


Gambar 1.4. Memori data internal



Gambar 1.5. RAM internal 128 byte paling bawah

Semua pada lokasi RAM 128 byte paling bawah dapat diakses baik dengan menggunakan pengalamatan langsung dan tak langsung. 128 byte paling atas hanya dapat diakses dengan cara tak langsung, gambar 1.6.



Gambar 1.6. RAM internal 128 byte paling atas

Special Function Register

Sebuah peta memori yang disebut ruang special function register (SFR) ditunjukkan pada gambar berikut. Perhatikan bahwa tidak semua alamat-alamat tersebut ditempati, dan alamat-alamat yang tak ditempati tidak diperkenankan untuk diimplementasikan. Akses baca untuk alamat ini akan menghasilkan data random, dan akses tulis akan menghasilkan efek yang tak jelas.

Accumulator

ACC adalah register akumulator. Mnemonik untuk instruksi spesifik akumulator ini secara sederhana dapat disingkat sebagai A.

Register B

Register B digunakan pada saat operasi perkalian dan pembagian. Selain untuk keperluan tersebut diatas, register ini dapat digunakan untuk register bebas.

Program Status Word

Register PSW terdiri dari informasi status dari program yang secara detail ditunjukkan pada Tabel 1.1.

Stack Pointer

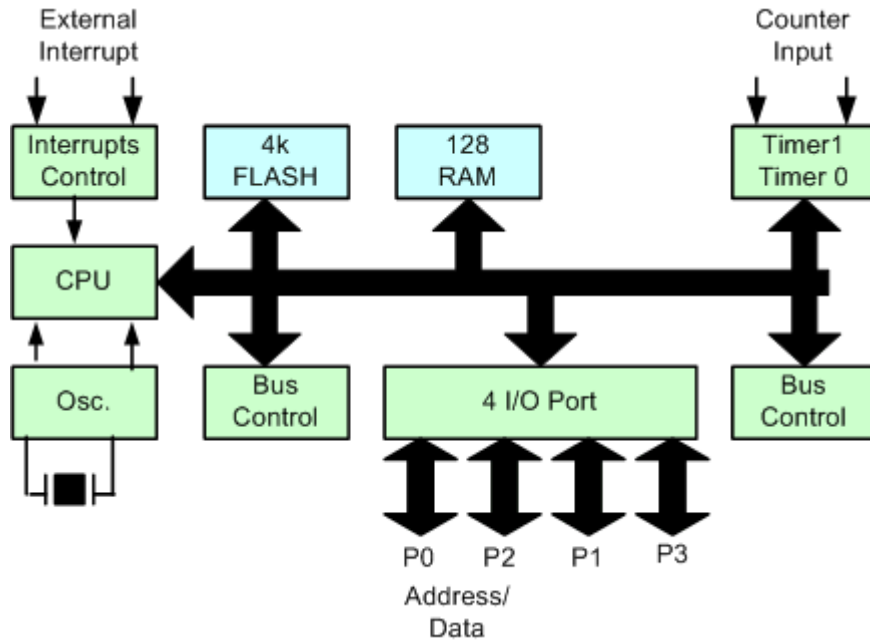
Register Pointer stack mempunyai lebar data 8 bit. Register ini akan bertambah sebelum data disimpan selama eksekusi push dan call. Sementara stack dapat berada disembarang tempat RAM. Pointer stack diawali di alamat 07h setelah reset. Hal ini menyebabkan stack untuk memulai pada lokasi 08h.

Data Pointer

Pointer Data (DPTR) terdiri dari byte atas (DPH) dan byte bawah (DPL). Fungsi ini ditujukan untuk menyimpan data 16 bit. Dapat dimanipulasi sebagai register 16 bit atau dua 8 bit register yang berdiri sendiri.

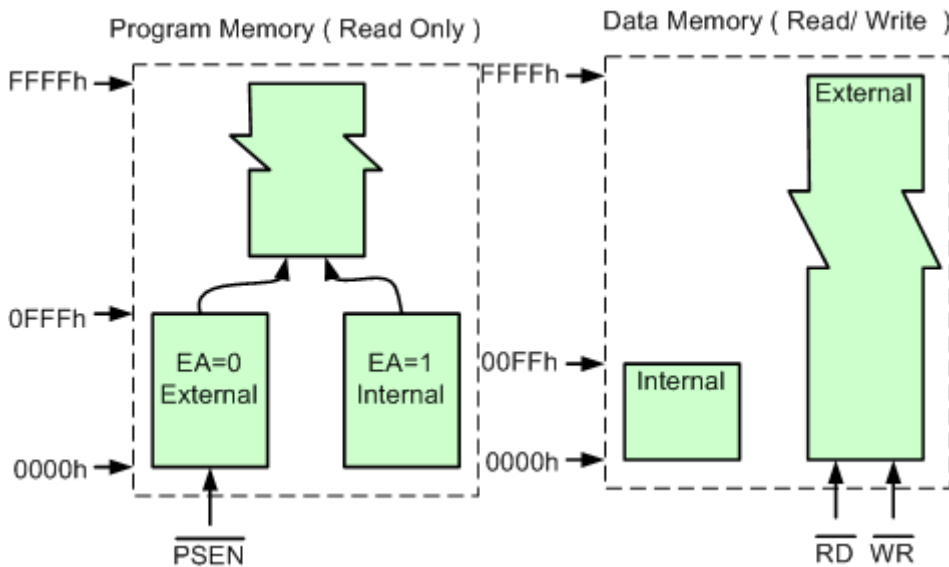
Organisasi Memori

Semua divais 8051 mempunyai ruang alamat yang terpisah untuk memori program dan memori data, seperti yang ditunjukkan pada gambar 1.1. dan gambar 1.2. Pemisahan secara logika dari memori program dan data, memungkinkan memori data untuk diakses dengan pengalamatan 8 bit, yang dengan cepat dapat disimpan dan dimanipulasi dengan CPU 8 bit. Selain itu, pengalamatan memori data 16 bit dapat juga dibangkitkan melalui register DPTR. Memori program (ROM, EPROM dan FLASH) hanya dapat dibaca, tidak ditulis. Memori program dapat mencapai sampai 64K byte. Pada 89S51, 4K byte memori program terdapat didalam chip. Untuk membaca memori program eksternal mikrokontroller mengirim sinyal PSEN (program store enable)



Gambar 1.1. Diagram blok mikrokontroler 8051

Memori data (RAM) menempati ruang alamat yang terpisah dari memori program. Pada keluarga 8051, 128 byte terendah dari memori data, berada didalam chip. RAM eksternal (maksimal 64K byte). Dalam pengaksesan RAM Eksternal, mikrokontroler mengirimkan sinyal RD (baca) dan WR (tulis).



Gambar 1.2. Arsitektur Memori Mikrokontroler 8051

Pengalamatan

Mode pengalamatan, mengacu bagaimana anda mengalami suatu lokasi memori tertentu Mode pengalamatan pada set instruksi 8051 adalah ditunjukkan sebagai berikut

Immediate Addressing	MOV A,#20h
Direct Addressing	MOV A,30h
Indirect Addressing	MOV A,@R0
External Direct	MOVX A,@DPTR
Code Indirect	MOVC A,@A+DPTR

1.2.1 Immediate Addressing

Immediate addressing dinamakan seperti ini, karena nilai yang akan disimpan didalam memori, secara langsung berada dalam kode.

```
        org 0h
start:MOV A,#20h; put constant 20 into Acc
        end
Org 0h
Start:MOV 70h,#0h; put constant 0 into RAM 70h
        MOV 71h,#1h;
        MOV 72h,#2h;
        end
;
        Org 0h
Start:MOV DPTR,#1234h;put constant 1234 into DPTR
        end
        Org 0h
Start:MOV PSW,#0; Select register bank 0
        MOV R0,#0; put 0 into register 0
        MOV R1,#1; put 1 into register 1
        MOV R2,#2; put 2 into register 2
        MOV R3,#3; put 3 into register 3
        MOV R4,#4; put 4 into register 4
        MOV R5,#5; put 5 into register 5
        MOV R6,#6; put 6 into register 6
        MOV R7,#7; put 7 into register 7
        end
;
org 0h
Start:MOV PSW,#8; Select register bank 1
        MOV R0,#0; put 0 into register 0
        MOV R1,#1; put 1 into register 1
        MOV R2,#2; put 2 into register 2
        MOV R3,#3; put 3 into register 3
        MOV R4,#4; put 4 into register 4
        MOV R5,#5; put 5 into register 5
        MOV R6,#6; put 6 into register 6
        MOV R7,#7; put 7 into register 7
        end
```

Immediate addressing adalah pengalamatan yang sangat cepat karena nilai yang akan diloadkan berada didalam instruksi tersebut.

1.2.2. Direct Addressing

Disebut direct addressing karena nilai yang akan disimpan didalam memori, diperoleh secara langsung dari memori yang lain.

```
    org 0h
Start:MOV A,30h; coment
    end
    Org 0h
Start: Mov 70h,#1;put constant 1 into RAM 70h
    Mov A, 70h;copy RAM 70 content into Acc
    Mov A,#0 ;put constant 0 into Acc
    Mov 90h,A ;copy Acc content into RAM 90h
    end
Inbyte equ 70h
Port1 equ 90h
    Org 0h
Start: Mov Inbyte,#3;put constant 3 into RAM 70h
    Mov A,Inbyte ;copy RAM 70h content into Acc
    Mov A,#0 ;Clear accumulator
    Mov Port1,A ;copy Acc content into RAM 90h
    end
Org 0h
Mov DPTR,#Character
Start: Mov A, #0
    Inc DPTR
    Movc A, @A+DPTR
    Mov R0,A
    Sjmp Start
Character:
    DB 0,1,2,3,4,5,6,7,8,9
```

1.2.3 Indirect Addressing

Indirect addressing adalah mode pengalamatan yang sangat ampuh, yang memberikan fleksibilitas dalam hal transfer data. Indirect addressing juga satu-satunya cara untuk mengakses 128 byte ekstra dari internal RAM yang ditemukan pada keluarga 8052.

MOV A,@R0

Instruksi ini menyebabkan 8051 menganalisa nilai dari register R0. 8051 kemudian akan mengambil data dari akumulator dengan nilai yang berasal dari alamat RAM internal yang ditunjukkan oleh R0. Sebagai contoh, misal R0 akan digunakan untuk menandai alamat RAM 40h yang berisi data 67h. Ketika instruksi diatas, dieksekusi maka 8051 akan melihat nilai dari R0, yang berisi 40h, dan mengirimkan isi RAM 40h (dalam hal ini mempunyai nilai 67h) ke akumulator.

```
MOV R0,#99h ;
MOV @R0,#01h;
```

Instruksi tersebut adalah tidak valid. Karena indirect addressing selalu mengacu ke RAM internal, dua instruksi ini akan menulis nilai 01 ke RAM internal alamat 99h pada 8052.

Pada 8051 instruksi tersebut akan menghasilkan hasil yang tak terdefinisi, karena 8051 hanya mempunyai internal RAM 128 byte

```
Org 0h
```

```
Start: Mov PSW, #0 ; choose register bank 0
      Mov R0, #78h; put constant 78h into R0
      Mov @R0, #1 ; put constant 1 into 78h
      end
```

```
Org 0h
```

```
Start: Mov PSW, #0; pilih register bank 1
      Mov R0, 90h; copy RAM 90h content into R0
      Mov @R0, #1; put constant 1 into 90h
      End
      ;
```

1.3. Set Instruksi

Program pengendali mikrokontroler disusun dari kumpulan instruksi, instruksi tersebut setara dengan kalimat perintah bahasa manusia yang hanya terdiri atas predikat dan objek. Dengan demikian tahap pertama pembuatan program pengendali mikrokontroler dimulai dengan pengenalan dan pemahaman predikat (kata kerja) dan objek apa saja yang dimiliki mikrokontroler.

Objek dalam pemrograman mikrokontroler adalah data yang tersimpan di dalam memori, register dan input/output. Sedangkan 'kata kerja' yang dikenal pun secara umum dikelompokkan menjadi perintah untuk perpindahan data, aritmetik, operasi logika, pengaturan alur program dan beberapa hal khusus. Kombinasi dari 'kata kerja' dan objek itulah yang membentuk perintah pengatur kerja mikrokontroler.

Intruksi MOV A,\$7F merupakan contoh sebuah intruksi dasar yang sangat spesifik, MOV merupakan 'kata kerja' yang memerintahkan peng-copy-an data, merupakan predikat dalam kalimat perintah ini. Sedangkan objeknya adalah data yang di-copy-kan, dalam hal ini adalah data yang ada di dalam memori nomor \$7F di-copy-kan ke Akumulator A.

Penyebutan data dalam MCS51

Data bisa berada diberbagai tempat yang berlainan, dengan demikian dikenal beberapa cara untuk menyebut data (dalam bahasa Inggris sering disebut sebagai 'Addressing Mode'), antara lain sebagai berikut.

1. Penyebutan data konstan (immediate addressing mode): MOV A,#\$20. Data konstan merupakan data yang berada di dalam instruksi. Contoh instruksi ini mempunyai makna data konstan \$20 (sebagai data konstan ditandai dengan '#') di-copy-kan ke Akumulator A. Yang perlu benar-benar diperhatikan dalam perintah ini adalah bilangan \$20 merupakan bagian dari instruksi.
2. Penyebutan data secara langsung (direct addressing mode), cara ini dipakai untuk menunjuk data yang berada di dalam memori dengan cara menyebut nomor memori tempat data tersebut berada : MOV A,\$30. Contoh instruksi ini mempunyai makna data yang berada di dalam memori nomor \$30 di-copy-kan ke Akumulator. Sekilas intruksi ini sama dengan instruksi data konstan di atas, perbedaannya instruksi di atas memakai tanda '#' yang menandai \$20 adalah data konstan, sedangkan dalam instruksi ini karena tidak ada tanda '#' maka \$30 adalah nomor dari memori.
3. Penyebutan data secara tidak langsung (indirect addressing mode), cara ini dipakai untuk menunjuk data yang berada di dalam memori, kalau memori penyimpanan data ini letaknya berubah-ubah sehingga nomor memori tidak disebut secara langsung tapi di-'titip'-kan ke register lain : MOV A,@R0.
4. Dalam instruksi ini register serba guna R0 dipakai untuk mencatat nomor memori, sehingga instruksi ini mempunyai makna memori yang nomornya tercatat dalam R0 isinya di-copy-kan ke Akumulator A.
5. Tanda '@' dipakai untuk menandai nomor memori disimpan di dalam R0.
6. Bandingkan dengan instruksi penyebutan nomor memori secara langsung di atas, dalam instruksi ini nomor memori terlebih dulu disimpan di R0 dan R0 berperan

menunjuk memori mana yang dipakai, sehingga kalau nilai R0 berubah memori yang ditunjuk juga akan berubah pula.

7. Dalam instruksi ini register serba guna R0 berfungsi dengan register penampung alamat (indirect address register), selain R0 register serba guna R1 juga bisa dipakai sebagai register penampung alamat.

8. Penyebutan data dalam register (register addressing mode): MOV A,R5. Instruksi ini mempunyai makna data dalam register serba guna R5 di-copy-kan ke Akumulator A. Instruksi ini membuat register serba guna R0 sampai R7 sebagai tempat penyimpanan data yang sangat praktis yang kerjanya sangat cepat.

9. Data yang dimaksud dalam bahasan di atas semuanya berada di dalam memori data (termasuk register serba guna letaknya juga di dalam memori data). Dalam penulisan program, sering-sering diperlukan tabel baku yang disimpan bersama dengan program tersebut. Tabel semacam ini sesungguhnya merupakan data yang berada di dalam memori program!

10. Untuk keperluan ini, MCS51 mempunyai cara penyebutan data dalam memori program yang dilakukan secara indirect (code indirect addressing mode) : MOVC A,@A+DPTR.

Perhatikan dalam instruksi ini MOV digantikan dengan MOVC, tambahan huruf C tersebut dimaksud untuk membedakan bahwa instruksi ini dipakai di memori program. (MOV tanpa huruf C artinya instruksi dipakai di memori data).

Tanda '@' dipakai untuk menandai A+DPTR dipakai untuk menyatakan nomor memori yang isinya di-copy-kan ke Akumulator A, dalam hal ini nilai yang tersimpan dalam DPTR (Data Pointer Register - 2 byte) ditambah dengan nilai yang tersimpan dalam Akumulator A (1 byte) dipakai untuk menunjuk nomor memori program.

Secara keseluruhan AT8951 mempunyai sebanyak 255 macam instruksi, yang dibentuk dengan mengkombinasikan 'kata kerja' dan objek. "Kata kerja" tersebut secara kelompok dibahas sebagai berikut :

1.3.1 Instruksi copy data

Kode dasar untuk kelompok ini adalah MOV, singkatan dari MOVE yang artinya memindahkan, meskipun demikian lebih tepat dikatakan perintah ini mempunyai makna peng-copy-an data. Hal ini bisa dijelaskan berikut : setelah instruksi MOV A,R7 dikerjakan, Akumulator A dan register serba guna R7 berisikan data yang sama, yang asalnya tersimpan di dalam R7.

Perintah MOV dibedakan sesuai dengan jenis memori AT89Cx051. Perintah ini pada memori data dituliskan menjadi MOV, misalkan :

```
MOV A,$20
MOV A,@R1
MOV A,P1
MOV P3,A
```

Untuk pemakaian pada memori program, perintah ini dituliskan menjadi MOVC, hanya ada 2 jenis instruksi yang memakai MOVC, yakni:

MOVC A,@A+DPTR ; DPTR sebagai register indirect
 MOVC A,@A+PC ; PC sebagai register indirect

Selain itu, masih dikenal pula perintah MOVX, yakni perintah yang dipakai untuk memori data eksternal (X singkatan dari External). Perintah ini hanya dimiliki oleh anggota keluarga MCS51 yang mempunyai memori data eksternal, misalnya AT89C51 dan lain sebagainya, dan jelas tidak dikenal oleh kelompok AT89Cx051 yang tidak punya memori data eksternal. Hanya ada 6 macam instruksi yang memakai MOVX, instruksi-instruksi tersebut adalah:

MOVX A,@DPTR
 MOVX A,@R0
 MOVX A,@R1
 MOVX @DPTR,A
 MOVX @R0,A
 MOVX @R1,A

Mnemonic	Operation	Addressing Mode				Exect.
		Dir	Ind	Reg	Imm	Timer uS
Mov A,<src>	A=<src>	V	V	V	V	1
Mov <dest>,A	<dest>=A	V	V	V	V	1
Mov <dest>,<src>	<dest>=<src>	V	V	V	V	1
Mov DPTR,#data16	DPTR=16 bit immediate const	Accumulator Only				1
Push <src>	Inc SP	V	V	V		1
Pop <src>	Dec SP	Data Pointer Only				2
Xch A,<byte>	Acc and <byte> exchange data	Accumulator Only				1
Xchd A,@Ri	Acc and @Ri exchange low nibbles	V	V	V		1

1.3.2 Instruksi Aritmatika

Perintah ADD dan ADDC

Isi Akumulator A ditambah dengan bilangan 1 byte, hasil penjumlahan akan ditampung kembali dalam Akumulator. Dalam operasi ini bit Carry (C flag dalam PSW – Program Status Word) berfungsi sebagai penampung limpahan hasil penjumlahan. Jika hasil penjumlahan tersebut melimpah (nilainya lebih besar dari 255) bit Carry akan bernilai '1', kalau tidak bit Carry bernilai '0'. ADDC sama dengan ADD, hanya saja dalam ADDC nilai bit Carry dalam proses sebelumnya ikut dijumlahkan bersama.

Bilangan 1 byte yang ditambahkan ke Akumulator, bisa berasal dari bilangan konstan, dari register serba guna, dari memori data yang nomor memorinya disebut secara langsung maupun tidak langsung, seperti terlihat dalam contoh berikut :

```
ADD A,R0 ; register serba guna
ADD A,#$23 ; bilangan konstan
ADD A,@R0 ; no memori tak langsung
ADD A,P1 ; no memori langsung (port 1)
```

Perintah SUBB

Isi Akumulator A dikurangi dengan bilangan 1 byte berikut dengan nilai bit Carry, hasil pengurangan akan ditampung kembali dalam Akumulator. Dalam operasi ini bit Carry juga berfungsi sebagai penampung limpahan hasil pengurangan. Jika hasil pengurangan tersebut melimpah (nilainya kurang dari 0) bit Carry akan bernilai '1', kalau tidak bit Carry bernilai '0'.

```
SUBB A,R0 ; A = A - R0 - C
SUBB A,#$23 ; A = A - $23
SUBB A,@R1
SUBB A,P0
```

Perintah DA

Perintah DA (Decimal Adjust) dipakai setelah perintah ADD; ADDC atau SUBB, dipakai untuk merubah nilai biner 8 bit yang tersimpan dalam Akumulator menjadi 2 buah bilangan desimal yang masing-masing terdiri dari nilai biner 4 bit.

Perintah MUL AB

Bilangan biner 8 bit dalam Akumulator A dikalikan dengan bilangan biner 8 bit dalam register B. Hasil perkalian berupa bilangan biner 16 bit, 8 bit bilangan biner yang bobotnya lebih besar ditampung di register B, sedangkan 8 bit lainnya yang bobotnya lebih kecil ditampung di Akumulator A.

Bit OV dalam PSW (Program Status Word) dipakai untuk menandai nilai hasil perkalian yang ada dalam register B. Bit OV akan bernilai '0' jika register B bernilai \$00, kalau tidak bit OV bernilai '1'.

MOV A,#10
 MOV B,#20
 MUL AB

Perintah DIV AB

Bilangan biner 8 bit dalam Akumulator A dibagi dengan bilangan biner 8 bit dalam register B. Hasil pembagian berupa bilangan biner 8 bit ditampung di Akumulator, sedangkan sisa pembagian berupa bilangan biner 8 bit ditampung di register B. Bit OV dalam PSW (Program Status Word) dipakai untuk menandai nilai sebelum pembagian yang ada dalam register B. Bit OV akan bernilai '1' jika register B asalnya bernilai \$00.

Table 1.3. Instruksi Aritmatika

Mnemonic	Operation	Addressing Mode				Exect.
		Dir	Ind	Reg	Imm	Timer uS
Add A,<byte>	A=A+<byte>	V	V	V	V	1
Addc A,<byte>	A=A+<byte>+C	V	V	V	V	1
Subb A,<byte>	A=A-<byte>-C	V	V	V	V	1
Inc A	A=A+1	Accumulator Only				1
Inc <byte>	<byt>=<byt>+1	V	V	V		1
Inc DPTR	DPTR=DPTR+1	Data Pointer Only				2
Dec A	A=A-1	Accumulator Only				1
Dec <byte>	<byt>=<byt>-1	V	V	V		1
Mul AB	B:A=BxA	Accumulator and B Only				4
Div AB	A=Int[A/B] B=Mod[A/B]	Accumulator and B only				4
DA A	Dec Adjust	Accumulator Only				1

1.3.3 Instruksi Logika

Kelompok perintah ini dipakai untuk melakukan operasi logika mikrokontroler MCS51, operasi logika yang bisa dilakukan adalah operasi AND (kode operasi ANL), operasi OR (kode operasi ORL) dan operasi Exclusive-OR (kode operasi XRL).

Data yang dipakai dalam operasi ini bisa berupa data yang berada dalam Akumulator atau data yang berada dalam memori-data, hal ini sedikit berlainan dengan operasi aritmatik yang harus melihatkan Akumulator secara aktif.

Hasil operasi ditampung di sumber data yang pertama.

1. Operasi logika AND banyak dipakai untuk me-‘0’-kan beberapa bit tertentu dari sebuah bilangan biner 8 bit, caranya dengan membentuk sebuah bilangan biner 8 bit sebagai data konstan yang di-ANL-kan bilangan asal. Bit yang ingin di-‘0’-kan diwakili dengan ‘0’ pada data konstan, sedangkan bit lainnya diberi nilai ‘1’, misalnya. Instruksi ANL P1,#%01111110 akan mengakibatkan bit 0 dan bit 7 dari Port 1 (P1) bernilai ‘0’ sedangkan bit-bit lainnya tetap tidak berubah nilai.

2. Operasi logika OR banyak dipakai untuk me-‘1’-kan beberapa bit tertentu dari sebuah bilangan biner 8 bit, caranya dengan membentuk sebuah bilangan biner 8 bit sebagai data konstan yang di-ORL-kan bilangan asal. Bit yang ingin di-‘1’-kan diwakili dengan ‘1’ pada data konstan, sedangkan bit lainnya diberi nilai ‘0’, misalnya :Instruksi ORL A,#%01111110 akan mengakibatkan bit 1 sampai dengan bit 6 dari Akumulator bernilai ‘1’ sedangkan bit-bit lainnya tetap tidak berubah nilai.

3. Operasi logika Exclusive-OR banyak dipakai untuk membalik nilai (complement) beberapa bit tertentu dari sebuah bilangan biner 8 bit, caranya dengan membentuk sebuah bilangan biner 8 bit sebagai data konstan yang di-XRL-kan bilangan asal. Bit yang ingin dibalik-nilai diwakili dengan ‘1’ pada data konstan, sedangkan bit lainnya diberi nilai ‘0’, misalnya: Instruksi XRL A,#%01111110 akan mengakibatkan bit 1 sampai dengan bit 6 dari Akumulator berbalik nilai, sedangkan bit-bit lainnya tetap tidak berubah nilai.

Mnemonic	Operation	Addressing Mode				Exect.
		Dir	Ind	Reg	Imm	Timer uS
Anl A,<byte>	A=A and <byte>	V	V	V	V	1
Anl <byte>,A	<byte>=<byte>anl A	V	V	V	V	1
Anl <byte>,#data	<byte>=<byte>and #data	V	V	V	V	1
Orl A,<byte>	A=A or <byte>	Accumulator Only				1
Orl <byte>,A	<byt>=<byt>orl A	V	V	V		1
Orl <byte>,#data	<byte>=<byte> or #data	Data Pointer Only				2
Xrl A,<byte>	A=A xor<byte>	Accumulator Only				1
Xrl<byte>,A	<byt>=<byt>xor A	V	V	V		1

Xrl <byte>,#data	<byte>=<byte>xor #data	Accumulator and B Only	4
CLR A	A=00h	Accumulator only	1
CPL A	A= not A	Accumulator only	1
RL A	Rotate A left 1 bit	Accumulator only	1
RLC A	Rotate A left trough Carry	Accumulator only	1
RR A	Rotate A right 1 bit	Accumulator only	1
RRC	Rotate A right trough carry	Accumulator only	1
SWAP A	Swap nibbles in A	Accumulator only	1

Operasi logika pada umumnya mencakup empat hal, yaitu operasi AND, operasi OR, operasi EX-OR dan operasi NOT. MCS51 hanya bisa melaksanakan tiga jenis operasi logika yang ada, yakni intruksi ANL (AND Logical) untuk operasi AND intruksi ORL (OR Logical) untuk operasi OR, CPL (Complement bit) untuk operasi NOT. Bit Carry pada PSW diperlakukan sebagai 'akumulator bit', dengan demikian operasi AND dan operasi OR dilakukan antara bit yang tersimpan pada bit Carry dengan salah satu dari 256 bit data yang dibahas di atas. Contoh dari intruksi-instruksi ini adalah :

ANL C,P1.1
ANL C,/P1.2

Instruksi ANL C,P1.1 meng-AND-kan nilai pada bit Carry dengan nilai Port 1 bit 1 (P1.1), dan hasil operasi tersebut ditampung pada bit Carry. Instruksi ANL C,/P1.1 persis sama dengan instruksi sebelumnya, hanya saja sebelum di-AND-kan, nilai P1.1 dibalik (complemented) lebih dulu, jika nilai P1.1='0' maka yang di-AND-kan dengan bit Carry adalah '1', demikian pula sebaliknya. Hal serupa berlaku pada instruksi ORL. Instruksi CPL dipakai untuk membalik (complement) nilai semua 256 bit data yang dibahas di atas. Misalnya :

CPL C
CPL P1.0

CPL C akan membalik nilai biner dalam bit Carry (jangan lupa bit Carry merupakan salah satu bit yang ada dalam 256 bit yang dibahas di atas, yakni bit nomor \$E7 atau PSW.7)

1.3.4 Instruksi Lompatan

Pada dasarnya program dijalankan intruksi demi instruksi, artinya selesai menjalankan satu instruksi mikrokontroler langsung menjalankan instruksi berikutnya, untuk keperluan ini mikrokontroler dilengkapi dengan Program Counter yang mengatur pengambilan intruksi secara berurutan. Meskipun demikian, program yang kerjanya hanya berurutan saja tidaklah banyak artinya, untuk keperluan ini mikrokontroler dilengkapi dengan instruksi-instruksi untuk mengatur alur program.

Secara umum kelompok instruksi yang dipakai untuk mengatur alur program terdiri atas instruksi-instruksi JUMP (setara dengan statemen GOTO dalam Pascal), instruksi-instruksi untuk membuat dan memakai sub-rutin/modul (setara dengan PROCEDURE dalam Pascal), instruksi-instruksi JUMP bersyarat (conditional Jump, setara dengan statemen IF .. THEN dalam Pascal). Di samping itu ada pula instruksi PUSH dan POP yang bisa memengaruhi alur program.

Karena Program Counter adalah satu-satunya register dalam mikrokontroler yang mengatur alur program, maka kelompok instruksi pengatur program yang dibicarakan di atas, semuanya merubah nilai Program Counter, sehingga pada saat kelompok instruksi ini dijalankan, nilai Program Counter akan tidak akan runtun dari nilai instruksi sebelumnya.

Selain karena instruksi-instruksi di atas, nilai Program Counter bisa pula berubah karena pengaruh perangkat keras, yaitu saat mikrokontroler di-reset atau menerima sinyal interupsi dari perangkat input/output. Hal ini akan dibicarakan secara detail dibagian lagi.

Mikrokontroler menjalankan intruksi demi instruksi, selesai menjalankan satu instruksi mikrokontroler langsung menjalankan instruksi berikutnya, hal ini dilakukan dengan cara nilai Program Counter bertambah sebanyak jumlah byte yang membentuk instruksi yang sedang dijalankan, dengan demikian pada saat instruksi bersangkutan dijalankan Program Counter selalu menyimpan nomor memori-program yang menyimpan instruksi berikutnya.

Pada saat mikrokontroler menjalankan kelompok instruksi JUMP, nilai Program Counter yang runtun sesuai dengan alur program diganti dengan nomor memori-program baru yang dikehendaki programer.

Mikrokontroler MCS51 mempunyai 3 macam intruksi JUMP, yakni instruksi LJMP (Long Jump), instruksi AJMP (Absolute Jump) dan instruksi SJMP (Short Jump). Kerja dari ketiga instruksi ini persis sama, yakni memberi nilai baru pada Program Counter, kecepatan melaksanakan ketiga instruksi ini juga persis sama, yakni memerlukan waktu 2 periode instruksi (jika MCS51 bekerja pada frekuensi 12 MHz, maka instruksi ini dijalankan dalam waktu 2 mikro-detik), yang berbeda dalam jumlah byte pembentuk instruksinya, instruksi LJMP dibentuk dengan 3 byte, sedangkan instuksi AJMP dan SJMP cukup 2 byte.

Instruksi LJMP

Kode untuk instruksi LJMP adalah \$02, nomor memori-program baru yang dituju dinyatakan dengan bilangan biner 16 bit, dengan demikian instruksi ini bisa menjangkau semua memori-program MCS51 yang jumlahnya 64 KiloByte. Instruksi LJMP terdiri

atas 3 byte, yang bisa dinyatakan dengan bentuk umum 02 aa aa, aa yang pertama adalah nomor memori-program bit 8 sampai dengan bit 15, sedangkan aa yang kedua adalah nomor memori-program bit 0 sampai dengan bit 7.

Pemakaian instruksi LJMP bisa dipelajari dari potongan program berikut :

```
LJMP TugasBaru
```

```
...
```

```
ORG $2000
```

```
TugasBaru:
```

```
MOV A,P3.1
```

Dalam potongan program di atas, ORG adalah perintah pada assembler agar berikutnya assembler bekerja pada memori-program nomor yang disebut di belakang ORG (dalam hal ini minta assembler berikutnya bekerja pada memori-program nomor \$2000).

TugasBaru disebut sebagai LABEL, yakni sarana assembler untuk menandai/ menamai nomor memori-program. Dengan demikian, dalam potongan program di atas, memori-program nomor \$2000 diberi nama TugasBaru, atau bisa juga dikatakan bahwa TugasBaru bernilai \$2000. (Catatan : LABEL ditulis minimal satu huruf lebih kiri dari instruksi, artinya LABEL ditulis setelah menekan tombol Enter, tapi instruksi ditulis setelah menekan tombol Enter, kemudian diikuti dengan 1 tombol spasi atau tombol TAB).

Dengan demikian intruksi LJMP TugasBaru di atas, sama artinya dengan LJMP \$2000 yang oleh assembler akan diterjemahkan menjadi 02 20 00 (heksadesimal).

Instruksi AJMP

Nomor memori-program baru yang dituju dinyatakan dengan bilangan biner 11 bit, dengan demikian instruksi ini hanya bisa menjangkau satu daerah memori-program MCS51 sejauh 2 KiloByte. Instruksi AJMP terdiri atas 2 byte, byte pertama merupakan kode untuk instruksi AJMP (00001b) yang digabung dengan nomor memori-program bit nomor 8 sampai dengan bit nomor 10, byte kedua dipakai untuk menyatakan nomor memori-program bit nomor 0 sampai dengan bit nomor 7.

Berikut ini adalah potongan program untuk menjelaskan pemakaian instruksi AJMP:

```
ORG $800
```

```
AJMP DaerahIni
```

```
AJMP DaerahLain
```

```
ORG $900
```

```
DaerahIni:
```

```
...
```

```
ORG $1000
```

```
DaerahLain:
```

```
...
```

Potongan program di atas dimulai di memori-program nomor \$800, dengan demikian instruksi AJMP DaerahIni bisa dipakai, karena nomor-memori \$800 (tempat instruksi AJMP DaerahIni) dan LABEL DaerahIni yang terletak di dalam satu daerah memori-

program 2 KiloByte yang sama dengan. (Dikatakan terletak di dalam satu daerah memori-program 2 KiloByte yang sama, karena bit nomor 11 sampai dengan bit nomor 15 dari nomor memorinya sama).

Tapi AJMP DaerahLain akan di-salah-kan oleh Assembler, karena DaerahLain yang terletak di memori-program nomor \$1000 terletak di daerah memori-program 2 KiloByte yang lain.

Karena instruksi AJMP hanya terdiri dari 2 byte, sedangkan instruksi LJMP 3 byte, maka memakai instruksi AJMP lebih hemat memori-program dibanding dengan LJMP. Hanya saja karena jangkauan instruksi AJMP hanya 2 KiloByte, pemakaiannya harus hati-hati. Memori-program IC mikrokontroler AT89C1051 dan AT89C2051 masing-masing hanya 1 KiloByte dan 2 KiloByte, dengan demikian program untuk kedua mikrokontroler di atas tidak perlu memakai instruksi LJMP, karena program yang ditulis tidak mungkin menjangkau lebih dari 2 KiloByte memori-program.

Instruksi SJMP

Nomor memori-program dalam instruksi ini tidak dinyatakan dengan nomor memori-program yang sesungguhnya, tapi dinyatakan dengan 'pergeseran relatif' terhadap nilai Program Counter saat instruksi ini dilaksanakan.

Pergeseran relatif tersebut dinyatakan dengan 1 byte bilangan 2's complement, yang bisa dipakai untuk menyatakan nilai antara -128 sampai dengan +127. Nilai minus dipakai untuk menyatakan bergeser ke instruksi-instruksi sebelumnya, sedangkan nilai positif untuk menyatakan bergeser ke instruksi-instruksi sesudahnya.

Meskipun jangkauan instruksi SJMP hanya -128 sampai +127, tapi instruksi ini tidak dibatasi dengan pengertian daerah memori-program 2 KiloByte yang membatasi instruksi AJMP.

```
ORG $0F80
SJMP DaerahLain
...
ORG $1000
DaerahLain:
```

Dalam potongan program di atas, memori-program \$0F80 tidak terletak dalam daerah memori-program 2 KiloByte yang sama dengan \$1000, tapi instruksi SJMP DaerahLain tetap bisa dipakai, asalkan jarak antara instruksi itu dengan LABEL DaerahLain tidak lebih dari 127 byte.

Instruksi sub-rutin

Instruksi-instruksi untuk membuat dan memakai sub-rutin/modul program, selain melibatkan Program Counter, melibatkan pula Stack yang diatur oleh Register Stack Pointer.

Sub-rutin merupakan suatu potong program yang karena berbagai pertimbangan dipisahkan dari program utama. Bagian-bagian di program utama akan 'memanggil' (CALL) sub-rutin, artinya mikrokontroler sementara meninggalkan alur program utama untuk mengerjakan instruksi-instruksi dalam sub-rutin, selesai mengerjakan sub-rutin mikrokontroler kembali ke alur program utama.

Satu-satunya cara membentuk sub-rutin adalah memberi instruksi RET pada akhir potongan program sub-rutin. Program sub-rutin di-'panggil' dengan instruksi ACALL atau LCALL.

Agar nantinya mikrokontroler bisa meneruskan alur program utama, pada saat menerima instruksi ACALL atau LCALL, sebelum mikrokontroler pergi mengerjakan sub-rutin, nilai Program Counter saat itu disimpan dulu ke dalam Stack (Stack adalah sebagian kecil dari memori-data yang dipakai untuk menyimpan nilai Program Counter secara otomatis, kerja dari Stack dikendalikan oleh Register Stack Pointer).

Selanjutnya mikrokontroler mengerjakan instruksi-instruksi di dalam sub-rutin sampai menjumpai instruksi RET yang berfungsi sebagai penutup dari sub-rutin. Saat menerima instruksi RET, nilai asal Program Counter sebelum mengerjakan sub-rutin yang disimpan di dalam Stack, dikembalikan ke Program Counter sehingga mikrokontroler bisa meneruskan pekerjaan di alur program utama.

Instruksi ACALL dipakai untuk me-'manggil' program sub-rutin dalam daerah memori-program 2 KiloByte yang sama, setara dengan instruksi AJMP yang sudah dibahas di atas. Sedangkan instruksi LCALL setara dengan instruksi LCALL, yang bisa menjangkau seluruh memori-program mikrokontroler MCS51 sebanyak 64 KiloByte. (Tapi tidak ada instruksi SCALL yang setara dengan instruksi SJMP).

Program untuk AT89C1051 dan AT89C2051 tidak perlu memakai instruksi LCALL. Instruksi RET dipakai untuk mengakhiri program sub-rutin, di samping itu masih ada pula instruksi RETI, yakni instruksi yang dipakai untuk mengakhiri Program Layanan Interupsi (Interrupt Service Routine), yaitu semacam program sub-rutin yang dijalankan mikrokontroler pada saat mikrokontroler menerima sinyal permintaan interupsi.

Catatan : Saat mikrokontroler menerima sinyal permintaan interupsi, mikrokontroler akan melakukan satu hal yang setara dengan instruksi LCALL untuk menjalankan Program Layanan Interupsi dari sinyal interupsi bersangkutan. Di samping itu, mikrokontroler juga me-'mati'-kan sementara mekanisme layanan interupsi, sehingga permintaan interupsi berikutnya tidak dilayani. Saat menerima instruksi RETI, mekanisme layanan interupsi kembali diaktifkan dan mikrokontroler melaksanakan hal yang setara dengan instruksi RET.

Instruksi Lompatan Bersyarat

Instruksi Jump bersyarat merupakan instruksi inti bagi mikrokontroler, tanpa kelompok instruksi ini program yang ditulis tidak banyak berarti. Instruksi-instruksi ini selain melibatkan Program Counter, melibatkan pula kondisi-kondisi tertentu yang biasanya dicatat dalam bit-bit tertentu yang dihimpun dalam Register tertentu.

Khusus untuk keluarga mikrokontroler MCS51 yang mempunyai kemampuan menangani operasi dalam level bit, instruksi jump bersyarat dalam MCS51 dikaitkan pula dengan kemampuan operasi bit MCS51.

Nomor memori-program baru yang harus dituju tidak dinyatakan dengan nomor memori-program yang sesungguhnya, tapi dinyatakan dengan 'pergeseran relatif' terhadap nilai Program Counter saat instruksi ini dilaksanakan. Cara ini dipakai pula untuk instruksi SJMP.

Instruksi JZ / JNZ

Instruksi JZ (Jump if Zero) dan instruksi JNZ (Jump if not Zero) adalah instruksi JUMP bersyarat yang memantau nilai Akumulator A.

```
MOV A,#0
JNZ BukanNol
JZ Nol
...
BukanNol:
...
Nol :
...
```

Dalam contoh program di atas, MOV A,#0 membuat A bernilai nol, hal ini mengakibatkan instruksi JNZ BukanNol tidak akan pernah dikerjakan (JNZ artinya Jump kalau nilai $A \neq 0$, syarat ini tidak pernah dipenuhi karena saat instruksi ini dijalankan nilai $A=0$), sedangkan instruksi JZ Nol selalu dikerjakan karena syaratnya selalu dipenuhi.

Instruksi JC / JNC

Instruksi JC (Jump on Carry) dan instruksi JNC (Jump on no Carry) adalah instruksi jump bersyarat yang memantau nilai bit Carry di dalam Program Status Word (PSW). Bit Carry merupakan bit yang banyak sekali dipakai untuk keperluan operasi bit, untuk menghemat pemakaian memori-program disediakan 2 instruksi yang khusus untuk memeriksa keadaan bit Carry, yakni JC dan JNC. Karena bit akan diperiksa sudah pasti, yakni bit Carry, maka instruksi ini cukup dibentuk dengan 2 byte saja, dengan demikian bisa lebih menghemat memori program.

```
JC Periksa
JB PSW.7,Periksa
```

Hasil kerja kedua instruksi di atas sama, yakni MCS51 akan JUMP ke Periksa jika ternyata bit Carry bernilai '1' (ingat bit Carry sama dengan PSW bit 7). Meskipun sama tapi instruksi JC Periksa lebih pendek dari instruksi JB PSW.7,Periksa, instruksi pertama dibentuk dengan 2 byte dan instruksi yang kedua 3 byte. Instruksi JBC sama dengan instruksi JB, hanya saja jika ternyata bit yang diperiksa memang benar bernilai '1', selain MCS51 akan JUMP ke instruksi lain yang dikehendaki MCS51 akan me-nol-kan bit yang baru saja diperiksa

Instruksi JB / JNB / JBC

Instruksi JB (Jump on Bit Set), instruksi JNB (Jump on not Bit Set) dan instruksi JBC (Jump on Bit Set Then Clear Bit) merupakan instruksi Jump bersyarat yang memantau nilai-nilai bit tertentu. Bit-bit tertentu bisa merupakan bit-bit dalam register status maupun kaki input mikrokontroler MCS51.

Pengujian Nilai Boolean dilakukan dengan instruksi JUMP bersyarat, ada 5 instruksi yang dipakai untuk keperluan ini, yakni instruksi JB (JUMP if bit set), JNB (JUMP if bit

Not Set), JC (JUMP if Carry Bit set), JNC (JUMP if Carry Bit Not Set) dan JBC (JUMP if Bit Set and Clear Bit).

Dalam instruksi JB dan JNB, salah satu dari 256 bit yang ada akan diperiksa, jika keadaannya (false atau true) memenuhi syarat, maka MCS51 akan menjalankan instruksi yang tersimpan di memori-program yang dimaksud. Alamat memori-program dinyatakan dengan bilangan relatif terhadap nilai Program Counter saat itu, dan cukup dinyatakan dengan angka 1 byte. Dengan demikian instruksi ini terdiri dari 3 byte, byte pertama adalah kode operasinya (\$29 untuk JB dan \$30 untuk JNB), byte kedua untuk menyatakan nomor bit yang harus diuji, dan byte ketiga adalah bilangan relatif untuk instruksi tujuan.

Contoh pemakaian instruksi JB dan JNB sebagai berikut :

```
JB P1.1,$
```

```
JNB P1.1,$
```

Instruksi-instruksi di atas memantau keadaan kaki IC MCS51 Port 1 bit 1. Instruksi pertama memantau P1.1, jika P1.1 bernilai '1' maka MCS51 akan mengulang instruksi ini, (tanda \$ mempunyai arti jika syarat terpenuhi kerjakan lagi instruksi bersangkutan). Instruksi berikutnya melakukan hal sebaliknya, yakni selama P1.1 bernilai '0' maka MCS51 akan tertahan pada instruksi ini.

Instruksi proses dan test

Instruksi-instruksi Jump bersyarat yang dibahas di atas, memantau kondisi yang sudah terjadi yang dicatat MCS51. Ada dua instruksi yang melakukan dulu suatu proses baru kemudian memantau hasil proses untuk menentukan apakah harus Jump. Kedua instruksi yang dimaksud adalah instruksi DJNZ dan instruksi CJNE.

Instruksi DJNZ

Instruksi DJNZ (Decrement and Jump if not Zero), merupakan instruksi yang akan mengurangi 1 nilai register serbaguna (R0..R7) atau memori-data, dan Jump jika ternyata setelah pengurangan 1 tersebut hasilnya tidak nol.

Contoh berikut merupakan potongan program untuk membentuk waktu tunda secara sederhana :

```
MOV R0,#$23
```

```
DJNZ R0,$
```

Instruksi MOV R0,#\$23 memberi nilai \$23 pada R0, selanjutnya setiap kali instruksi DJNZ R0,\$ dikerjakan, MCS51 akan mengurangi nilai R0 dengan '1', jika R0 belum menjadi nol maka MCS51 akan mengulang instruksi tersebut (tanda \$ dalam instruksi ini maksudnya adalah kerjakan kembali instruksi ini). Selama mengerjakan 2 instruksi di atas, semua pekerjaan lain akan tertunda, waktu tundanya ditentukan oleh besarnya nilai yang diisikan ke R0.

Instruksi CJNE

Instruksi CJNE (Compare and Jump if Not Equal) membandingkan dua nilai yang disebut dan MCS akan Jump kalau kedua nilai tersebut tidak sama!

```
MOV A,P0  
CJNE A,#0FBh,TidakSama  
( Kondisi ya atau sama)
```

```
SJMP EXIT
```

```
;
```

```
TidakSama:
```

```
...
```

Instruksi MOV A,P0 membaca nilai input dari Port 0, instruksi CJNE A,#0FBh ,Tidaksama memeriksa apakah nilai Port 0 yang sudah disimpan di A sama dengan #0FBh, jika tidak maka Jump ke TidakSama

1.7. Bahasa Assembly

Secara fisik, kerja dari sebuah mikrokontroler dapat dijelaskan sebagai siklus pembacaan instruksi yang tersimpan di dalam memori. Mikrokontroler menentukan alamat dari memori program yang akan dibaca, dan melakukan proses baca data di memori. Data yang dibaca diinterpretasikan sebagai instruksi. Alamat instruksi disimpan oleh mikrokontroler di register, yang dikenal sebagai program counter. Instruksi ini misalnya program aritmatika yang melibatkan 2 register. Sarana yang ada dalam program assembly sangat minim, tidak seperti dalam bahasa pemrograman tingkat atas (high level language programming) semuanya sudah siap pakai. Penulis program assembly harus menentukan segalanya, menentukan letak program yang ditulisnya dalam memori-program, membuat data konstan dan tabel konstan dalam memori-program, membuat variabel yang dipakai kerja dalam memori-data dan lain sebagainya.

1.7.1 Program sumber assembly

Program-sumber assembly (assembly source program) merupakan kumpulan dari baris-baris perintah yang ditulis dengan program penyunting-teks (text editor) sederhana, misalnya program EDIT.COM dalam DOS, atau program NOTEPAD dalam Windows atau MIDE-51. Kumpulan baris-perintah tersebut biasanya disimpan ke dalam file dengan nama ekstensi *.ASM dan lain sebagainya, tergantung pada program Assembler yang akan dipakai untuk mengolah program-sumber assembly tersebut.

Setiap baris-perintah merupakan sebuah perintah yang utuh, artinya sebuah perintah tidak mungkin dipecah menjadi lebih dari satu baris. Satu baris perintah bisa terdiri atas 4 bagian, bagian pertama dikenali sebagai label atau sering juga disebut sebagai symbol, bagian kedua dikenali sebagai kode operasi, bagian ketiga adalah operand dan bagian terakhir adalah komentar.

Antara bagian-bagian tersebut dipisahkan dengan sebuah spasi atau tabulator.

Bagian label

Label dipakai untuk memberi nama pada sebuah baris-perintah, agar bisa mudah menyebitnya dalam penulisan program. Label bisa ditulis apa saja asalkan diawali dengan huruf, biasa panjangnya tidak lebih dari 16 huruf. Huruf-huruf berikutnya boleh merupakan angka atau tanda titik dan tanda garis bawah. Kalau sebuah baris-perintah tidak memiliki bagian label, maka bagian ini boleh tidak ditulis namun spasi atau tabulator sebagai pemisah antara label dan bagian berikutnya mutlak tetap harus ditulis. Dalam sebuah program sumber bisa terdapat banyak sekali label, tapi tidak boleh ada label yang kembar.

Sering sebuah baris-perintah hanya terdiri dari bagian label saja, baris demikian itu memang tidak bisa dikatakan sebagai baris-perintah yang sesungguhnya, tapi hanya sekedar memberi nama pada baris bersangkutan.

Bagian label sering disebut juga sebagai bagian symbol, hal ini terjadi kalau label tersebut tidak dipakai untuk menandai bagian program, melainkan dipakai untuk menandai bagian data.

Bagian kode operasi

Kode operasi (operation code atau sering disingkat sebagai OpCode) merupakan bagian perintah yang harus dikerjakan. Dalam hal ini dikenal dua macam kode operasi, yang pertama adalah kode-operasi untuk mengatur kerja mikroprosesor / mikrokontroler. Jenis kedua dipakai untuk mengatur kerja program assembler, sering dinamakan sebagai assembler directive.

Kode-operasi ditulis dalam bentuk mnemonic, yakni bentuk singkatan-singkatan yang relatif mudah diingat, misalnya adalah MOV, ACALL, RET dan lain sebagainya. Kode-operasi ini ditentukan oleh pabrik pembuat mikroprosesor/mikrokontroler, dengan demikian setiap prosesor mempunyai kode-operasi yang berlainan.

Kode-operasi berbentuk mnemonic tidak dikenal mikroprosesor/mikrokontroler, agar program yang ditulis dengan kode mnemonic bisa dipakai untuk mengendalikan prosesor, program semacam itu diterjemahkan menjadi program yang dibentuk dari kode-operasi kode-biner, yang dikenali oleh mikroprosesor/mikrokontroler.

Tugas penerjemahan tersebut dilakukan oleh program yang dinamakan sebagai Program Assembler.

Di luar kode-operasi yang ditentukan pabrik pembuat mikroprosesor/mikrokontroler, ada pula kode-operasi untuk mengatur kerja dari program assembler, misalnya dipakai untuk menentukan letak program dalam memori (ORG), dipakai untuk membentuk variabel (DS), membentuk tabel dan data konstan (DB, DW) dan lain sebagainya.

Bagian operand

Operand merupakan pelengkap bagian kode operasi, namun tidak semua kode operasi memerlukan operand, dengan demikian bisa terjadi sebuah baris perintah hanya terdiri dari kode operasi tanpa operand. Sebaliknya ada pula kode operasi yang perlu lebih dari satu operand, dalam hal ini antara operand satu dengan yang lain dipisahkan dengan tanda koma.

Bentuk operand sangat bervariasi, bisa berupa kode-kode yang dipakai untuk menyatakan Register dalam prosesor, bisa berupa nomor-memori (alamat memori) yang dinyatakan dengan bilangan atau pun nama label, bisa berupa data yang siap di-operasi-kan.

Semuanya disesuaikan dengan keperluan dari kode-operasi.

Untuk membedakan operand yang berupa nomor-memori atau operand yang berupa data yang siap di-operasi-kan, dipakai tanda-tanda khusus atau cara penulisan yang berlainan.

Di samping itu operand bisa berupa persamaan matematis sederhana atau persamaan Boolean, dalam hal semacam ini program Assembler akan menghitung nilai dari persamaan-persamaan dalam operand, selanjutnya merubah hasil perhitungan tersebut ke kode biner yang dimengerti oleh prosesor. Jadi perhitungan di dalam operand dilakukan oleh program assembler bukan oleh prosesor!

Bagian komentar

Bagian komentar merupakan catatan-catatan penulis program, bagian ini meskipun tidak mutlak diperlukan tapi sangat membantu masalah dokumentasi. Membaca komentar-komentar pada setiap baris-perintah, dengan mudah bisa dimengerti maksud tujuan baris bersangkutan, hal ini sangat membantu orang lain yang membaca program.

Pemisah bagian komentar dengan bagian sebelumnya adalah tanda spasi atau tabulator, meskipun demikian huruf pertama dari komentar sering-sering berupa tanda titik-koma,

merupakan tanda pemisah khusus untuk komentar.

Untuk keperluan dokumentasi yang intensip, sering-sering sebuah baris yang merupakan komentar saja, dalam hal ini huruf pertama dari baris bersangkutan adalah tanda titik-koma.

AT89S51 memiliki sekumpulan instruksi yang sangat lengkap. Instruksi MOV untuk byte dikelompokkan sesuai dengan mode pengalamatan (addressing modes). Mode pengalamatan menjelaskan bagaimana operand dioperasikan. Berikut penjelasan dari berbagai mode pengalamatan. Bentuk program assembly yang umum ialah sebagai berikut :

Label/Symbol	Opcode	Operand	Komentar
	Org	0H	
Start:	Mov	A, #11111110b	
	Mov	R0, #7	; Isi Akumulator
Kiri:	Mov	P0, A	; Isi R0 dengan 7
	Call	Delay	; Copy A ke P0
	RL	A	; Panggil Delay
	DEC	R0	
	CJNE	R0, #0, Kiri	
	Sjmp	Start	
Delay:	mov	R1, #255	
Del1:	mov	R2, #255	
Del2:	djnz	R2, del2	
	djnz	R1, del1	
	ret		
	end		

Isi memori ialah bilangan heksadesimal yang dikenal oleh mikrokontroler kita, yang merupakan representasi dari bahasa assembly yang telah kita buat. Mnemonic atau opcode ialah kode yang akan melakukan aksi terhadap operand . Operand ialah data yang diproses oleh opcode. Sebuah opcode bisa membutuhkan 1 ,2 atau lebih operand, kadang juga tidak perlu operand. Sedangkan komentar dapat kita berikan dengan menggunakan tanda titik koma (;). Berikut contoh jumlah operand yang berbeda beda dalam suatu assembly.

CJNE R5,#22H, aksi ;dibutuhkan 3 buah operand
 MOVX @DPTR, A ;dibutuhkan 2 buah operand
 RL A ;1 buah operand
 NOP ; tidak memerlukan operand

Program yang telah selesai kita buat dapat disimpan dengan ekstension .asm. Lalu kita dapat membuat program objek dengan ekstension HEX dengan menggunakan compiler MIDE-51, yang dijelaskan sebagai berikut:

1.7.2 Assembly Listing

Program-sumber assembly di atas, setelah selesai ditulis diserahkan ke program Assembler untuk diterjemahkan. Setiap prosesor mempunyai program assembler tersendiri, bahkan satu macam prosesor bisa memiliki beberapa macam program Assembler buatan pabrik perangkat lunak yang berlainan.

Hasil utama pengolahan program Assembler adalah program-obyek. Program-obyek ini bisa berupa sebuah file tersendiri, berisikan kode-kode yang siap dikirimkan ke memori-program mikroprosesor/mikrokontroler, tapi ada juga program-obyek yang disisipkan pada program-sumber assembly seperti terlihat dalam Assembly Listing di Gambar 2. Bagian kanan Gambar 2 merupakan program-sumber Assembly karya asli penulis program, setelah diterjemahkan oleh program Assembler kode-kode yang dihasilkan berikut dengan nomor-nomor memori tempat penyimpanan kode-kode tadi, disisipkan pada bagian kiri setiap baris perintah, sehingga bentuk program ini tidak lagi dikatakan sebagai program-sumber assembly tapi dikatakan sebagai Assembly Listing. Membaca Assembly Listing bisa memberikan gambaran yang lebih jelas bagi program yang ditulis, bagi pemula Assembly Listing memberi pengertian yang lebih mendalam tentang isi memori-program, sehingga bisa lebih dibayangkan bagaimana kerja dari sebuah program.

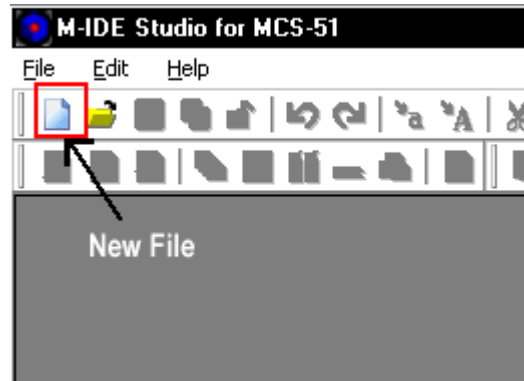
Line	Addr	Code	Source
1:			Org 0H
2:	0000	74 FE	Start: Mov A,#11111110b
3:	0002	78 07	Mov R0,#7
4:	0004	F5 80	Kiri: Mov P0,A
5:	0006	12 00 1C	Call Delay
6:	0009	23	RL A
7:	000A	18	DEC R0
8:	000B	B8 00 F6	CJNE R0,#0,Kiri
9:	000E	78 07	Mov R0,#7
10:	0010	F5 80	Kanan: Mov P0,A
11:	0012	12 00 1C	Call Delay
12:	0015	03	RR A
13:	0016	18	DEC R0
14:	0017	B8 00 F6	CJNE R0,#0,Kanan
15:	001A	80 E4	Sjmp Start
16:	;		
17:	001C	79 FF	Delay: mov R1,#255
18:	001E	7A FF	Del1: mov R2,#255
19:	0020	DA FE	Del2: djnz R2,del2
20:	0022	D9 FA	djnz R1,del1
21:	0024	22	ret
22:			end

1.8. Perangkat Lunak

1.8.1 Compiler MIDE Studio

M-IDE Studio adalah salah satu cara yang digunakan untuk menjalankan kompilasi untuk divais MCS-51. M-IDE Studio mempunyai beberapa fitur yang dapat digunakan untuk edit, compil, dan debug file.

The M-IDE Studio juga dapat digunakan untuk menulis program dalam bahasa C. Dengan menggunakan software ini, maka kita dapat melihat error pada report file LST.



Gambar 1.19. M-IDE Studio

Bila anda perhatikan pada menu toolbar dan menu pilihan, tampak terlihat disable. Hal ini karena file belum dibuat. Untuk membuat sebuah file, lakukan langkah-langkah berikut:

1. Membuat File Baru

Untuk membuat file baru, klik pada menu File atau short cut seperti yang ditunjukkan pada gambar, sehingga akan tampak halaman kosong.

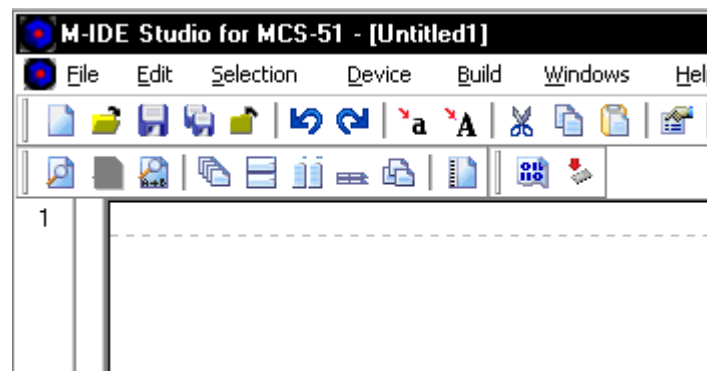
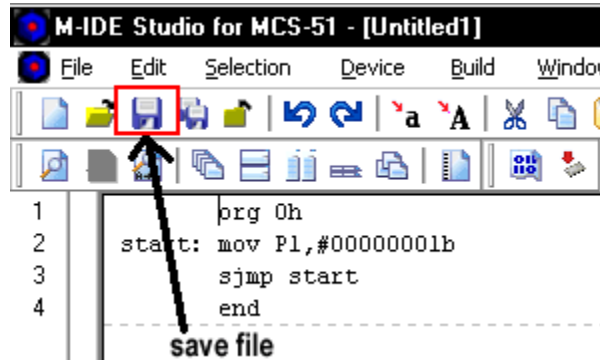


Figure 1.20. File baru dengan halaman kosong

2. Menulis sebuah program

Tulis program assembly pada halaman kosong, dan lakukan penyimpanan file. Bila file

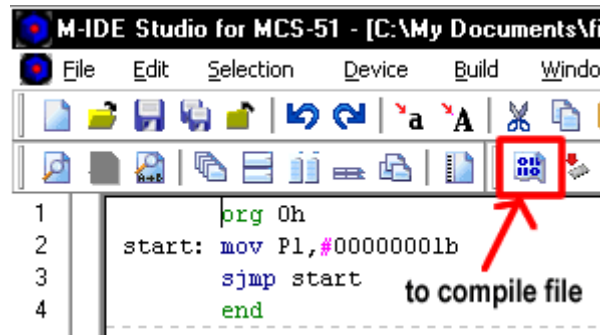
telah tersimpan maka akan tampak teks instruksi yang berwarna-warni. Sebagaimana yang ditunjukkan pada gambar 3.



Gambar 1.21 Menu penyimpanan file

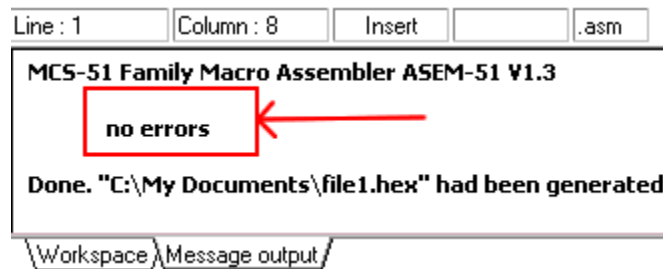
3. Kompilasi

Agar file dengan ekstensi ASM tersebut dapat diupload ke mikrokontroler, maka perlu dilakukan kompilasi dari file ASM ke HEX.



Gambar 1.22 Kompilasi file

4. Debug



Gambar 1.23 Debug file

```
Mulai :  
    Mov A,P2  
    CJNE A,#0FEh,satu      ;banding input S0  
    Mov P3,#0FDh          ;putar kanan  
    SJMP Mulai  
  
satu:  
    CJNE A,#0FDh,dua      ;banding input S1  
    Mov P3,#0FEh          ; putar kiri  
    sjmp Mulai  
  
dua  
    :  
    CJNE A,#0FBh,mulai    ;banding input s3  
    Mov P3,#0FFh  
    sjmp mulai  
  
end
```

```
org 0h
mulai:
    mov a,p2
    cjne a,#0FEh,satu
    mov p1,#0c0h
    sjmp mulai
satu: cjne a,#0FDh,dua
    mov p1,#0F9h
    sjmp mulai
dua: cjne a,#0FBh,mulai
    mov p1,#0A4h
    sjmp mulai
end
```