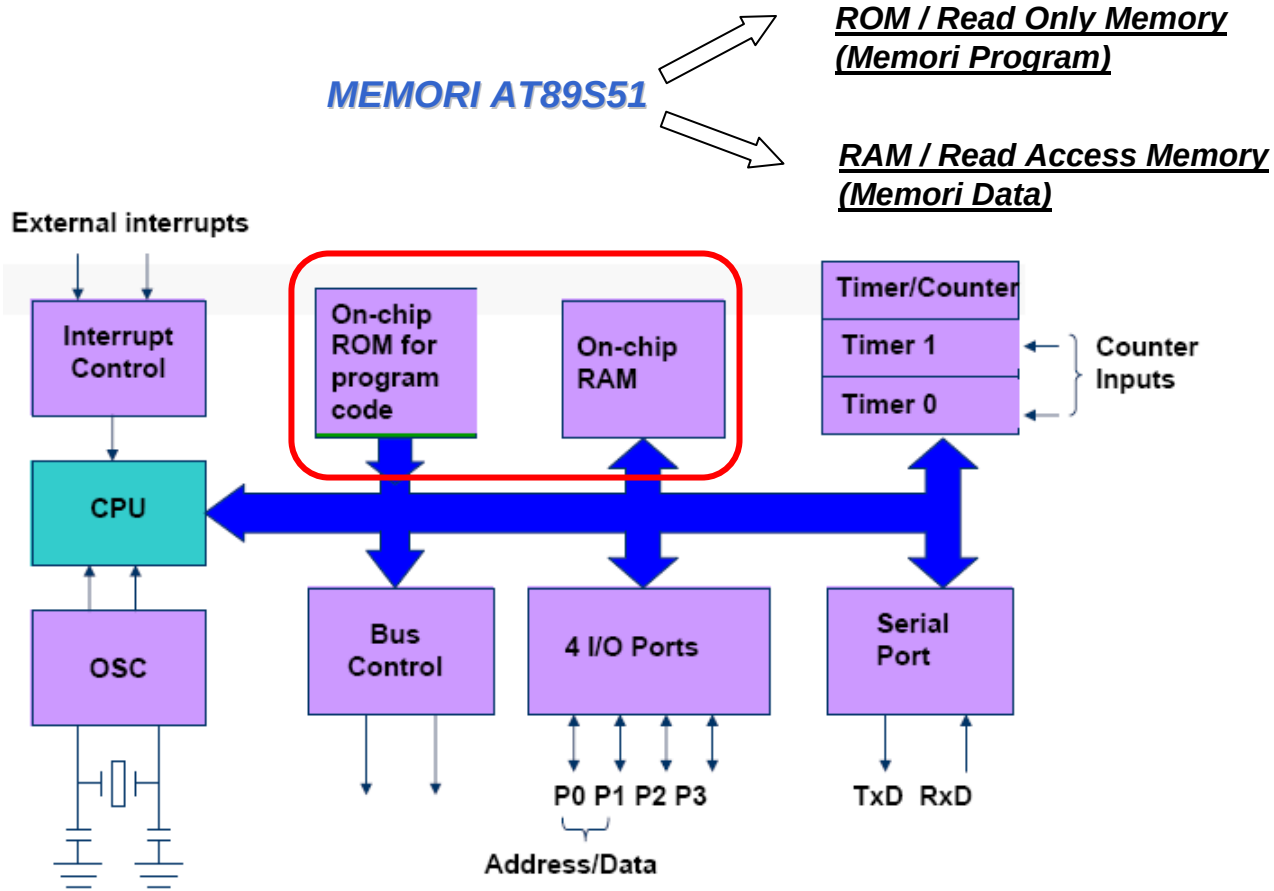
MIKROKONTROLER

Organisasi Memori

Memori AT89S51



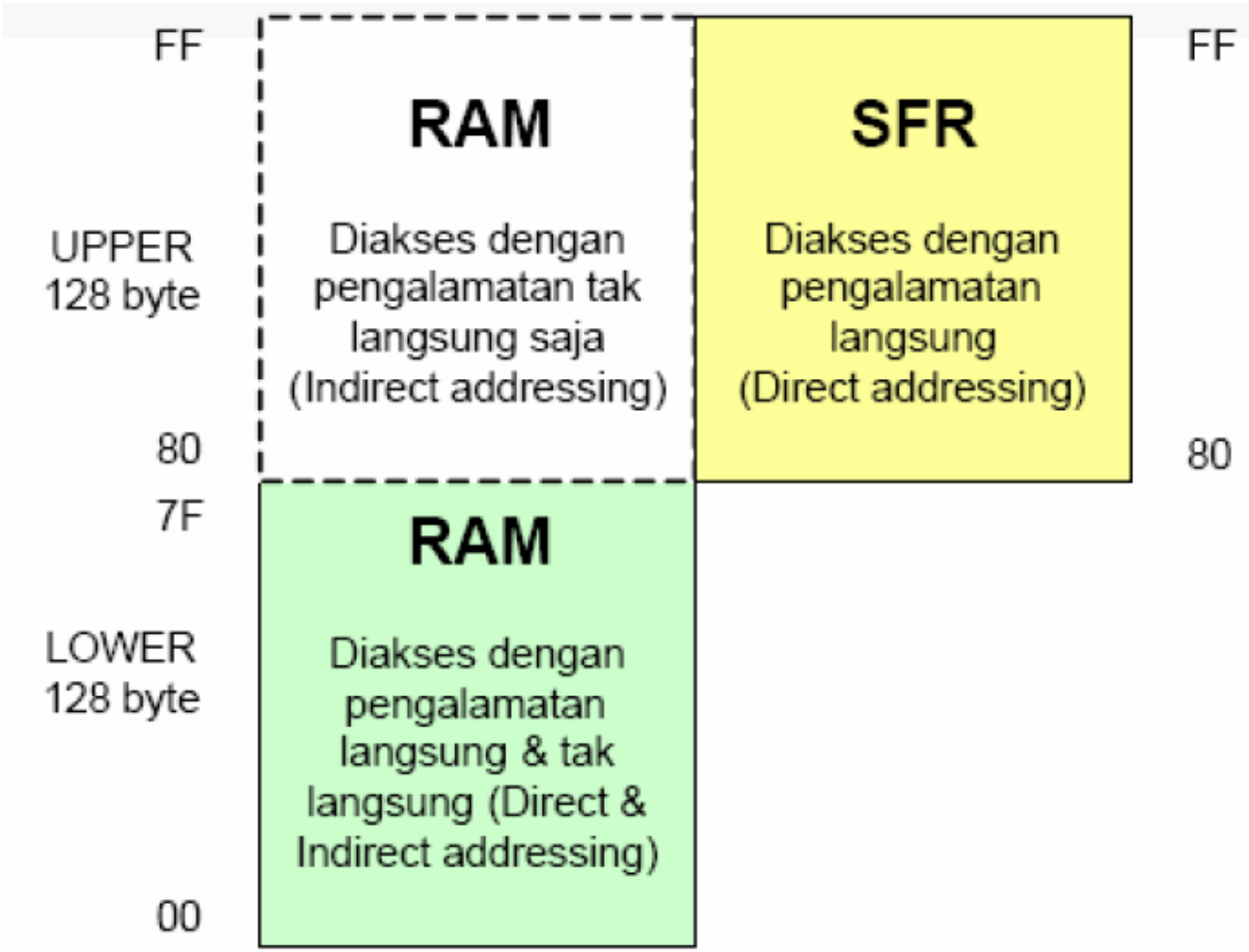
ROM (Memori Program) AT89S51

- ROM (*Read Only Memory)* : Tempat menyimpan program / *source code*
- Sifat ROM : Non Volatile (data/program tidak akan hilang walaupun tegangan supply tidak ada)
- Kapasitas ROM AT89S51 : 4 KByte
- Alamat : 0000 H – 0FFF H
- Diakses Bila pin EA/VPP berlogika High

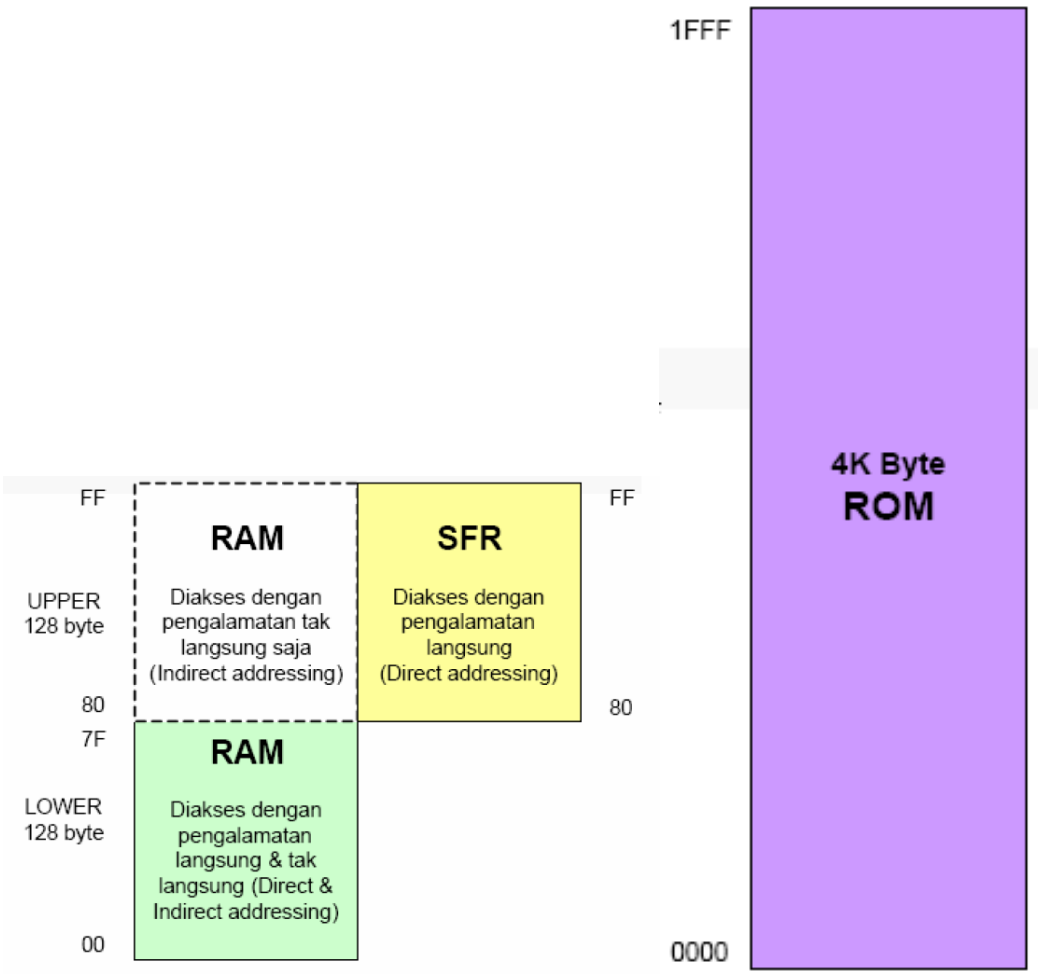
RAM (Memori Data) / AT89S51

- RAM (*Read Access Memory*) : Tempat menyimpan data
- Sifat RAM : *Volatile* (data akan hilang jika tegangan supply tidak ada)
- RAM AT89S51, ada 3 blok:
  - Lower 128 byte (00 H – 7F H) : Dpt diakses dengan pengalamatan langsung maupun tidak langsung
  - Upper 128 byte (80 H – FF H) : Dpt diakses dengan pengalamatan tak langsung saja
  - SFR/Special Function Register (80 H – FFH) : Register yg mempunyai fungsi tertentu. Walaupun pny alamat sama dengan upper 128 byte tp secara fisik berbeda

Peta Memori Data Internal



Peta Memori Internal AT89S51



Catatan:
Gambar disamping adalah peta memori internal 89S51 yang terdiri dari **RAM**, **SFR** dan **ROM**.
Tampak bahwa ada kesamaan address antara **RAM**, **SFR** dan **ROM** yaitu pada address **00 s/d FF**.
Atas pertimbangan inilah maka biasanya source code ditulis setelah address 00FF yaitu **0100** pada ROM
Hal ini dimaksudkan agar data RAM dan SFR tidak terisi oleh byte source code.

Organisasi RAM Internal (Lower Byte)

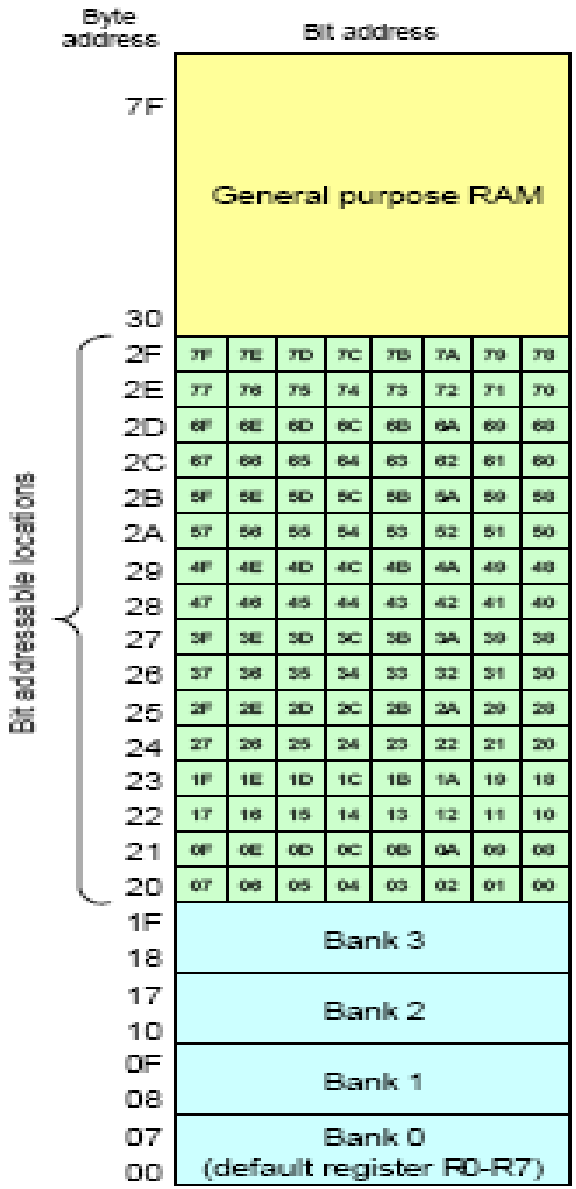
RAM (Random Access Memory) internal AT89S51 berfungsi untuk menyimpan data sementara. Data akan tetap disimpan selama ada supply tegangan ke mikrokontroler.

Pada AT89S51, RAM dibagi menjadi 3 bagian yaitu :

- **Register serba guna**
  Terdiri dari Bank 0, Bank 1, Bank 2, Bank 3.
  Tiap bank register terdiri dari 8 register 8 bit yaitu R0, R1,.. ,R7
  Pemilihan bank register ditentukan pada register PSW
  Rentang address : 00 s/d 1F

- **Bit addressable RAM**
  Adalah RAM yang dapat diakses per bit.
  Ini diperlukan pada saat kita ingin menyimpan data yang panjangnya hanya 1 bit. Setiap bit pada lokasi RAM ini memiliki address sendiri-sendiri seperti terlihat pada gambar.
  Rentang address : 20 s/d 2F

- **General purpose RAM**
  Adalah RAM yang dapat diakses per byte.
  Ini diperlukan pada saat kita ingin menyimpan data yang panjangnya 8 bit.
  Rentang address : 30 s/d 7F



SFR (Special Function Register)

0F8H								0FFH
0F0H	B 00000000							0F7H
0E8H								0EFH
0E0H	ACC 00000000							0E7H
0D8H								0DFH
0D0H	PSW 00000000							0D7H
0C8H								0CFH
0C0H								0C7H
0B8H	IP XX000000							0BFH
0B0H	P3 11111111							0B7H
0A8H	IE 0X000000							0AFH
0A0H	P2 11111111		AUXR1 XXXXXXX0				WDTRST XXXXXXXX	0A7H
98H	SCON 00000000	SBUF XXXXXXXX						9FH
90H	P1 11111111							97H
88H	TCON 00000000	TMOD 00000000	TL0 00000000	TL1 00000000	TH0 00000000	TH1 00000000	AUXR XXX0X0X0	8FH
80H	P0 11111111	SP 00000111	DP0L 00000000	DP0H 00000000	DP1L 00000000	DP1H 00000000		PCON 0XXX0000	87H

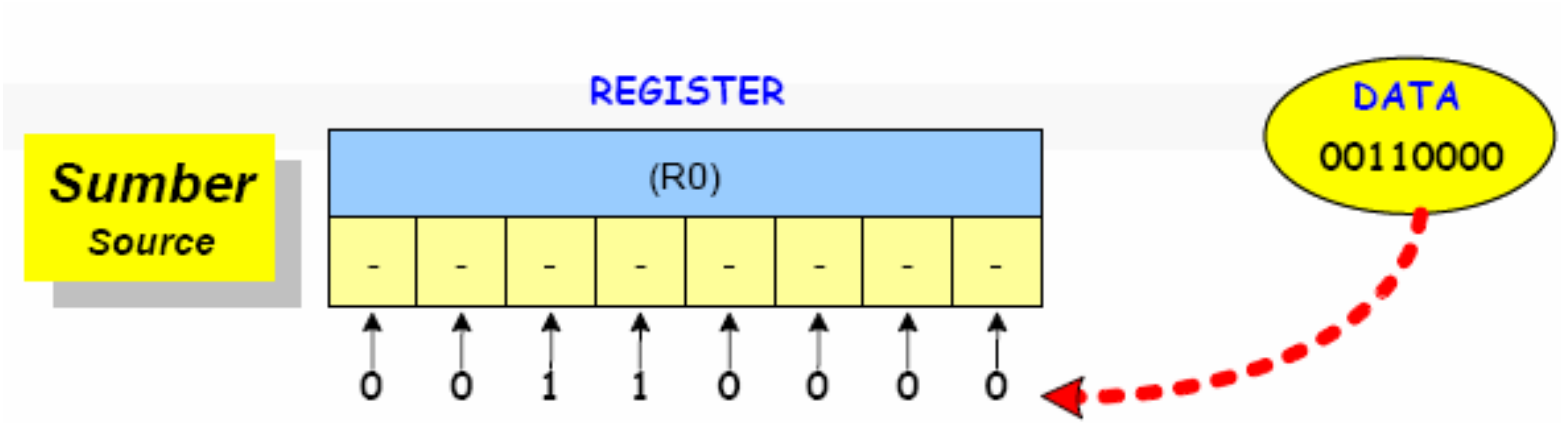
SFR

Simbol	Nama	Alamat
ACC	Akumulator	E0H
B	B register	F0H
PSW	Program Status Word	D0H
SP	Stack Pointer	81H
DPTR0	Data Pointer 0 16 bit DP0L Byte rendah DP0H Byte tinggi	 82H 83H
DPTR1	Data Pointer 0 16 bit DP1L Byte rendah DP1H Byte tinggi	 84H 85H
P0	Port 0	80H
P1	Port 1	90H
P2	Port 2	A0H
P3	Port 3	B0H
IP	Interrupt Priority Control	B8H
IE	Interrupt Enable Control	A8H
TMOD	Timer/Counter Mode Control	89H
TCON	Timer/Counter Control	88H

SFR

Simbol	Nama	Alamat
TH0	Timer/Counter 0 High Byte	8CH
TL0	Timer/Counter 0 Low Byte	8AH
TH1	Timer/Counter 1 High Byte	8DH
TL1	Timer/Counter 1 Low Byte	8BH
SCON	Serial Control	98H
SBUF	Serial Data Buffer	99H
PCON	Power Control	87H
WDTRST	Watchdog Timer Reset	A6H
AUXR	Auxiliary Register	8EH

Immediate Adressing Mode



Immediate Addressing Mode

Immediate Addressing : Data langsung dipindahkan ke register
Contoh :

```
MOV     A,#30h          ; Copy the immediate data 30h to A register
MOV     R0,#00110000B   ; Copy the immediate data 30h to R0 register
MOV     DPTR,#48        ; Copy the immediate data 30h to DPTR register
```

Catatan :
30hexa = **00110000**biner = **48**desimal

Akses Memori Program (ROM) Eksternal

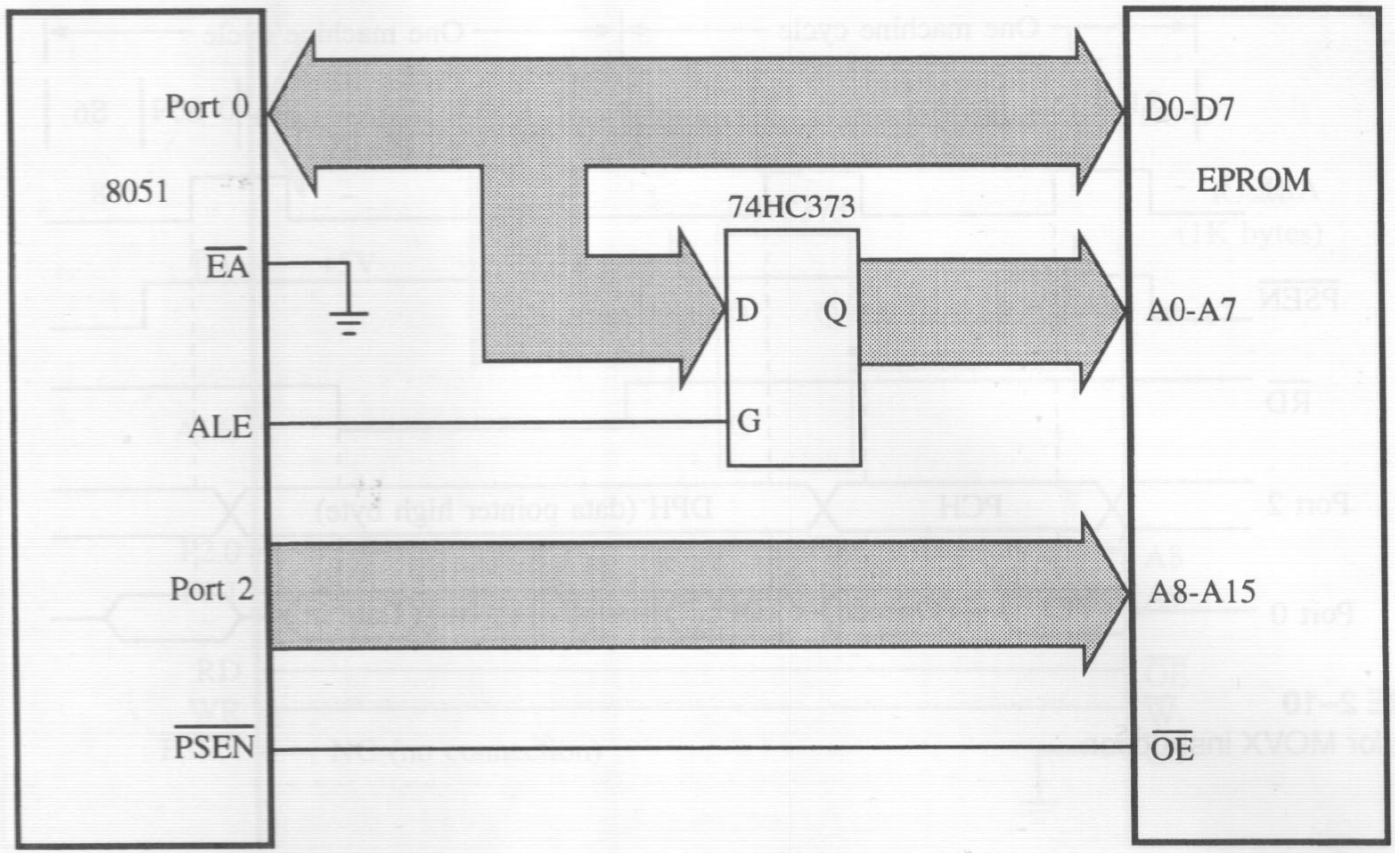
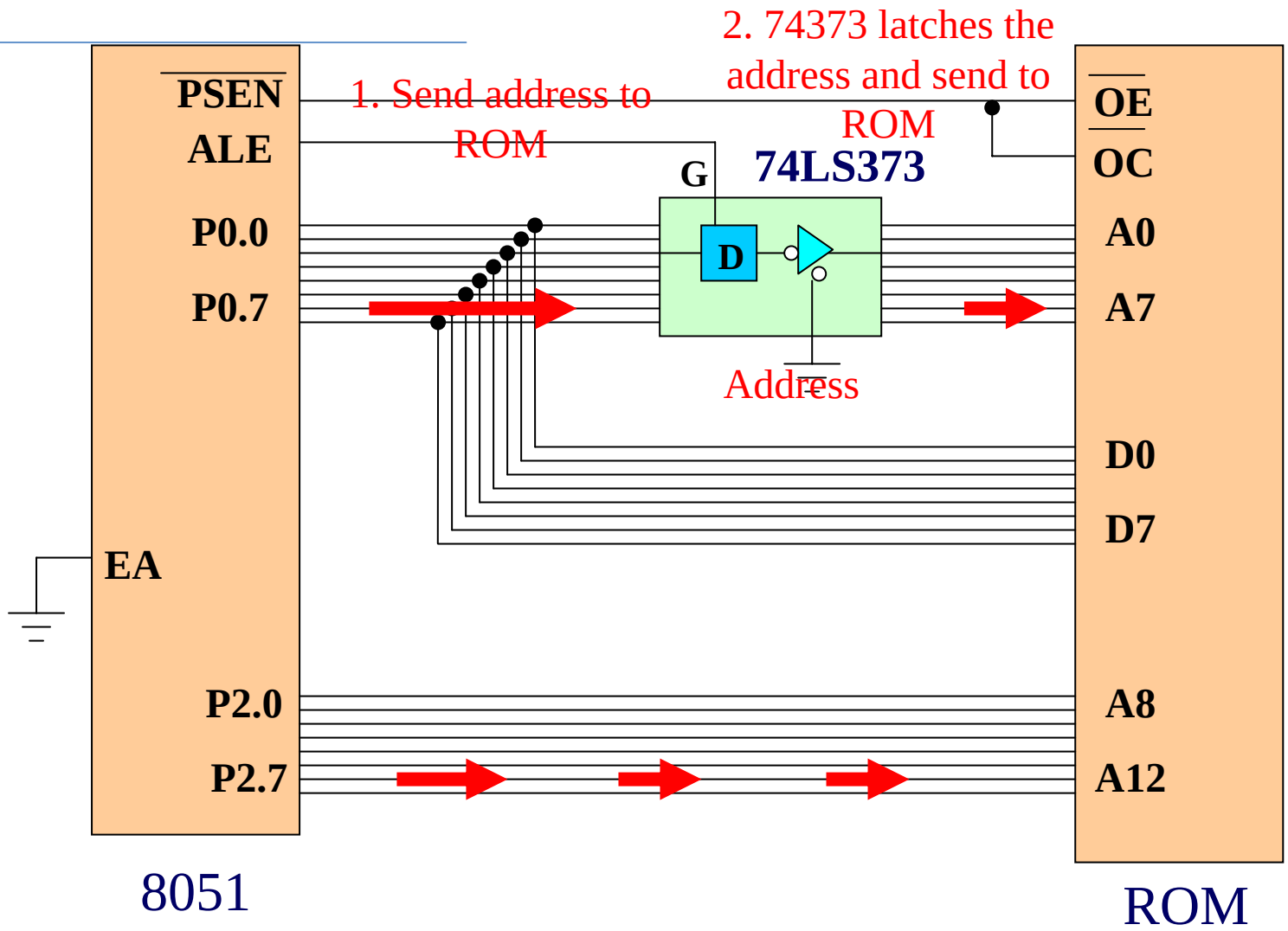
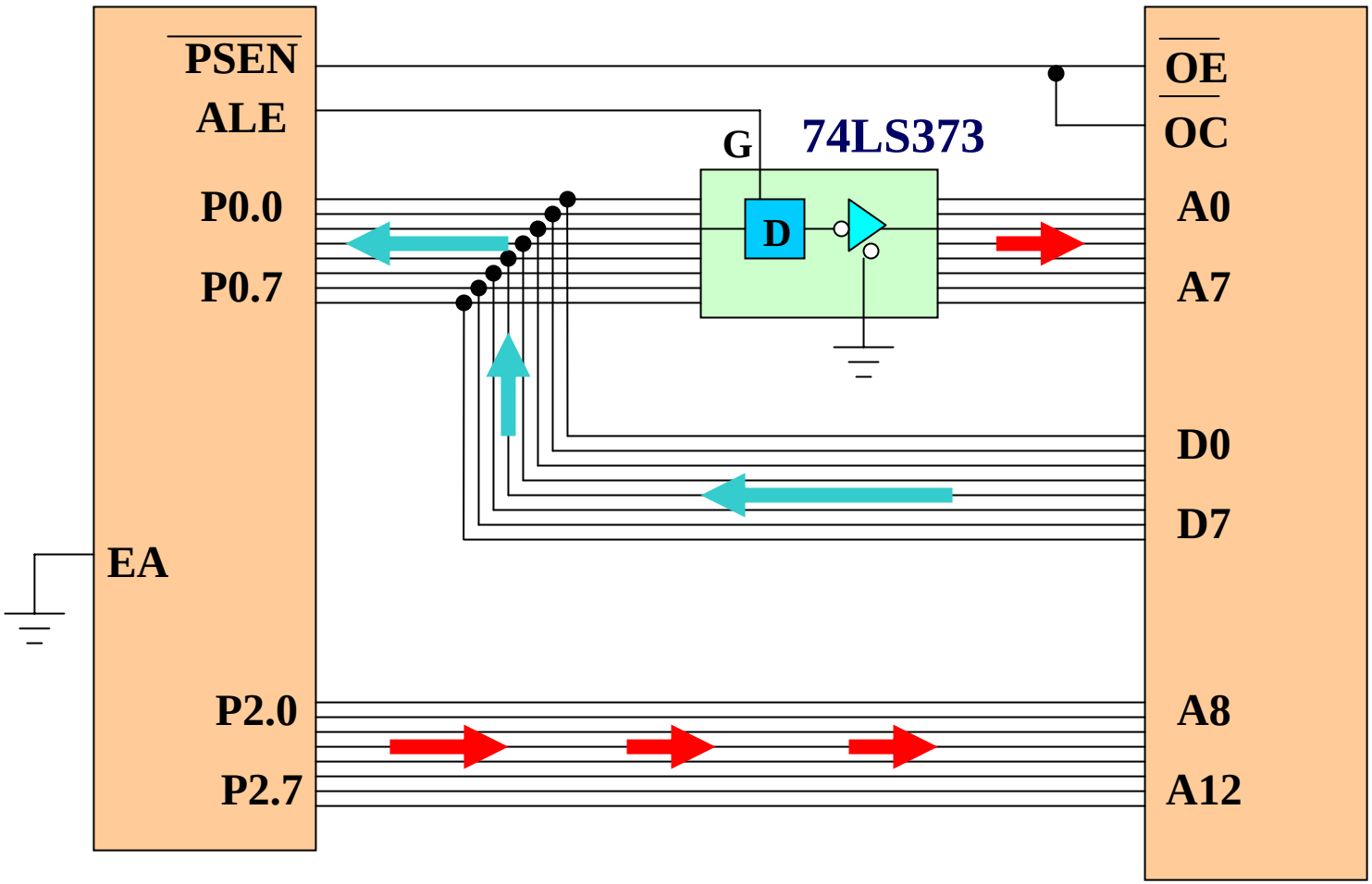


FIGURE 2–8
Accessing external code memory

Membaca Program Dari ROM Eksternal



Membaca Program Dari ROM Eksternal



Akses RAM/ROM Eksternal Lebih Dari 1

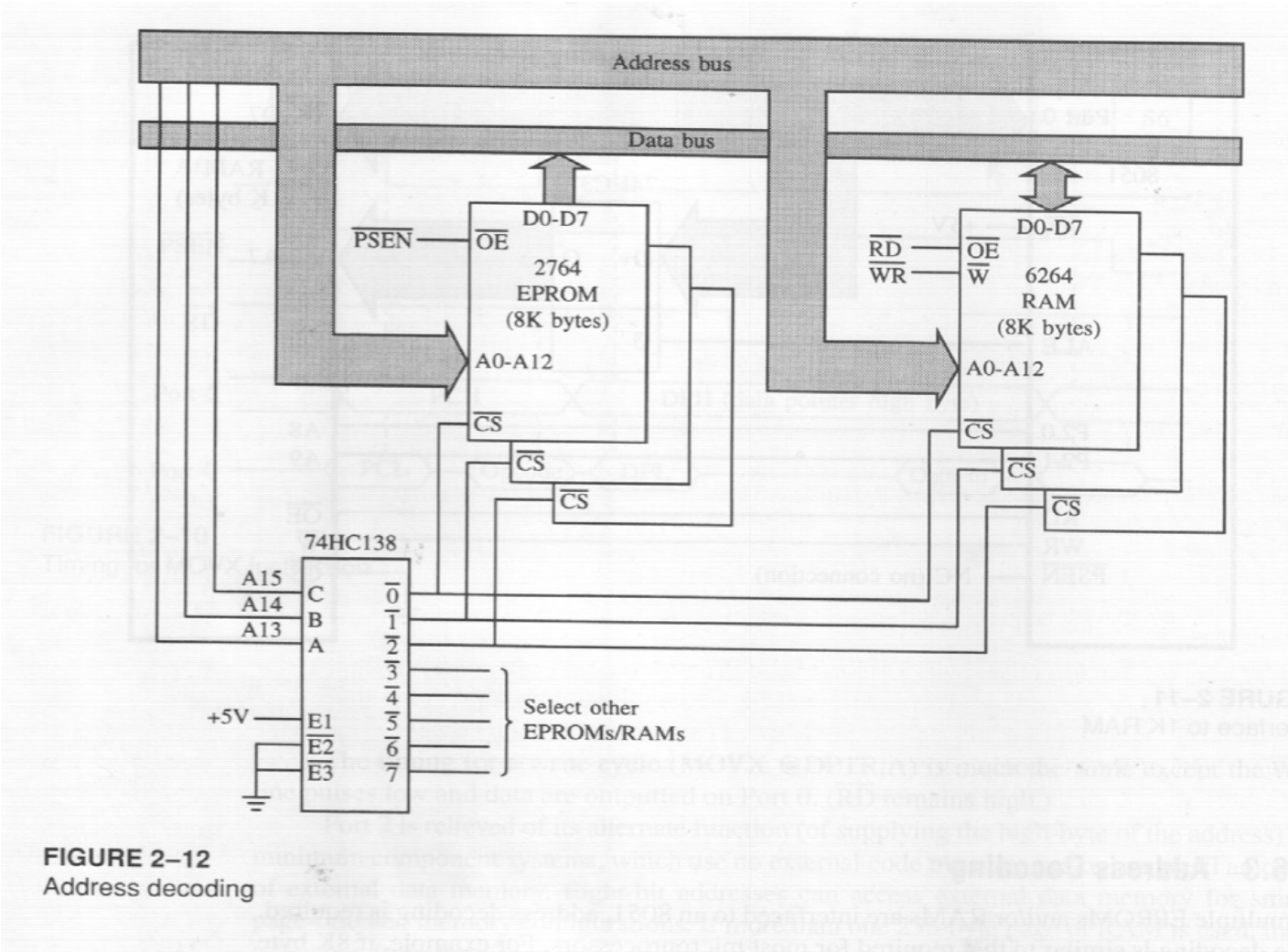


FIGURE 2–12
Address decoding

Akses Memori Data (RAM) Eksternal

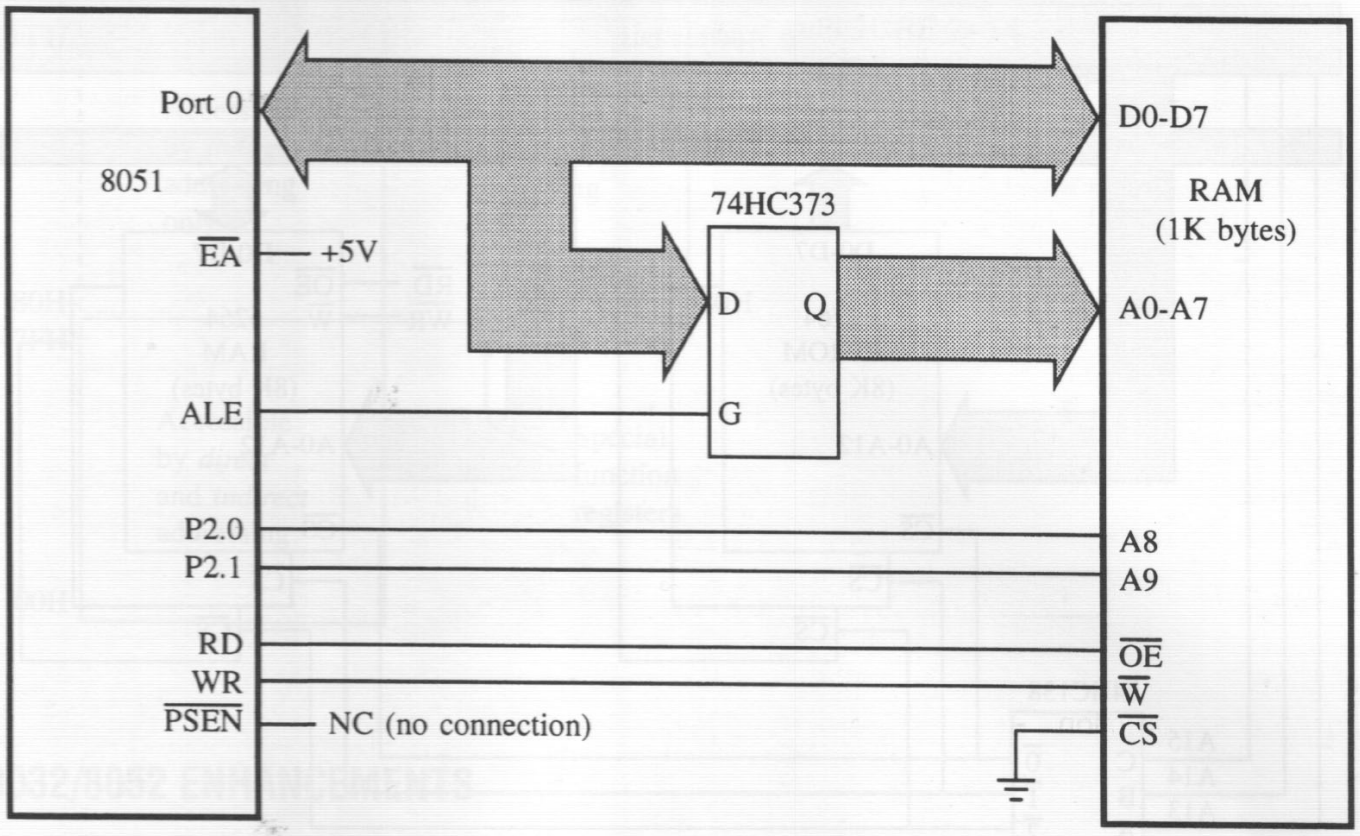
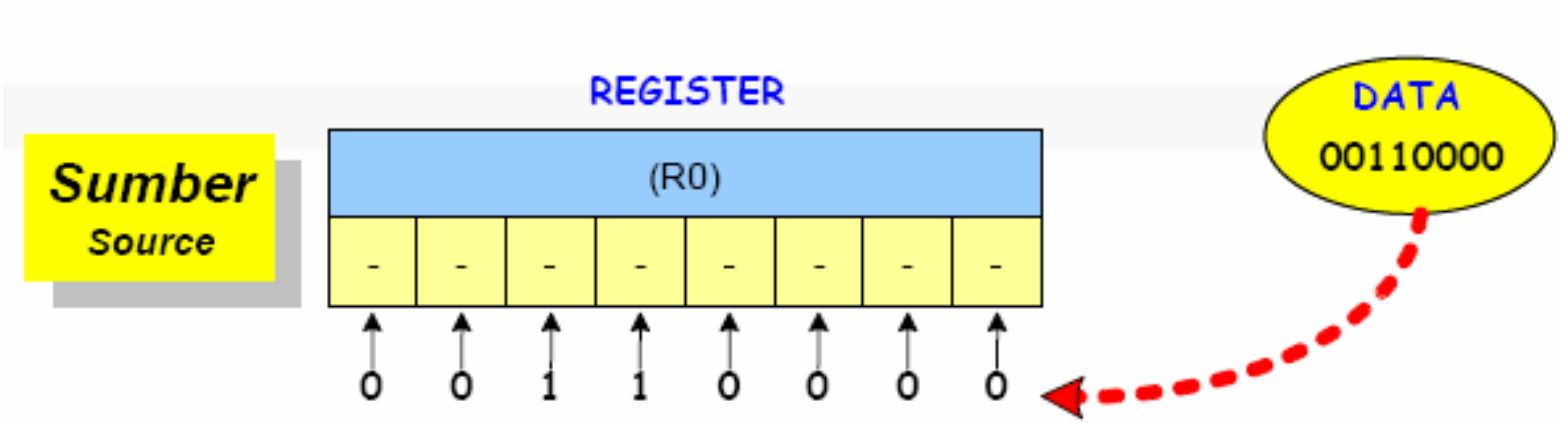


FIGURE 2–11
Interface to 1K RAM

Immediate Adressing Mode



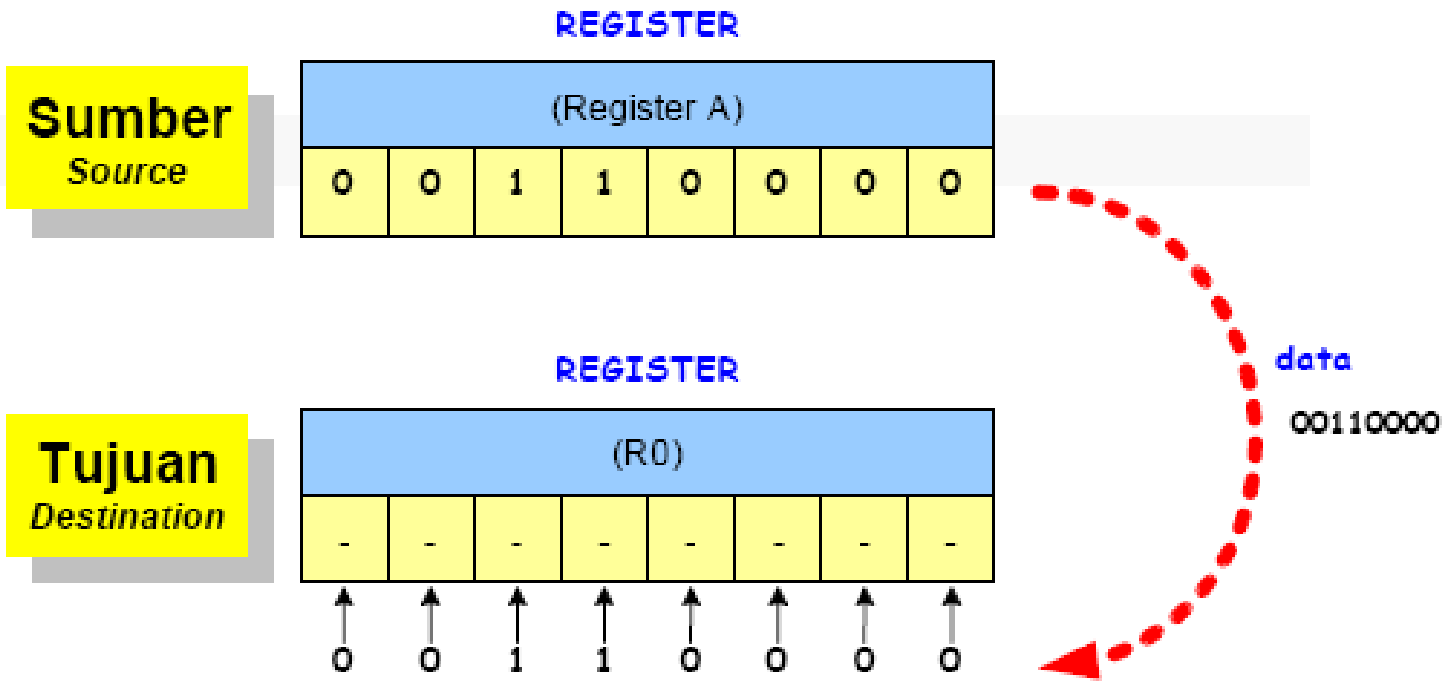
Immediate Addressing Mode

Immediate Addressing : Data langsung dipindahkan ke register
Contoh :

```
MOV     A,#30h          ; Copy the immediate data 30h to A register
MOV     R0,#00110000B   ; Copy the immediate data 30h to R0 register
MOV     DPTR,#48        ; Copy the immediate data 30h to DPTR register
```

Catatan :
30hexa = **00110000**biner = **48**desimal

Register Addressing Mode



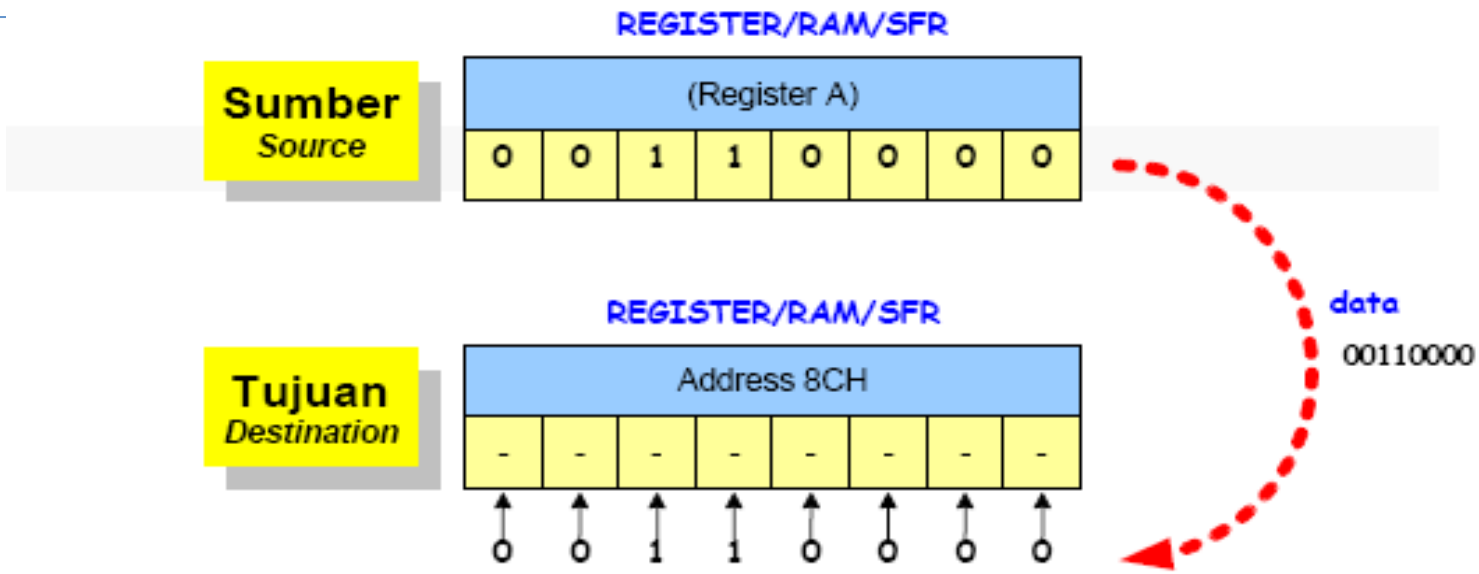
Register Addressing Mode

Register Addressing : Data dari register sumber dipindahkan ke register tujuan

Contoh :

```
MOV     R0,A        ; Copy data from A register to R0 register
MOV     A,R7        ; Copy data from R7register to A register
```

Direct Addressing Mode



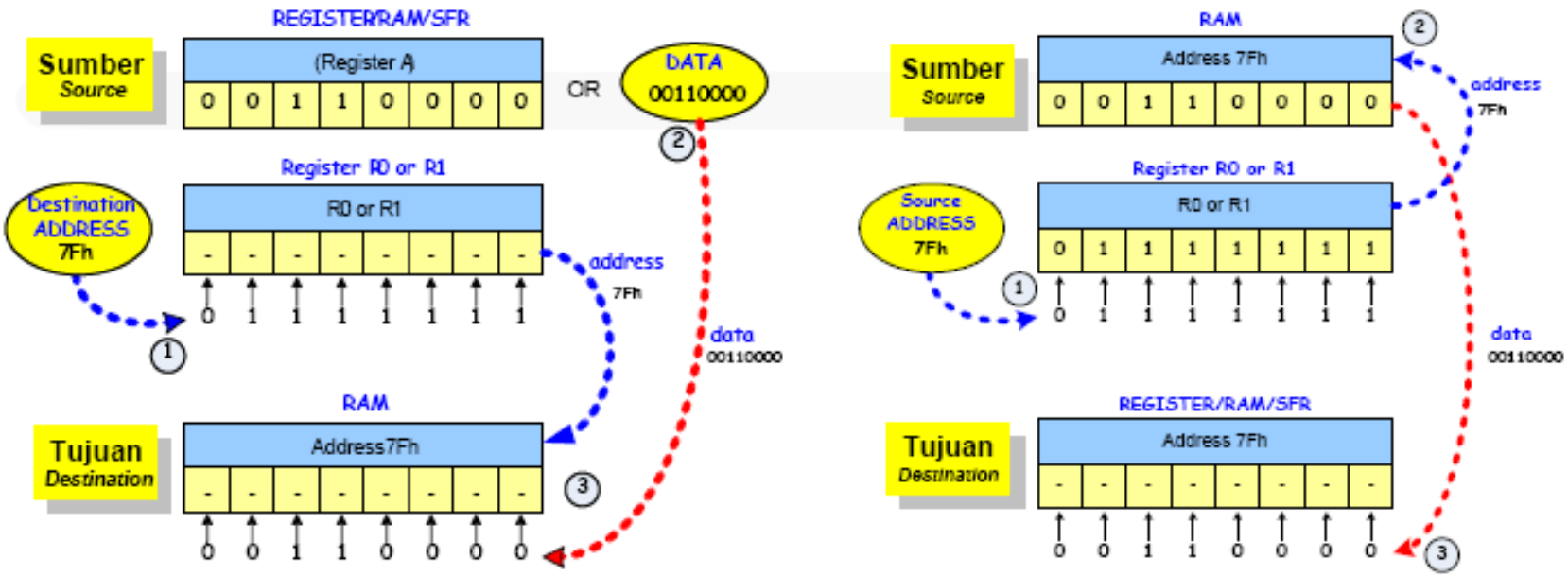
Direct Addressing Mode

Register Addressing : Data dari register/RAM/SFR sumber dipindahkan ke register/RAM/SFR tujuan

Contoh :

```
MOV     A,80h       ; Copy data from port 0 pins to A register
MOV     80h,A       ; Copy data from A register to the port 0
MOV     3Ah,#3Ah    ; Copy immediate data 3Ah to RAM location 3Ah
MOV     R0,12h      ; Copy data from RAM location 12h to R0 register
MOV     8Ch,R7      ; Copy data from R7 register to SFR timer 0 high byte
MOV     5Ch,A       ; Copy data from A register to RAM location 5Ch
MOV     0A8h,77h    ; Copy data from RAM location 77h to IE register
```

Indirect Addressing Mode



R0 atau R1 digunakan untuk menunjukkan Destination Address

```
MOV     @R0,#30h    ; Copy immediate data 30h to the address in R0
MOV     @R0,30h     ; Copy the content of address 30h to the address in R0
MOV     @R0,A       ; Copy the data in A to the address in R0
```

R0 atau R1 digunakan untuk menunjukkan Source Address

```
MOV     7Fh,@R0     ; Copy the content of the address in R0 to address 7Fh
MOV     A,@R0       ; Copy the content of the address in R0 to A
```

Ringkasan Organisasi Memori

- Mikrokontroler 89S51 Mempunyai 2 jenis memori, yaitu Memori Program (ROM) dan Memori Data (RAM)
- Memori Program besarnya 4 KByte menempati alamat 000H – FFFH
- Memori Data dibagi menjadi 3 blok:
  - 128 byte bawah (00H-7FH)
  - 128 byte atas (80H-FFH)
  - SFR (Special Function Register) (80H-FFH)

Bahasa Mesin Vs Assembly

Machine Language	Assembly Language
7D 25	MOV R5,#25H
7F 34	MOV R7,#34H
74 00	MOV A,#0
2D	ADD A,R5

Perangkat Lunak Yang Dibutuhkan Untuk Membuat Aplikasi Mikrokontroler AT89S51

- Kompiler ASM51
- Mengubah berkas Objek (.OBJ) ke Heksa (.HEX)
- Mengubah Berkas Heksa (.HEX) ke Binair (.BIN)
- Simulator / Emulator 8051 berfungsi untuk melakukan pengujian/simulasi dari program

Langkah-Langkah Membuat Aplikasi Mikrokontroler AT89S51

- Menulis Program Assembly ke editor teks (edit, notepad), kemudian simpan kode program dengan ekstensi “ *.ASM “
- Melakukan Kompilasi Program Dengan Cara:

  asm51 <NAMA_FILE.ASM>

- Jika terjadi kesalahan akan ditunjukkan dan harus diperbaiki. Kesalahan akan ditunjukkan dengan membuka file dengan ekstensi “ *.1ST “
- Bila tidak terjadi kesalahan akan ado file objek yang berekstensi “ *.OBJ “

Langkah-Langkah Membuat Aplikasi Mikrokontroler AT89S51

- File yang telah di kompile dan berhasil menghasilkan berkas “ *.OBJ “ di ubah ke format heksa (“ *.HEX “) dengan perintah

  OH <NAMA_FILE.OBJ>

- Program yang berekstensi “ *.HEX “ biasanya sudah bisa dimanfaatkan ke simulator/emulator.
- Kadang ada beberapa software yang membutuhkan format biner.

H <NAMA_FILE.HEX> <NAMA_FILE.BIN>

Pemrograman Assembly AT89S51

- Bahasa Assembly digunakan untuk menggantikan kode heksa dari bahasa mesin dengan “mnemonik” yang mudah diingat.

Pemrograman Assembly AT89S51

Bahasa Assembly, berisi:

- Instruksi-Instruksi Mesin : Mnemonik yang menyatakan instruksi, contoh → **MOV**
- Pengarah-pengarah assembler : Instruksi yang menyatakan struktur program, simbol-simbol data, konstantata, contoh → ORG
- Kontrol-Kontrol Assembler : Menentukan mode-mode Assembler, contoh → $TITLE
- Komentar : Ditulis agar program mudah dibaca, tidak harus per instruksi bisa sekumpulan instruksi

Format Assembly AT89S51

[label:] mnemonic [operan] [,operan] [...] [;komentar]

- **Label** : Mewakili alamat dari instruksi, biasanya digunakan sebagai operan pada instruksi percabangan. Label harus diawali dengan **huruf**, **tanda tanya** atau **garis bawah** kemudian diikuti dengan **huruf**, **angka**, **tanda tanya** atau **garis bawah** hingga **31 karakter**.
- ***Mnemonic*** : misalnya MOV, ADD, INC

Format Assembly AT89S51

[label:] mnemonic [operan] [,operan] [...] [;komentar]

- **Operan :** Bisa berupa alamat atau data yang berdasar pada kode mnemoniknya. Ada kode mnemonik yang tidak membutuhkan operan, misal **RET**
- **Komentar** : Diawali dengan tanda “ **;** ”

Simbol-Simbol Khusus Assembler

Assembler telah menyediakan beberapa simbol untuk menunjukkan register tertentu sebagai operand.

Contoh:

A	Akumulator
R0 …. R7	Register Serbaguna
DPTR	Data Pointer Register 16 Bit
PC	Program Counter
C	Carry Flag
B	Register B

Simbol-Simbol Khusus Assembler

Assembler telah menyediakan beberapa simbol untuk menunjukkan register tertentu sebagai operand.

Contoh:

A	Akumulator
R0 …. R7	Register Serbaguna
DPTR	Data Pointer Register 16 Bit
PC	Program Counter
C	Carry Flag
B	Register B

Pengalamatan Tak Langsung

Operand pengalamatan tak langsung menunjuk ke sebuah register yang berisi lokasi alamat memory yang akan digunakan dalam operasi.

Lokasi yang nyata tergantung pada isi register saat instruksi dijalankan.

Pengalamatan Tak Langsung

Untuk melaksanakan pengalamatan tak langsung digunakan simbol @

Contoh:

```
MOV     A,@R1
MOV     @R0,A
MOV     @R1,24H
```

Pengalamatan Tak Langsung

Pengalamatan tak langsung (*Indirect*) ini biasa digunakan untuk melakukan penulisan, pemindahan atau pembacaan beberapa data dalam lokasi memori yang mempunyai urutan beraturan.

Jika proses ini dilakukan dengan menggunakan pengalamatan langsung jumlah baris program yang diperlukan akan cukup panjang.

Pengalamatan Tak Langsung

Contohnya penulisan data 08H pada alamat 50H hingga 57H.

Listing 1:

```
ORG     0H
MOV     50H,#08H
MOV     51H,#08H
MOV     52H,#08H
MOV     53H,#08H
MOV     54H,#08H
MOV     55H,#08H
MOV     56H,#08H
MOV     57H,#08H
END
```

Pengalamatan Tak Langsung

Dengan digunakan sistem pengalamatan tak langsung, dapat diubah menjadi :

Listing 2:

```
        ORG   0H
        MOV   R0,#50H    ;

LOOP:   MOV   @R0,#08H
        INC   R0
        CJNE  R0,#58H,LOOP
        END
```

Pengalamatan Tak Langsung

Dalam listing program 2 diatas, R0 digunakan sebagai register yang menyimpan alamat dari data yang akan dituliskan. Dengan melakukan penambahan pada isi R0 dan mengulang perintah penulisan data ke alamat yang ditunjuk R0 hingga register ini menunjukkan nilai 57H + 1, atau 58H. Dengan demikian, barisan perintah pada Listing 1 dapat dieliminasi.

• Pengalamatan Tak Langsung

MCS-51 mempunyai sebuah register 16 bit (DPTR) dan dua buah register 8 bit (R0 dan R1) yang dapat digunakan untuk melakukan pengalamatan tidak langsung.

Contoh-contoh perintah yang menggunakan sistem pengalamatan tak langsung adalah :

```
MOV     @R0,A      ; R0 sebagai reg. penyimpan alamat
MOV     A,@R1      ; R1 sebagai reg. penyimpan alamat
ADD     A,@R0      ; R0 sebagai reg. penyimpan alamat
MOVX    @DPTR,A    ; DPTR sebagai reg. penyimpan alamat
MOVC    A,@A+DPTR  ; DPTR sebagai register
                     penyimpan alamat
```

Pengalamatan Langsung (Immediate Data)

Proses pengalamatan ini terjadi pada sebuah perintah ketika nilai operand merupakan data yang akan diproses.

Biasanya operand tersebut selalu diawali dengan tanda ‘#’ seperti pada contoh berikut.

```
MOV     A,#05H
MOV     A,#45H
MOV     B,#0E4H
MOV     DPTR,#4356H
```

Pengalamatan Langsung (Immediate Data)

Operand yang digunakan pada immediate data juga dapat berupa bilangan bertanda mulai - 256 hingga + 256.

Contoh :

```
MOV     A,# -1
```

sama dengan

```
MOV     A,#0FFH
```

Bilangan -1 adalah sama dengan bilangan 0 dikurangi 1, dalam bentuk heksa bilangan 00H. Jika dikurangi dengan 1, hasilnya adalah 0FFH. Dengan pengertian seperti ini, bilangan -1 dapat dianggap sama dengan 0FFH.

Pengalamatan Data

Proses pengalamatan ini terjadi pada sebuah perintah ketika nilai operand merupakan alamat dari data yang akan diisi, dipindahkan atau diproses.

Contoh :

```
MOV     P0,A
```

Port 0 adalah salah satu I/O pada MCS-51 yang mempunyai alamat 80H. Perintah pada contoh di atas selain mengirimkan data akumulator ke Port 0 juga merupakan perintah pemindahan data dari akumulator ke alamat 80H sehingga dapat juga dituliskan

```
MOV   80H,A.
```

Pengalamatan Bit

- Salah satu kelebihan dari mikrokontroler adalah bisa mengalamati per bit.
- Lokasi yang teralamati bit harus menyediakan suatu alamat bit dalam memori data internal (00H-7FH) dan SFR (80H-FFH)
- Cara Penulisannya ada tiga cara:
  - Menggunakan alamatnya langsung (SETB 0EH)
  - Menggunakan tanda titik antara alamat byte dan posisi bit (SETB ACC.7)
  - Menggunakan simbol baku (JNB TI,$)

JUMP dan CALL

- **ASM51** membolehkan kita untuk menggunakan mnemonik **JMP** atau **CALL** yang umum. Mnemonik **JMP** digunakan sebagai wakil dari **SJMP**, **AJMP** atau **LJMP**, sedangkan mnemonik **CALL** mewakili **ACALL** atau **LCALL**. Assembler akan mengkonversi mnemonik umum ini menjadi instruksi yang sesungguhnya mengikuti beberapa aturan sederhana.

JUMP dan CALL

- Diubah ke **SJMP** jika tidak ada dalam acuan alamat didepan (tujuan lompatan sebelum instruksi **JMP** yang bersangkutan) dan jangkauan (lompatan berada dalam 128 byte).
- Diubah ke bentuk **AJMP** jika tidak ada acuan lompatan didepan dan tujuan lompatan masih berada didalam blok 2K yang sama;
- Jika aturan 2 dan 3 tidak terpenuhi maka akan diubah ke bentuk **LJMP**.

Tidak selamanya konversi merupakan cara pemrograman yang baik.

JUMP dan CALL

- Misalnya tujuan lompatan ada beberapa didepan (setelah instruksi **JMP** yang bersangkutan) maka **JMP** yang umum tersebut akan diubah kebentuk **LJMP**, walau **SJMP** lebih cocok.

EKSPRESI-EKSPRESI ASSEMBLER (BASIS BILANGAN)

- Penulisan Bilangan akhir konstanta harus ditulis “B” untuk biner, “O” atau “Q” untuk oktaf, “D” atau tanpa simbol untuk desimal dan “H” untuk heksadesimal.
- Instruksi-instruksi berikut ini artinya sama:

```
MOV A,#15
MOV A,#1111B
MOV A,#0FH
MOV A,#17Q
MOV A,#15D
```

- Khusus untuk format heksa, jika digit awal adalah huruf (A,B,C,D,E atau F), penulisannya harus diawali “0” (nol)

EKSPRESI-EKSPRESI ASSEMBLER (STRING KARAKTER)

- Operan dapat berupa string yang terdiri dari satu atau karakter yang diapit tanda petik tunggal (‘). Kode ASCII dari karakter tersebut kemudian diterjemahkan sebagai bilangan biner yang sesuai dengan *Assembler*.

```
CJNE A, #’Q’, Lagi
SUBB A, #’O’ ; konversi digit ASCII ke digit biner
MOV DPTR, #’AB’ ; dua perintah
MOV DPTR, #4142 ; ini sama hasilnya
```

EKSPRESI-EKSPRESI ASSEMBLER (OPERATOR-OPERATOR ARITMETIK)

Operator-operator aritmetik meliputi:

+	penambahan
-	pengurangan
*	perkalian
/	pembagian
MOD	modulo, sisa pembagian

Misalnya, dua instruksi ini hasilnya sama:

```
MOV A, #10+10h
MOV A, #1Ah
```

EKSPRESI-EKSPRESI ASSEMBLER (OPERATOR-OPERATOR LOGIKA)

Operator logika meliputi

OR	logika OR
AND	logika AND
XOR	logika eksklusif OR
NOT	logika komplemen

Operasi logika tersebut masing-masing bit pada operator, misalnya, dua instruksi berikut hasilnya sama:

```
MOV A, #'9' AND 0Fh
MOV A,#9
```

EKSPRESI-EKSPRESI ASSEMBLER (OPERATOR-OPERATOR LOGIKA)

Operator NOT hanya membutuhkan satu operan.

Tiga instruksi MOV berikut ini, hasilnya sama:

```
TIGA        EQU 3
MIN_TIGA    EQU -3
MOV         A, #(NOT TIGA) + 1
MOV         A, #MIN_TIGA
MOV         A, #11111101B
```

EKSPRESI-EKSPRESI ASSEMBLER (OPERATOR-OPERATOR KHUSUS)

Operator-operator khusus meliputi:

SHR	geser kanan
SHL	geser kiri
HIGH	byte_tinggi (d7 s/d d4)
LOW	byte_rendah (d3 s/d d0)

Misalnya, dua instruksi berikut hasilnya sama:

```
MOV A, #8 SHL 1
MOV A, #10h
```

EKSPRESI-EKSPRESI ASSEMBLER (OPERATOR-OPERATOR KHUSUS)

Kedua instruksi berikut juga sama:

MOV A, #HIGH 1234h

MOV A, #12h

EKSPRESI-EKSPRESI ASSEMBLER (OPERATOR-OPERATOR RELASIONAL)

Jika suatu operator relasional digunakan antara dua operan, maka hasilnya selalu salah (0000h) atau benar (FFFFh). Operator-operator relasional ini meliputi:

EQ	=	Sama dengan
NE	<>	Tidak sama dengan
LT	<	Lebih kecil dari
LE	<=	Lebih kecil sama dengan
GT	>	Lebih besar dari
GE	>=	Lebih besar sama dengan

EKSPRESI-EKSPRESI ASSEMBLER (OPERATOR-OPERATOR RELASIONAL)

Contoh berikut menghasilkan benar (FFFFh):

```
MOV A, #5 = 5
MOV A, #5 NE 3
MOV A, #'X' LT 'Z'
MOV A, #'X' >= 'X'
MOV A, #$ > 0
MOV A, #100 GE 50
```

Dengan kata lain semua instruksi tersebut, sama dengan instruksi:

```
MOV A, #0FFh
```

EKSPRESI-EKSPRESI ASSEMBLER (PRIORITAS OPERATOR)

( )

HIGH LOW

* / MOD SHL SHR

+ -

EQ NE LT LE GT GE = <> < <= > >=

NOT

AND

OR XOR

EKSPRESI-EKSPRESI ASSEMBLER (PRIORITAS OPERATOR)

Jika lebih dari satu operator maka prioritas yang lebih tinggi didahulukan, jika prioritasnya sama maka akan dievaluasi dari kiri ke kanan, misalnya:

Ekspresi	Nilai
HIGH (‘A’ SHL 8)	0041h
HIGH ‘A’ SHL 8	0000h
NOT ‘A’ -1	FFBFh
‘A’ OR ‘A’ SHL 8	4141h

MIKROKONTROLER

Organisasi Memori

Ringkasan Organisasi Memori

- Mikrokontroler 89S51 Mempunyai 2 jenis memori, yaitu Memori Program (ROM) dan Memori Data (RAM)
- Memori Program besarnya 4 KByte menempati alamat 000H – FFFH
- Memori Data dibagi menjadi 3 blok:
  - 128 byte bawah (00H-7FH)
  - 128 byte atas (80H-FFH)
  - SFR (Special Function Register) (80H-FFH)

Bahasa Mesin Vs Assembly

Machine Language	Assembly Language
7D 25	MOV R5,#25H
7F 34	MOV R7,#34H
74 00	MOV A,#0
2D	ADD A,R5

Perangkat Lunak Yang Dibutuhkan Untuk Membuat Aplikasi Mikrokontroler AT89S51

- Kompiler ASM51
- Mengubah berkas Objek (.OBJ) ke Heksa (.HEX)
- Mengubah Berkas Heksa (.HEX) ke Binair (.BIN)
- Simulator / Emulator 8051 berfungsi untuk melakukan pengujian/simulasi dari program

Langkah-Langkah Membuat Aplikasi Mikrokontroler AT89S51

- Menulis Program Assembly ke editor teks (edit, notepad), kemudian simpan kode program dengan ekstensi “ *.ASM “
- Melakukan Kompilasi Program Dengan Cara:

  asm51 <NAMA_FILE.ASM>

- Jika terjadi kesalahan akan ditunjukkan dan harus diperbaiki. Kesalahan akan ditunjukkan dengan membuka file dengan ekstensi “ *.1ST “
- Bila tidak terjadi kesalahan akan ado file objek yang berekstensi “ *.OBJ “

Langkah-Langkah Membuat Aplikasi Mikrokontroler AT89S51

- File yang telah di kompile dan berhasil menghasilkan berkas “ *.OBJ “ di ubah ke format heksa (“ *.HEX “) dengan perintah

  OH <NAMA_FILE.OBJ>

- Program yang berekstensi “ *.HEX “ biasanya sudah bisa dimanfaatkan ke simulator/emulator.
- Kadang ada beberapa software yang membutuhkan format biner.

H <NAMA_FILE.HEX> <NAMA_FILE.BIN>

Pemrograman Assembly AT89S51

- Bahasa Assembly digunakan untuk menggantikan kode heksa dari bahasa mesin dengan “mnemonik” yang mudah diingat.

Pemrograman Assembly AT89S51

Bahasa Assembly, berisi:

- Instruksi-Instruksi Mesin : Mnemonik yang menyatakan instruksi, contoh → **MOV**
- Pengarah-pengarah assembler : Instruksi yang menyatakan struktur program, simbol-simbol data, konstantata, contoh → ORG
- Kontrol-Kontrol Assembler : Menentukan mode-mode Assembler, contoh → $TITLE
- Komentar : Ditulis agar program mudah dibaca, tidak harus per instruksi bisa sekumpulan instruksi

Format Assembly AT89S51

[label:] mnemonic [operan] [,operan] [...] [;komentar]

- **Label** : Mewakili alamat dari instruksi, biasanya digunakan sebagai operan pada instruksi percabangan. Label harus diawali dengan **huruf**, **tanda tanya** atau **garis bawah** kemudian diikuti dengan **huruf**, **angka**, **tanda tanya** atau **garis bawah** hingga **31 karakter**.
- ***Mnemonic*** : misalnya MOV, ADD, INC

Format Assembly AT89S51

[label:] mnemonic [operan] [,operan] [...] [;komentar]

- **Operan :** Bisa berupa alamat atau data yang berdasar pada kode mnemoniknya. Ada kode mnemonik yang tidak membutuhkan operan, misal **RET**
- **Komentar** : Diawali dengan tanda “ **;** ”

Simbol-Simbol Khusus Assembler

Assembler telah menyediakan beberapa simbol untuk menunjukkan register tertentu sebagai operand.

Contoh:

A	Akumulator
R0 …. R7	Register Serbaguna
DPTR	Data Pointer Register 16 Bit
PC	Program Counter
C	Carry Flag
B	Register B

Simbol-Simbol Khusus Assembler

Assembler telah menyediakan beberapa simbol untuk menunjukkan register tertentu sebagai operand.

Contoh:

A	Akumulator
R0 …. R7	Register Serbaguna
DPTR	Data Pointer Register 16 Bit
PC	Program Counter
C	Carry Flag
B	Register B

Pengalamatan Tak Langsung

Operand pengalamatan tak langsung menunjuk ke sebuah register yang berisi lokasi alamat memory yang akan digunakan dalam operasi.

Lokasi yang nyata tergantung pada isi register saat instruksi dijalankan.

Pengalamatan Tak Langsung

Untuk melaksanakan pengalamatan tak langsung digunakan simbol @

Contoh:

```
MOV     A,@R1
MOV     @R0,A
MOV     @R1,24H
```

Pengalamatan Tak Langsung

Pengalamatan tak langsung (*Indirect*) ini biasa digunakan untuk melakukan penulisan, pemindahan atau pembacaan beberapa data dalam lokasi memori yang mempunyai urutan beraturan.

Jika proses ini dilakukan dengan menggunakan pengalamatan langsung jumlah baris program yang diperlukan akan cukup panjang.

Pengalamatan Tak Langsung

Contohnya penulisan data 08H pada alamat 50H hingga 57H.

Listing 1:

```
ORG     0H
MOV     50H,#08H
MOV     51H,#08H
MOV     52H,#08H
MOV     53H,#08H
MOV     54H,#08H
MOV     55H,#08H
MOV     56H,#08H
MOV     57H,#08H
END
```

Pengalamatan Tak Langsung

Dengan digunakan sistem pengalamatan tak langsung, dapat diubah menjadi :

Listing 2:

```
        ORG   0H
        MOV   R0,#50H    ;

LOOP:   MOV   @R0,#08H
        INC   R0
        CJNE  R0,#58H,LOOP
        END
```

Pengalamatan Tak Langsung

Dalam listing program 2 diatas, R0 digunakan sebagai register yang menyimpan alamat dari data yang akan dituliskan. Dengan melakukan penambahan pada isi R0 dan mengulang perintah penulisan data ke alamat yang ditunjuk R0 hingga register ini menunjukkan nilai 57H + 1, atau 58H. Dengan demikian, barisan perintah pada Listing 1 dapat dieliminasi.

• Pengalamatan Tak Langsung

MCS-51 mempunyai sebuah register 16 bit (DPTR) dan dua buah register 8 bit (R0 dan R1) yang dapat digunakan untuk melakukan pengalamatan tidak langsung.

Contoh-contoh perintah yang menggunakan sistem pengalamatan tak langsung adalah :

```
MOV     @R0,A      ; R0 sebagai reg. penyimpan alamat
MOV     A,@R1      ; R1 sebagai reg. penyimpan alamat
ADD     A,@R0      ; R0 sebagai reg. penyimpan alamat
MOVX    @DPTR,A    ; DPTR sebagai reg. penyimpan alamat
MOVC    A,@A+DPTR  ; DPTR sebagai register
                     penyimpan alamat
```

Pengalamatan Langsung (Immediate Data)

Proses pengalamatan ini terjadi pada sebuah perintah ketika nilai operand merupakan data yang akan diproses.

Biasanya operand tersebut selalu diawali dengan tanda ‘#’ seperti pada contoh berikut.

```
MOV     A,#05H
MOV     A,#45H
MOV     B,#0E4H
MOV     DPTR,#4356H
```

Pengalamatan Langsung (Immediate Data)

Operand yang digunakan pada immediate data juga dapat berupa bilangan bertanda mulai - 256 hingga + 256.

Contoh :

```
MOV     A,# -1
```

sama dengan

```
MOV     A,#0FFH
```

Bilangan -1 adalah sama dengan bilangan 0 dikurangi 1, dalam bentuk heksa bilangan 00H. Jika dikurangi dengan 1, hasilnya adalah 0FFH. Dengan pengertian seperti ini, bilangan -1 dapat dianggap sama dengan 0FFH.

Pengalamatan Data

Proses pengalamatan ini terjadi pada sebuah perintah ketika nilai operand merupakan alamat dari data yang akan diisi, dipindahkan atau diproses.

Contoh :

```
MOV     P0,A
```

Port 0 adalah salah satu I/O pada MCS-51 yang mempunyai alamat 80H. Perintah pada contoh di atas selain mengirimkan data akumulator ke Port 0 juga merupakan perintah pemindahan data dari akumulator ke alamat 80H sehingga dapat juga dituliskan

```
MOV   80H,A.
```

Pengalamatan Bit

- Salah satu kelebihan dari mikrokontroler adalah bisa mengalamati per bit.
- Lokasi yang teralamati bit harus menyediakan suatu alamat bit dalam memori data internal (00H-7FH) dan SFR (80H-FFH)
- Cara Penulisannya ada tiga cara:
  - Menggunakan alamatnya langsung (SETB 0EH)
  - Menggunakan tanda titik antara alamat byte dan posisi bit (SETB ACC.7)
  - Menggunakan simbol baku (JNB TI,$)

JUMP dan CALL

- **ASM51** membolehkan kita untuk menggunakan mnemonik **JMP** atau **CALL** yang umum. Mnemonik **JMP** digunakan sebagai wakil dari **SJMP**, **AJMP** atau **LJMP**, sedangkan mnemonik **CALL** mewakili **ACALL** atau **LCALL**. Assembler akan mengkonversi mnemonik umum ini menjadi instruksi yang sesungguhnya mengikuti beberapa aturan sederhana.

JUMP dan CALL

- Diubah ke **SJMP** jika tidak ada dalam acuan alamat didepan (tujuan lompatan sebelum instruksi **JMP** yang bersangkutan) dan jangkauan (lompatan berada dalam 128 byte).
- Diubah ke bentuk **AJMP** jika tidak ada acuan lompatan didepan dan tujuan lompatan masih berada didalam blok 2K yang sama;
- Jika aturan 2 dan 3 tidak terpenuhi maka akan diubah ke bentuk **LJMP**.

Tidak selamanya konversi merupakan cara pemrograman yang baik.

JUMP dan CALL

- Misalnya tujuan lompatan ada beberapa didepan (setelah instruksi **JMP** yang bersangkutan) maka **JMP** yang umum tersebut akan diubah kebentuk **LJMP**, walau **SJMP** lebih cocok.

EKSPRESI-EKSPRESI ASSEMBLER (BASIS BILANGAN)

- Penulisan Bilangan akhir konstanta harus ditulis “B” untuk biner, “O” atau “Q” untuk oktaf, “D” atau tanpa simbol untuk desimal dan “H” untuk heksadesimal.
- Instruksi-instruksi berikut ini artinya sama:

```
MOV A,#15
MOV A,#1111B
MOV A,#0FH
MOV A,#17Q
MOV A,#15D
```

- Khusus untuk format heksa, jika digit awal adalah huruf (A,B,C,D,E atau F), penulisannya harus diawali “0” (nol)

EKSPRESI-EKSPRESI ASSEMBLER (STRING KARAKTER)

- Operan dapat berupa string yang terdiri dari satu atau karakter yang diapit tanda petik tunggal (‘). Kode ASCII dari karakter tersebut kemudian diterjemahkan sebagai bilangan biner yang sesuai dengan *Assembler*.

```
CJNE A, #’Q’, Lagi
SUBB A, #’O’ ; konversi digit ASCII ke digit biner
MOV DPTR, #’AB’ ; dua perintah
MOV DPTR, #4142 ; ini sama hasilnya
```

EKSPRESI-EKSPRESI ASSEMBLER (OPERATOR-OPERATOR ARITMETIK)

Operator-operator aritmetik meliputi:

+	penambahan
-	pengurangan
*	perkalian
/	pembagian
MOD	modulo, sisa pembagian

Misalnya, dua instruksi ini hasilnya sama:

```
MOV A, #10+10h
MOV A, #1Ah
```

EKSPRESI-EKSPRESI ASSEMBLER (OPERATOR-OPERATOR LOGIKA)

Operator logika meliputi

OR	logika OR
AND	logika AND
XOR	logika eksklusif OR
NOT	logika komplemen

Operasi logika tersebut masing-masing bit pada operator, misalnya, dua instruksi berikut hasilnya sama:

```
MOV A, #'9' AND 0Fh
MOV A,#9
```

EKSPRESI-EKSPRESI ASSEMBLER (OPERATOR-OPERATOR LOGIKA)

Operator NOT hanya membutuhkan satu operan.

Tiga instruksi MOV berikut ini, hasilnya sama:

```
TIGA        EQU 3
MIN_TIGA    EQU -3
MOV         A, #(NOT TIGA) + 1
MOV         A, #MIN_TIGA
MOV         A, #11111101B
```

EKSPRESI-EKSPRESI ASSEMBLER (OPERATOR-OPERATOR KHUSUS)

Operator-operator khusus meliputi:

SHR	geser kanan
SHL	geser kiri
HIGH	byte_tinggi (d7 s/d d4)
LOW	byte_rendah (d3 s/d d0)

Misalnya, dua instruksi berikut hasilnya sama:

```
MOV A, #8 SHL 1
MOV A, #10h
```

EKSPRESI-EKSPRESI ASSEMBLER (OPERATOR-OPERATOR KHUSUS)

Kedua instruksi berikut juga sama:

MOV A, #HIGH 1234h

MOV A, #12h

EKSPRESI-EKSPRESI ASSEMBLER (OPERATOR-OPERATOR RELASIONAL)

Jika suatu operator relasional digunakan antara dua operan, maka hasilnya selalu salah (0000h) atau benar (FFFFh). Operator-operator relasional ini meliputi:

EQ	=	Sama dengan
NE	<>	Tidak sama dengan
LT	<	Lebih kecil dari
LE	<=	Lebih kecil sama dengan
GT	>	Lebih besar dari
GE	>=	Lebih besar sama dengan

EKSPRESI-EKSPRESI ASSEMBLER (OPERATOR-OPERATOR RELASIONAL)

Contoh berikut menghasilkan benar (FFFFh):

```
MOV A, #5 = 5
MOV A, #5 NE 3
MOV A, #'X' LT 'Z'
MOV A, #'X' >= 'X'
MOV A, #$ > 0
MOV A, #100 GE 50
```

Dengan kata lain semua instruksi tersebut, sama dengan instruksi:

```
MOV A, #0FFh
```

EKSPRESI-EKSPRESI ASSEMBLER (PRIORITAS OPERATOR)

( )

HIGH LOW

* / MOD SHL SHR

+ -

EQ NE LT LE GT GE = <> < <= > >=

NOT

AND

OR XOR

EKSPRESI-EKSPRESI ASSEMBLER (PRIORITAS OPERATOR)

Jika lebih dari satu operator maka prioritas yang lebih tinggi didahulukan, jika prioritasnya sama maka akan dievaluasi dari kiri ke kanan, misalnya:

Ekspresi	Nilai
HIGH (‘A’ SHL 8)	0041h
HIGH ‘A’ SHL 8	0000h
NOT ‘A’ -1	FFBFh
‘A’ OR ‘A’ SHL 8	4141h

MIKROKONTROLER

INSTRUKSI-INSTRUKSI
BAHASA ASSEMBLY 9051

Ringkasan Format Bahasa Assembly

[label:] mnemonic [operan] [,operan] [...] [;komentar]

Operasi Aritmatika (INCREMENT dan DECREMENT)

Increment : Tambah satu isi register

Decrement : Turunkan satu isi register

Mnemonic	**Operation**	
INC	A	Tambah satu isi register A
INC	Rr	Tambah satu isi register Rr
INC	add	Tambah satu isi direct address
INC	@Rp	Tambah satu isi address dalam Rp
INC	DPTR	Tambah satu isi register 16 bit DPTR
DEC	A	Kurangi satu isi register A
DEC	Rr	Kurangi satu isi register Rr
DEC	add	Kurangi satu isi dirrect address
DEC	@Rp	Kurangi satu isi address dalam Rp

Operasi Aritmatika (INCREMENT dan DECREMENT)

Contoh

```
MOV     A,#3Ah          ; A = 3Ah
DEC     A               ; A = 39h
MOV     R0,#15h         ; R0 = 15h
MOV     15h,#12h        ; Internal RAM 15h = 12h
INC     @R0             ; Internal RAM 15h = 13h
DEC     15h             ; Internal RAM 15h = 12h
INC     R0              ; R0 = 16h
MOV     16h,A           ; Internal RAM 16h = 39h
INC     @R0             ; Internal RAM 16h = 3Ah
MOV     DPTR,#12FFh     ; DPTR = 12FFh
INC     DPTR            ; DPTR = 1300h
DEC     83h             ; DPTR = 1200h(SFR 83h adalah byte DPH
```

Operasi Aritmatika (PENJUMLAHAN)

Mnemonic Operation

ADD A,#n ;Tambahkan A dengan angka n, simpan hasilnya di A

ADD A,Rr ;Tambahkan A dengan register Rr, simpan hasilnya di A

ADD A,add ;Tambahkan A dengan isi address, simpan hasilnya di A

ADD A,@Rp ;Tambahkan A dengan isi address dalam Rp, simpan hasilnya di A

ADDC A,#n ;Tambahkan A, angka n dan Carry, simpan hasilnya di A.

ADDC A,Rr ;Tambahkan A, isi register Rr dan Carry, simpan hasil di A

ADDC A,add ;Tambahkan A, isi address dan Carry, simpan hasil di A

ADDC A,@Rp ;Tambahkan A, isi address dalam Rp dan Carry, simpan hasilnya di A.

Operasi Aritmatika (PENJUMLAHAN)

Catatan :

- Carry flag (C) akan 1 jika terdapat carry pada bit ke-7.
- Auxilliary Carry flag (AC) akan 1 jika terdapat carry pada bit ke-3.
- Over Flow flag (OV) akan 1 jika terdapat carry pada bit ke-7, tapi tidak terdapat carry pada bit ke-6 atau
- terdapat carry pada bit ke-6 tetapi tidak pada bit ke-7, dimana dapat dieksperikan dengan operasi logika sbb :
- OV = C7 XOR C6

Penjumlahan tak bertanda dan bertanda

- *Unsigned and Signed Addition*
- Unsigned number : 0 s/d 255d atau 00000000b s/d 11111111b
- Signed number : -128d s/d +127d atau 1000000b s/d 01111111b
- Penjumlahan unsigned number dapat menghasilkan carry flag jika hasil penjumlahan melebihi FFh, atau
- borrow flag jika angka pengurang lebih besar dari yang dikurangi.

Operasi Aritmatika (PENJUMLAHAN)

Catatan :

- Carry flag (C) akan 1 jika terdapat carry pada bit ke-7.
- Auxilliary Carry flag (AC) akan 1 jika terdapat carry pada bit ke-3.
- Over Flow flag (OV) akan 1 jika terdapat carry pada bit ke-7, tapi tidak terdapat carry pada bit ke-6 atau
- terdapat carry pada bit ke-6 tetapi tidak pada bit ke-7, dimana dapat dieksperikan dengan operasi logika sbb :
- OV = C7 XOR C6

Penjumlahan tak bertanda dan bertanda

- *Unsigned and Signed Addition*
- Unsigned number : 0 s/d 255d atau 00000000b s/d 11111111b
- Signed number : -128d s/d +127d atau 1000000b s/d 01111111b
- Penjumlahan unsigned number dapat menghasilkan carry flag jika hasil penjumlahan melebihi FFh, atau
- borrow flag jika angka pengurang lebih besar dari yang dikurangi.

Operasi Aritmatika (PENJUMLAHAN)

Penjumlahan Tak Bertanda / *Unsigned Addition*

Carry flag dapat digunakan untuk mendeteksi hasil penjumlahan yang melebihi FFh. Jika carry = 1 setelah

penjumlahan, maka carry tersebut dapat ditambahkan ke high byte sehingga hasil penjumlahan tidak hilang.

Misalnya :

95d = 01011111b

189d = 10111101b

284d 1 00011100b

C=1 dapat ditambahkan ke byte berikutnya (high byte)

Operasi Aritmatika (PENJUMLAHAN)

Penjumlahan Bertanda / *Signed Addition*

Hasil penjumlahan bertanda tidak boleh melebihi -128d atau +127d. Aturan ini tidak menjadi masalah ketika angka yang dijumlahkan positif dan negatif, misalnya :

- 001d = 11111111b

+027d = 00011011b

+026d = 00011010b = + 026d

Dari penjumlahan diatas terdapat carry dari bit ke-7, maka C=1. Pada bit ke-6 juga terdapat carry, maka OV=0.

Pada penjumlahan ini tidak perlu manipulasi apapun karena hasil penjumlahannya sudah benar.

Jika kedua angka yang dijumlahkan adalah positif, maka ada kemungkinan hasil penjumlahan melebihi +127d,

misalnya :

+100d = 01100100b

+050d = 00110010b

+150d = 10010110b = - 106d

Ada kelebihan 22d dari batas +127d. Tidak ada carry dari bit ke-7, maka C=0, ada carry dari bit ke-6, maka

OV=1.

Operasi Aritmatika (PENJUMLAHAN)

Contoh penjumlahan dua angka positif yang tidak melebihi +127d adalah :

+045d = 00101101b

+075d = 01001011b

+120d = 01111000b = + 120d

Dari penjumlahan diatas tidak terdapat carry dari bit ke-7 maupun bit ke-6, maka C=0 dan OV=0.

Operasi Aritmatika (PENJUMLAHAN)

Penjumlahan dua angka negatif yang tidak melebihi -128d adalah sbb :

- 030d = 11100010b
- 050d = 11001110b
- 080d = 10110000b = - 080d

Terdapat carry dari bit ke-7, maka C=1, ada carry dari bit ke-6, maka OV=0.

Operasi Aritmatika (PENJUMLAHAN)

Penjumlahan dua angka negatif yang hasilnya melebihi -128d adalah sbb :

- 070d = 10111010b
- 070d = 10111010b
- 140d = 01110100b = +116d *(Komplemen 116d = 139d)*

Ada kelebihan -12d. Ada carry dari bit ke-7, maka C=1, tidak ada carry dari bit ke-6, maka OV=1.

Operasi Aritmatika (PENJUMLAHAN)

- Dari semua contoh diatas, manipulasi program perlu dilakukan sbb :

FLAGS		ACTION
C	OV	
0	0	None
0	1	Complement the sign
1	0	None
1	1	Complement the sign

Operasi Aritmatika (Pengurangan)

Mnemonic		Operation
SUBB	A,#n	Kurangi A dengan angka n dan C flag, simpan hasilnya di A
SUBB	A,Rr	Kurangi A dengan register Rr dan C flag, simpan hasilnya di A
SUBB	A,add	Kurangi A dengan isi address dan C flag, simpan hasilnya di A
SUBB	A,@Rp	Kurangi A dengan isi address dalam Rp dan C flag, simpan hasilnya di A

Catatan :
Carry flag (C) akan 1 jika terdapat borrow pada bit ke-7.
Auxilliary Carry flag (AC) akan 1 jika terdapat borrow pada bit ke-3.
Over Flow flag (OV) akan 1 jika terdapat borrow pada bit ke-7, tapi tidak terdapat borrow pada bit ke-6 atau terdapat borrow pada bit ke-6 tetapi tidak pada bit ke-7, dimana dapat dieksperikan dengan operasi logika sbb :

OV = C7 XOR C6

Pengurangan tak bertanda dan bertanda
Unsigned and Signed Substraction

Unsigned number : Positif 255d (C=0, A=FFh) s/d Negatif 255d (C=1, A=01h)
Signed number : -128d s/d +127d atau 1000000b s/d 01111111b

Oleh karena carry flag selalu disertakan dalam pengurangan, maka carry flag harus di set 0 agar tidak mempengaruhi pengurangan.
Pengurangan unsigned number dapat menghasilkan borrow jika angka pengurang lebih besar dari yang dikurangi.

Pengurangan

• **Pengurangan Tak Bertanda / *Unsigned Substraction***

Berikut adalah contoh pengurangan dimana pengurang lebih besar dari yang dikurangi :

```
        015d      = 00001111b
SUBB    100d      = 01100100b
      - 085d      1 10101011b = 171d
```

C=1 dan OV=0. Komplemen 171d adalah 85d

Contoh lain :

```
        100d      = 01100100b
SUBB    015d      = 00001111b
        085d      0 01010101b = 85d
```

C=0 dan OV=0. Hasilnya sudah benar, jadi tidak perlu dikomplemenkan.

•**Pengurangan Bertanda / *Signed Substraction***

Hasil pengurangan bertanda tidak boleh melebihi -128d atau +127d.

Contoh 1 :

```
        +100d     = 01100100b    (Carry flag = 0 sebelum SUBB)
SUBB    +126d     = 01111110b
        - 026d    1 11100110b = - 026d
```

Dari pengurangan diatas terdapat borrow dari bit ke-7 dan ke-6, maka C=1 dan OV=0.Pada pengurangan ini tidak perlu manipulasi apapun karena hasil pengurangannya sudah benar.

Pengurangan

Contoh 2:

```
        - 061d      = 11000011b   (Carry flag = 0 sebelum SUBB)
SUBB    - 116d      = 10001100b
          +055d       00110111b = +55d
```

Tidak ada borrow dari bit ke-7 atau bit ke-6, maka C=0 dan OV=0.

Contoh 3 :

```
        - 099d      = 10011101b   (Carry flag = 0 sebelum SUBB)
SUBB    +100d       = 01100100b
          - 199d      00111001b = +057d    (Komplemen 57d = 198d)
```

Tidak ada borrow dari bit ke-7 tapi ada borrow dari bit ke-6, maka C=0 dan OV=1. Hasilnya perlu dimanipulasi.

Contoh 4 :

```
        +087d       = 01010111b   (Carry flag = 0 sebelum SUBB)
SUBB    - 052d      = 11001100b
          +139d       10001011b = -117d    (Komplemen 117d = 138d)
```

Ada borrow dari bit ke-7 dan tidak ada borrow dari bit ke-6, maka C=1 dan OV=1. Hasilnya perlu dimanipulasi.

Catatan : **Jika OV=1 maka komplemenkan hasilnya.**

Operasi Aritmatika (Perkalian)

Mnemonic		Operation
MUL	AB	Kalikan isi A dengan isi B, simpan lower byte di A dan high byte di B.

Catatan :
Over Flow flag (OV) akan 1 jika A X B > FFh.
Carry flag (C) selalu 0.
Nilai maksimum hasil perkalian antara A dan B adalah FE01h, jika A dan B berisi FFh.

Contoh :

```
MOV     A,#7Bh    ; A=7Bh
MOV     B,#02h    ; B=02h
MUL     AB        ; A=F6h dan B=00h; OV=0
MOV     A,#0FEh   ; A=FEh
MUL     AB        ; A=00h dan B=00h; OV=0
```

Operasi Aritmatika (Pembagian)

Mnemonic		Operation
DIV	AB	Bagi isi A dengan isi B, simpan hasilnya di A dan simpan sisanya di B.

Catatan :
Over Flow flag (OV) akan 1 jika terjadi pembagian dengan 0.
Carry flag (C) selalu 0.

Contoh :

```
MOV     A,#0FFh   ; A=FFh
MOV     B,#2Ch    ; B=2Ch
DIV     AB        ; A=05h dan B=23h
DIV     AB        ; A=00h dan B=05h
DIV     AB        ; A=00h dan B=00h
DIV     AB        ; A=?? Dan B=??, OV=1
```

Operasi Logika (Logika Boolean)

BOOLEAN OPERATOR	8051 MNEMONIC
AND	ANL (AND logical)
OR	ORL (OR logical)
NOT	CPL (Complement)
XOR	XRL (Exclusive OR logical)

AND

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

Contoh :

```
00001111
11110000  ANL
00000000
```

OR

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

Contoh :

```
00001111
11111111  ORL
11111111
```

XOR

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

Contoh :

```
11000011
11001100  XRL
00001111
```

NOT

A	/A
0	1
1	0

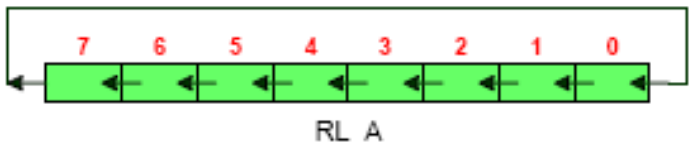
Contoh :

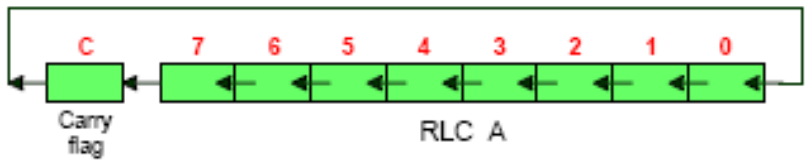
CPL 00001111

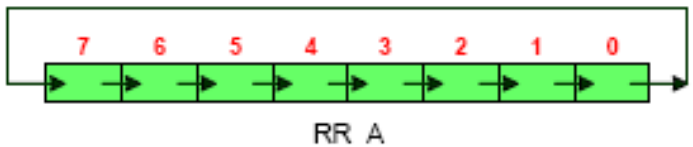
11110000

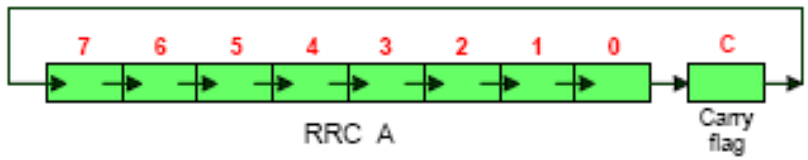
Operasi Logika (Rotate & Swap)

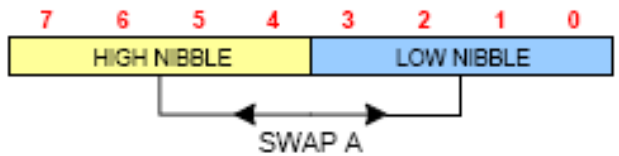
RL	**Rotate Left** : Geser isi register ke kiri satu bit.
RR	**Rotate Right** : Geser isi register ke kanan satu bit.
RLC	**Rotate Left and Carry** : Geser isi register dan carry flag ke kiri satu bit.
RRC	**Rotate Right and Carry** : Geser isi register dan carry flag ke kanan satu bit.
SWAP	**Swap** : Saling tukarkan nibble register, low nibble menjadi high nibble, dan sebaliknya.











Contoh Operasi Logika

- Contoh 1. Logika Boolean :

```
MOV   A,#0FFh   ; A = FFh
MOV   R0,#77h   ; R0 = 77h
ANL   A,R0      ; A = 77h
MOV   15h,A     ; 15h = 77h
CPL   A         ; A = 88h
ORL   15h,#88h  ; 15h = FFh
XRL   A,15h     ; A = 77h
XRL   A,R0      ; A = 00h
ANL   A,15h     ; A = 00h
ORL   A,R0      ; A = 77h
CLR   A         ; A = 00h
XRL   15h,A     ; 15h = FFh
XRL   A,R0      ; A = 77h
```

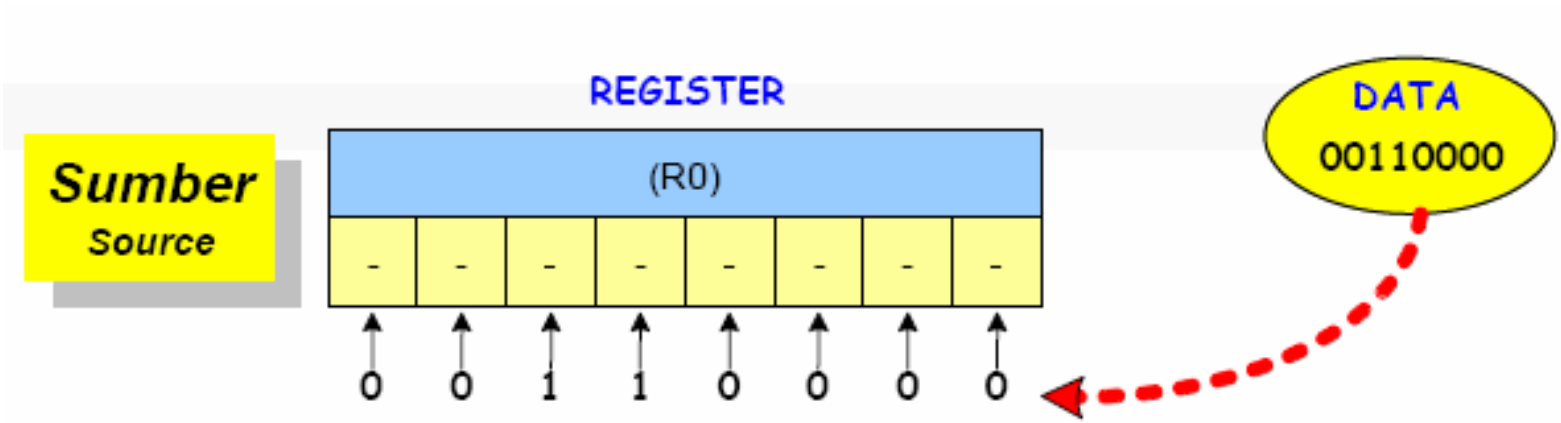
- Contoh 2. Rotate & Swap :

```
MOV   A,#0A5h   ; A = 10100101b = A5h
RR    A         ; A = 11010010b = D2h
RR    A         ; A = 01101001b = 69h
RR    A         ; A = 10110100b = B4h
RR    A         ; A = 01011010b = 5Ah
SWAP  A         ; A = 10100101b = A5h
CLR   C         ; C = 0, A =10100101b = A5h
RRC   A         ; C = 1, A = 01010010b = 52h
RRC   A         ; C = 0, A = 10101001b = A9h
RL    A         ; A = 01010011b = 53h
RL    A         ; A = 10100110b = A6h
SWAP  A         ; C = 0, A = 01101010b = 6Ah
RLC   A         ; C = 0, A = 11010100b = D4h
RLC   A         ; C = 1, A = 10101000b = A8h
SWAP  A         ; C = 1, A = 10001010b = 8Ah
```

MIKROKONTROLER

INSTRUKSI-INSTRUKSI
BAHASA ASSEMBLY 89051

Immediate Adressing Mode



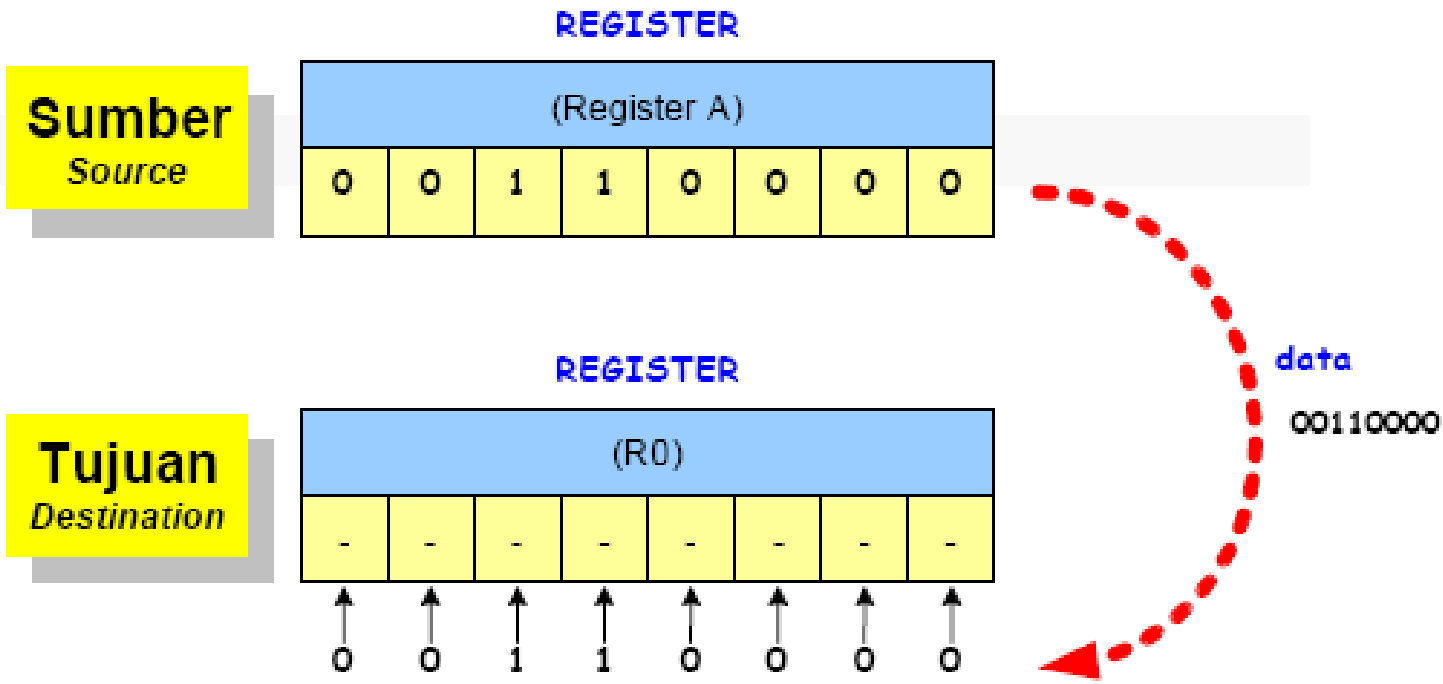
Immediate Addressing Mode

Immediate Addressing : Data langsung dipindahkan ke register
Contoh :

```
MOV     A,#30h          ; Copy the immediate data 30h to A register
MOV     R0,#00110000B   ; Copy the immediate data 30h to R0 register
MOV     DPTR,#48        ; Copy the immediate data 30h to DPTR register
```

Catatan :
30hexa = **00110000**biner = **48**desimal

Register Addressing Mode



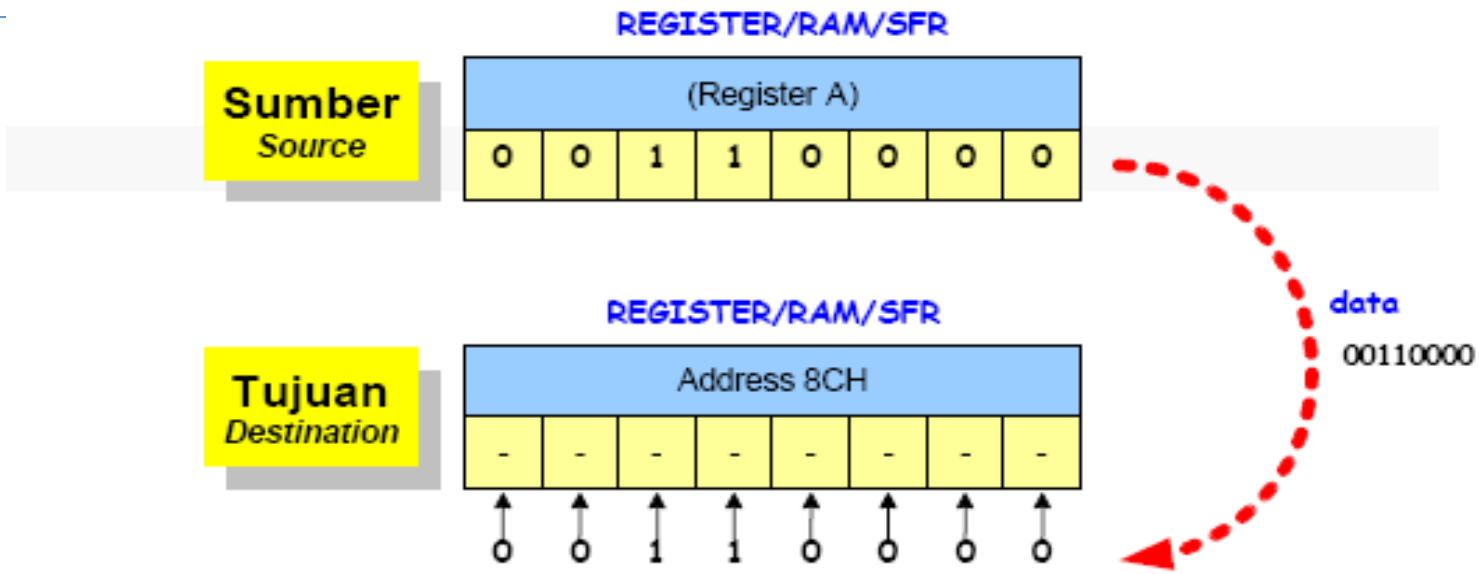
Register Addressing Mode

Register Addressing : Data dari register sumber dipindahkan ke register tujuan

Contoh :

```
MOV     R0,A        ; Copy data from A register to R0 register
MOV     A,R7        ; Copy data from R7register to A register
```

Direct Addressing Mode



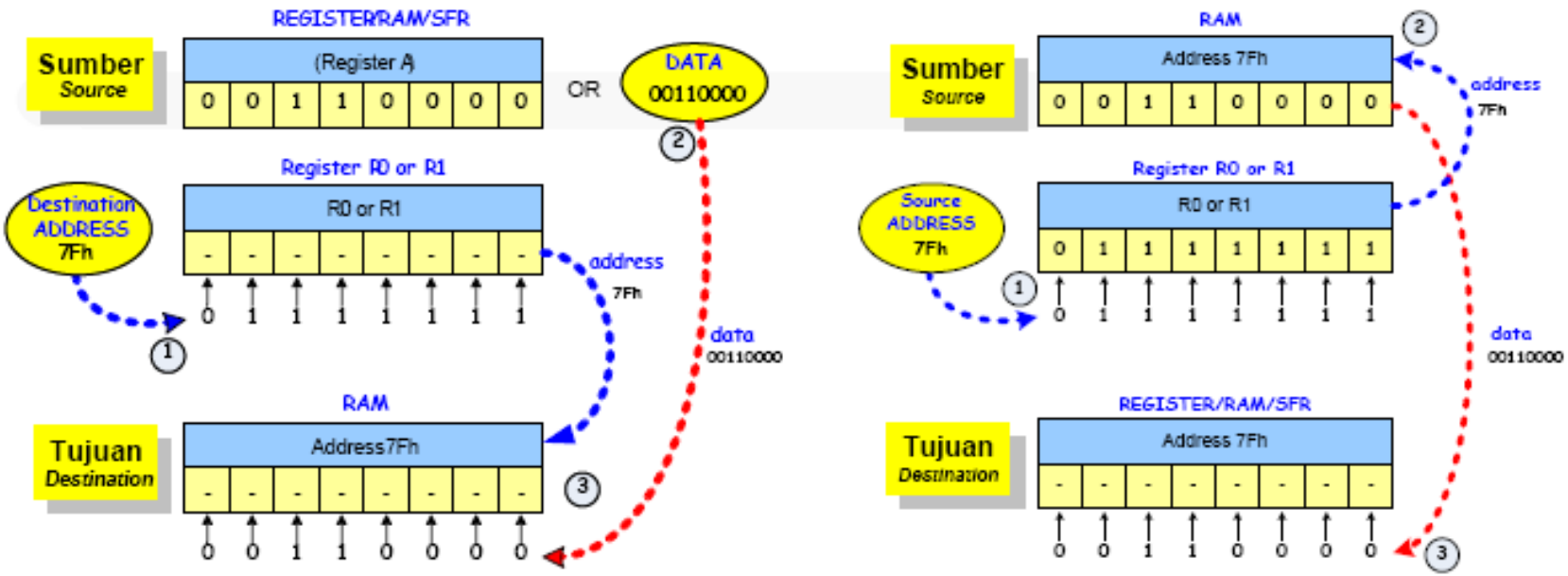
Direct Addressing Mode

Register Addressing : Data dari register/RAM/SFR sumber dipindahkan ke register/RAM/SFR tujuan

Contoh :

```
MOV     A,80h       ; Copy data from port 0 pins to A register
MOV     80h,A       ; Copy data from A register to the port 0
MOV     3Ah,#3Ah    ; Copy immediate data 3Ah to RAM location 3Ah
MOV     R0,12h      ; Copy data from RAM location 12h to R0 register
MOV     8Ch,R7      ; Copy data from R7 register to SFR timer 0 high byte
MOV     5Ch,A       ; Copy data from A register to RAM location 5Ch
MOV     0A8h,77h    ; Copy data from RAM location 77h to IE register
```

Indirect Addressing Mode



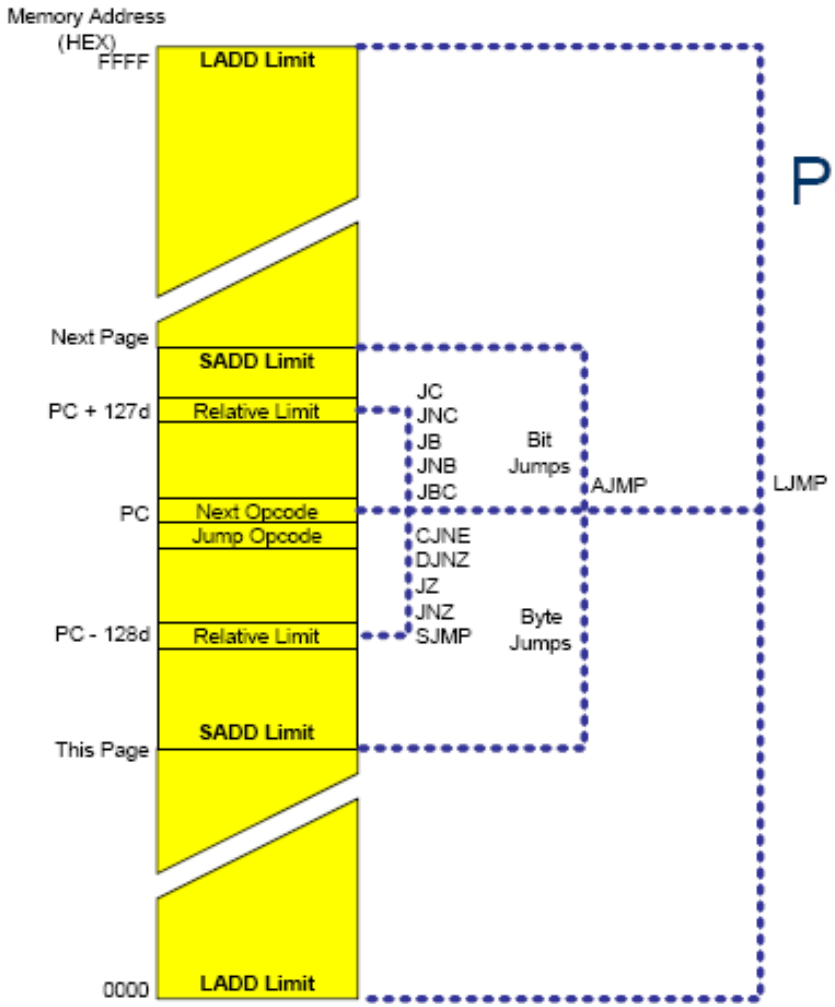
R0 atau R1 digunakan untuk menunjukkan Destination Address

```
MOV     @R0,#30h    ; Copy immediate data 30h to the address in R0
MOV     @R0,30h     ; Copy the content of address 30h to the address in R0
MOV     @R0,A       ; Copy the data in A to the address in R0
```

R0 atau R1 digunakan untuk menunjukkan Source Address

```
MOV     7Fh,@R0     ; Copy the content of the address in R0 to address 7Fh
MOV     A,@R0       ; Copy the content of the address in R0 to A
```

Operasi Percabangan



Rentang Percabangan

Note :
1 page = 2 KByte

JUMP

Instruksi JUMP digunakan untuk melompat ke instruksi tertentu.

Bit Jump

Mnemonic		Operation
JC	radd	Jump relative if the carry flag is set to 1
JNC	radd	Jump relative if the carry flag is clear to 0
JB	b,radd	Jump relative if addressable bit is set to 1
JNB	b,radd	Jump relative if addressable bit is clear to 0
JBC	b,radd	Jump relative if addressable bit is set, and clear the addressable bit to 0

Byte Jump

Mnemonic		Operation
CJNE	A,add,radd	Compare content of A register with the content of the direct address, if not equal jump to relative address, C=1 if A< add, C=0 if others.
CJNE	A,#n,radd	Compare content of A register with immediate number n, if not equal jump to relative address, C=1 if A< n, C=0 if others.
CJNE	Rn,#n,radd	Compare content of Rn register with immediate number n, if not equal jump to relative address, C=1 if Rn< n, C=0 if others.
CJNE	@Rp,#n,radd	Compare the content of address contained in register Rp to the number n, if not equal then jump to relative address, C=1 if content of Rn< n, C=0 if others.
DJNZ	Rn,radd	Decrement register Rn by 1 and jump to relative address if the result is not zero, no flag affected.
DJNZ	add,radd	Decrement the dirrect address by 1 and jump to relative address if the result is not zero, no flag affected.
JZ	radd	Jump to the relative address if A is 0.
JNZ	radd	Jump to the relative address if A is not 0.

Unconditional Jump

Mnemonic		Operation
JMP	@A+DPTR	Jump to address formed by adding A to the DPTR.
AJMP	sadd	Jump to absolute short range address sadd.
LJMP	ladd	Jump to absolute long range address ladd.
SJMP	radd	Jump to absolute relative address radd.
NOP		No operation. Do nothing and go to the next instruction.

Contoh Jump

Contoh 1. Bit Jump

```
        $MOD51
        ORG     0000H
        LJMP    MULAI
        ORG     0100H
MULAI:  MOV     A,#10H
        MOV     R0,A
ADDA:   ADD     A,R0
        JNC     ADDA
;
        MOV     A,#10H
ADDR:   ADD     A,R0
        JNB     CY,ADDR
        JBC     CY,MULAI
        SJMP    MULAI
        END
```

Contoh 2. Byte Jump

```
        $MOD51
        ORG     0000H
        LJMP    BGN
        ORG     0100H
BGN:    MOV     A,#30H
        MOV     50H,#00H
AGN:    CJNE    A,50H,AEQ
        SJMP    NXT
AEQ:    DJNZ    50H,AGN
        NOP
NXT:    MOV     R0,#0FFH
DWN:    DJNZ    R0,DWN
        MOV     A,R0
        JNZ     ABIG
        JZ      AZR0
ABIG:   NOP
;
        ORG     0116H
AZR0:   MOV     A,#08H
        MOV     DPTR,#1000H
        JMP     @A+DPTR
        NOP
        NOP
HERE:   AJMP    AZR0
        END
```

Call dan Sub Rutin

Instruksi CALL digunakan untuk memanggil sub rutin tertentu.

Mnemonic		Operation
ACALL	label	Absolute Call the subroutine named label.
LCALL	label	Long Call the subroutine named label.
RET		Return from subroutine.

Contoh

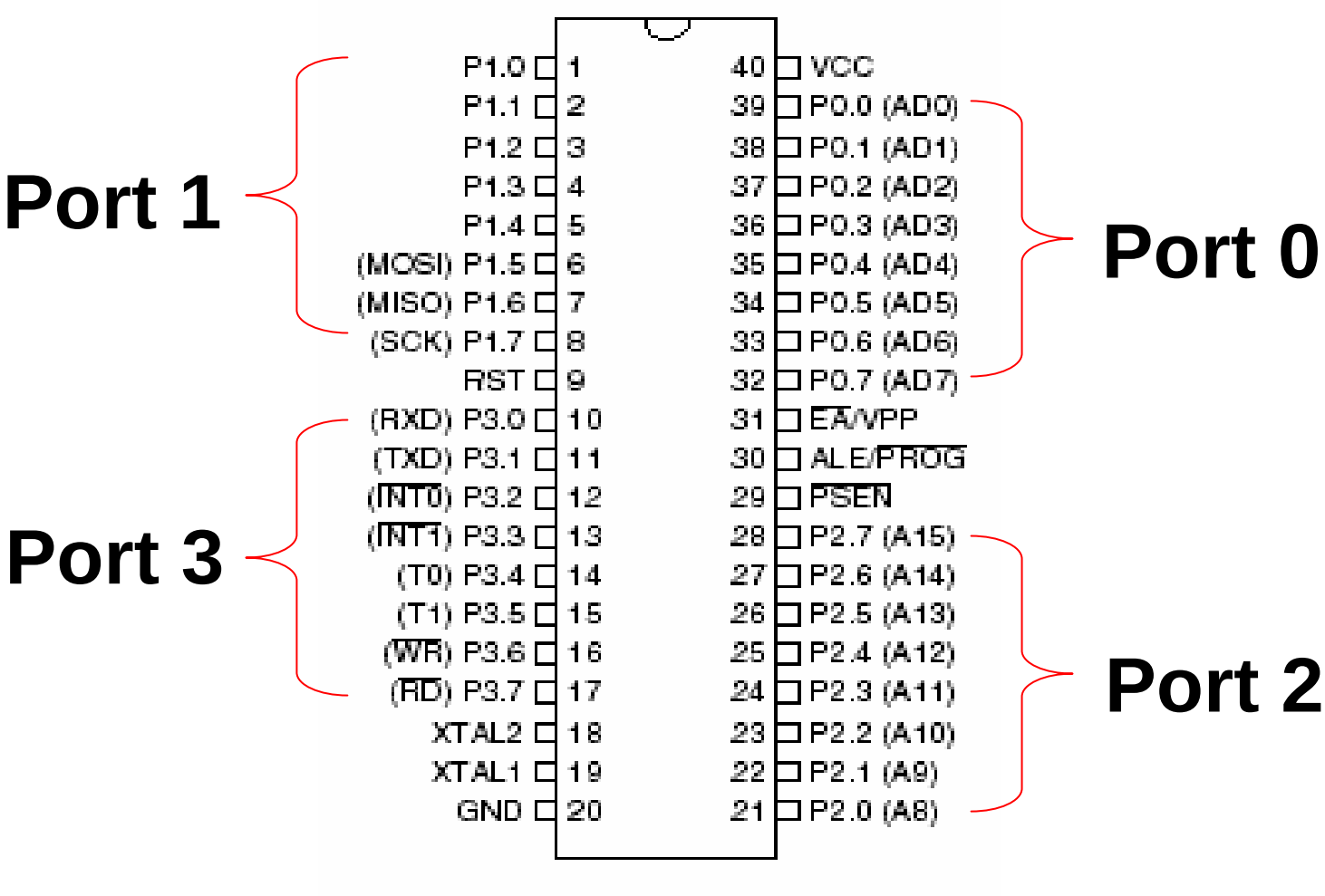
```
        $MOD51
        ORG     0000H
        LJMP    MULAI
        ORG     0100H
MULAI:  MOV     P0,#0FFH
        CALL    DELAY
        MOV     P0,#00H
        CALL    DELAY
        SJMP    MULAI
;
DELAY:  MOV     R0,#5H
DL0:    MOV     R1,#0FH
DL1:    DJNZ    R1,DL1
        DJNZ    R0,DL0
        RET
        END
```

MIKROKONTROLER
PENGGUNAAN PORT PARAREL

Port Pararel

- Port Pararel : Suatu saluran yang digunakan untuk I/O (masukan/keluaran) dimana cara penerimaan/pengiriman datanya dilakukan secara pararel.
- Mikrokontroler 8051 mempunyai 32 pin yang membentuk 4 buah port pararel, yaitu Port 0, Port 1, Port 2 dan Port 3

Diagram Pin


Port 1
Port 3
Port 0
Port 2
P1.0 1
P1.1 2
P1.2 3
P1.3 4
P1.4 5
(MOSI) P1.5 6
(MISO) P1.6 7
(SCK) P1.7 8
RST 9
(RXD) P3.0 10
(TXD) P3.1 11
(INT0) P3.2 12
(INT1) P3.3 13
(T0) P3.4 14
(T1) P3.5 15
(WR) P3.6 16
(RD) P3.7 17
XTAL2 18
XTAL1 19
GND 20
40 VCC
39 P0.0 (AD0)
38 P0.1 (AD1)
37 P0.2 (AD2)
36 P0.3 (AD3)
35 P0.4 (AD4)
34 P0.5 (AD5)
33 P0.6 (AD6)
32 P0.7 (AD7)
31 EA/VPP
30 ALE/PROG
29 PSEN
28 P2.7 (A15)
27 P2.6 (A14)
26 P2.5 (A13)
25 P2.4 (A12)
24 P2.3 (A11)
23 P2.2 (A10)
22 P2.1 (A9)
21 P2.0 (A8)

Fungsi Port I/O / Pararel

Port 0	Port 0 merupakan port paralel 8 bit dua arah (bi-directional) yang dapat digunakan untuk berbagai keperluan. Port 0 juga memultipleks alamat dan data jika digunakan untuk mengakses memori eksternal
Port 1	Port 1 merupakan port paralel 8 bit bi-directional dengan internal pull-up. Port 1 juga digunakan dalam proses pemrograman (In System Programming) → P1.5 MOSI; P1.6 MISO ; P1.7 SCK
Port 2	Port 2 merupakan port paralel 8 bit bi-directional dengan internal pull-up. Port 2 akan mengirim byte alamat jika digunakan untuk mengakses memori eksternal.
Port 3	Port 3 merupakan port paralel 8 bit bi-directional dengan internal pull-up. Port 3 juga bisa difungsikan untuk keperluan khusus

Fungsi Khusus Port 3

PIN	FUNGSI ALTERNATIF
P3.0	RXD (port input serial)
P3.1	TXD (port output serial)
P3.2	INT0 (interrupt eksternal 0)
P3.3	INT1 (interrupt eksternal 1)
P3.4	T0 (input eksternal timer 0)
P3.5	T1 (input eksternal timer 1)
P3.6	WR (strobe penulisan data eksternal)
P3.7	RD (strobe pembacaan data eksternal)

Struktur Port Dan Cara Kerja

Mempunyai 2 cara pengiriman data:

1. Bekerja pada port seutuhnya, artinya semua 8 jalur dari port diperlukan.

   Contoh: Mov P0,#FFh

   → Membuat 8 jalur dari Port 0 semuanya dalam kondisi logika ‘1’ (atau isinya 1111 1111 dalam biner).

2. Bekerja pada satu jalur atau bit dari port.

   Contoh: Setb P3.4

   → Membuat logika 1 bit ke 4 dari Port 3

Struktur Port Dan Cara Kerja

Mempunyai 2 cara penerimaan data:

1. Digunakan untuk membaca data pada seluruh bit.

   Contoh: Mov A,P3

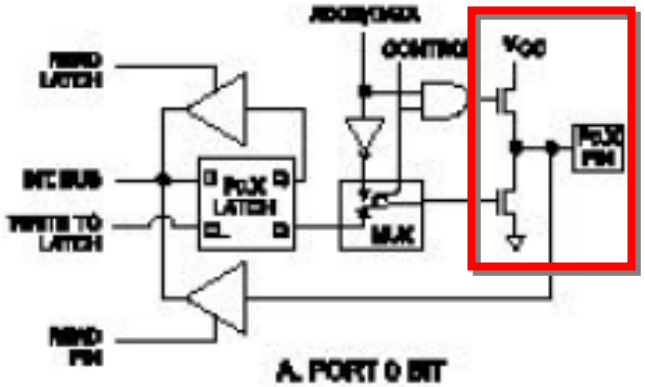
   → Membaca data seluruh bit pada Port 3 dan disimpan kedalam akumulator.

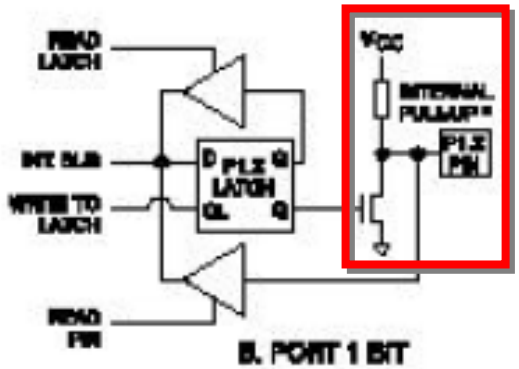
2. Pembacaan data dilakukan hanya pada 1 bit.

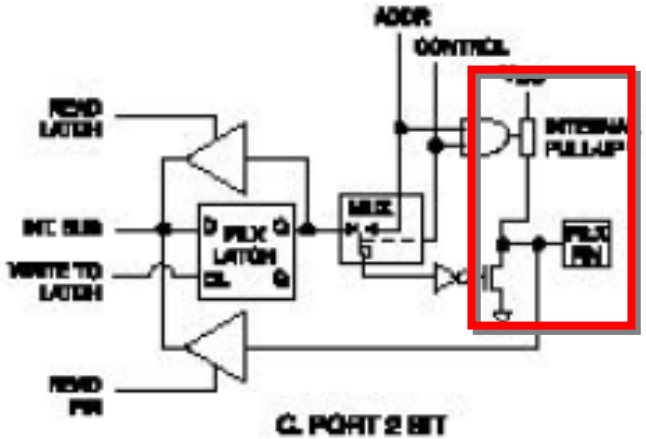
   Contoh: Jnb P3.7,$

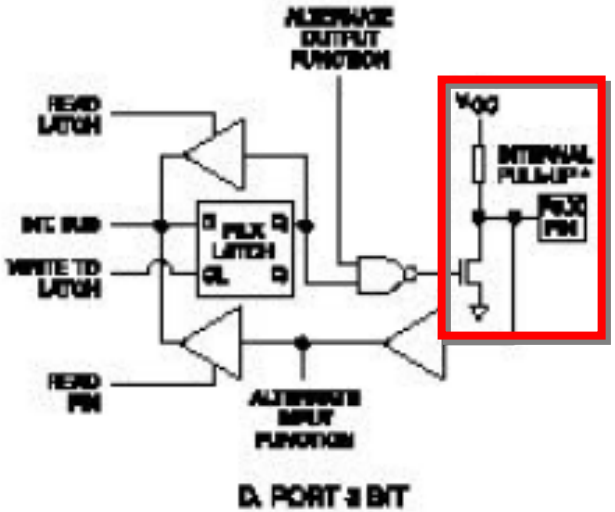
   → Digunakan untuk memantau bit ke 7 dari Port 3.

Konfigurasi Port





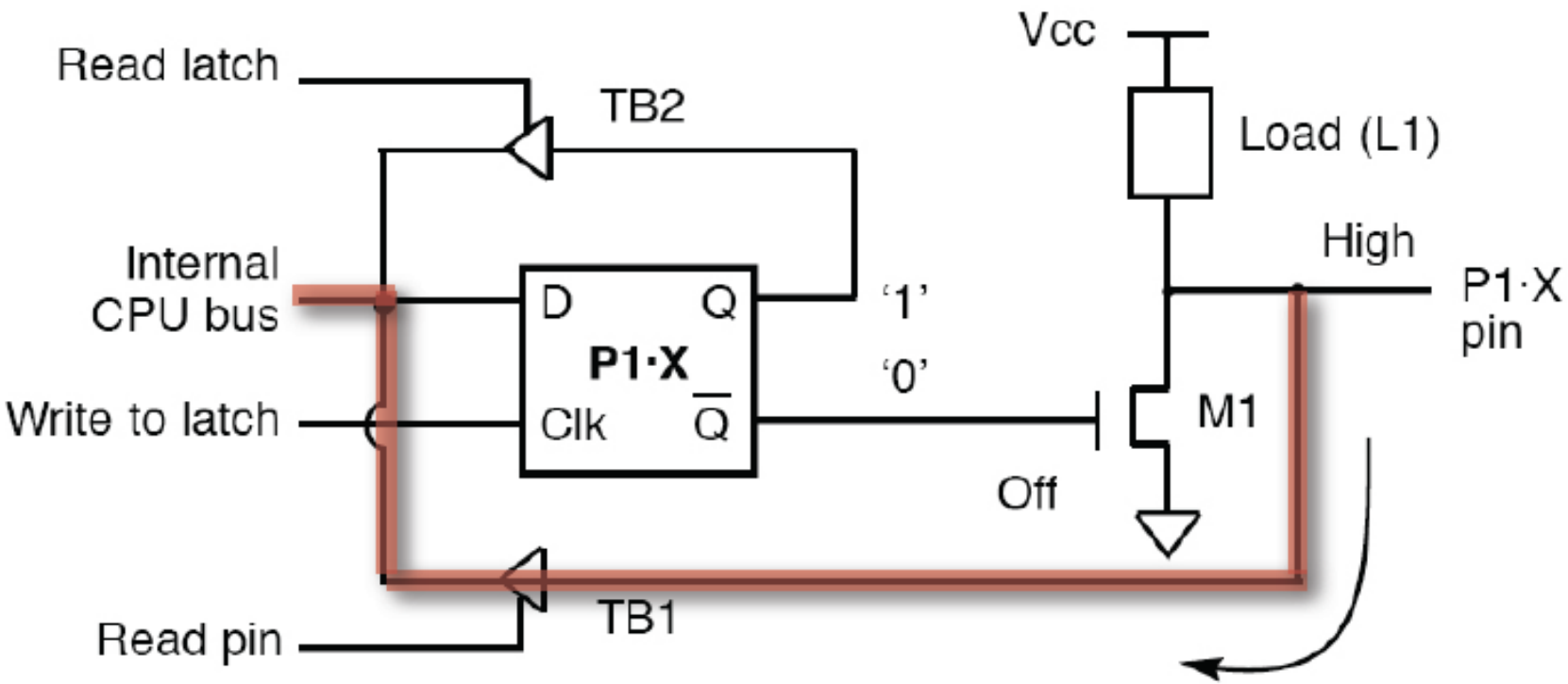




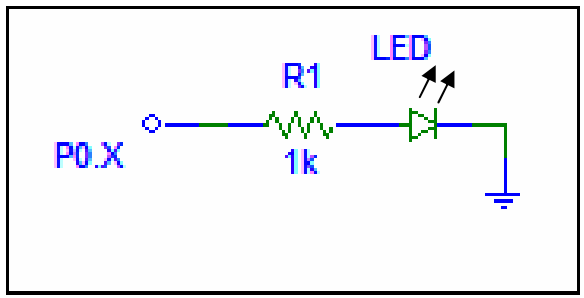
Konfigurasi Port

- Konfigurasi Port 0 menggunakan internal FET Pull Up
- Konfigurasi Port 1, 2 dan 3 menggunakan internal Resistor Pull Up

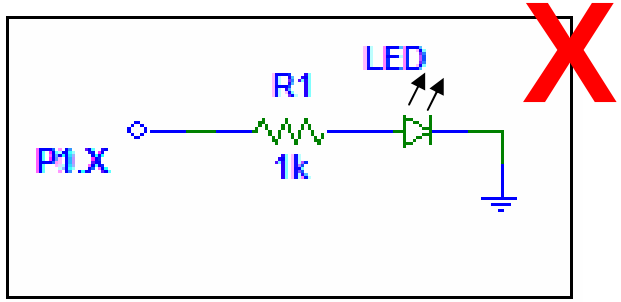
Pembacaan Data Melalui Port



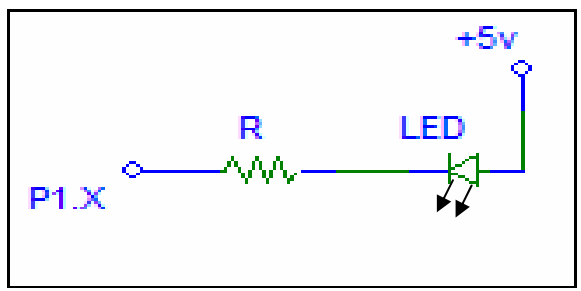
Pengiriman Data Melalui Port



OK Nyalakan LED P0.X=1



X

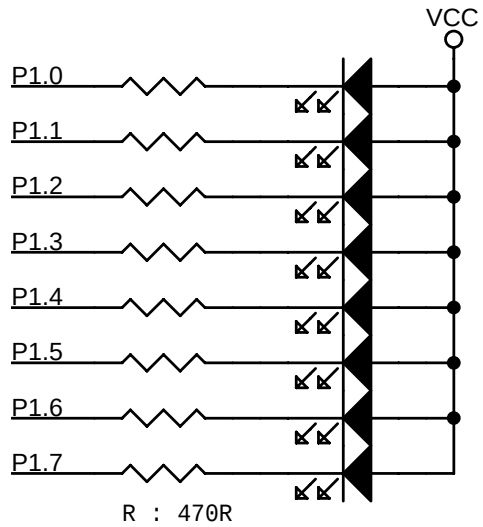


OK

Nyalakan LED P1.X=0

Contoh Aplikasi Rangkaian LED

Untuk menghidupkan LED pada Port 1 harus dikirim atau dituliskan logika ‘0’



Contoh Aplikasi
Menghidupkan dan Mematikan LED Bergantian

4 LED mati hidup secara bergantian:

```
1:          ORG 0H
2:  Mulai:  MOV  P1,#00001111B
3:          ACALL  Delay
4:          MOV  P1,#11110000B
5:          ACALL  Delay
6:          SJMP  Mulai
```

Contoh Aplikasi
Menghidupkan dan Mematikan LED Bergantian

Baris 1 digunakan agar instruksi dituliskan mulai alamat 0H.

Baris 2 mengirimkan data 00001111B (biner) ke Port 1 agar LED4-LED7 (Pada Port 1.4 – Port 1.7) menyala.

Baris 3 digunakan untuk memanggil subrutin delay

Baris 4 mengirimkan data 11110000B (biner) ke Port 1 agar LED0-LED3 (Pada Port 1.0 – Port 1.3) menyala. Kemudian memanggil sub rutin delay lagi.

Baris 5 digunakan untuk mengulang instruksi dari awal

Contoh Aplikasi
Menghidupkan dan Mematikan LED Bergantian

```
7:  ; subrutin delay
8:  Delay:  MOV R0,#5
9:  Delay1: MOV R1,#0FFH
10: Delay2: MOV R2,#0
11:         DJNZ R2,$
12:         DJNZ R1,Delay2
13:         RET
14:         END
```

Contoh Aplikasi
Menghidupkan dan Mematikan LED Bergantian

Baris 9 dikerjakan sebanyak 326.400x, karena instruksi tersebut dikerjakan selama 2 siklus totalnya 326.400x2=652.800 siklus, masih ditambah pengulangan kedua 255x3 = 765 siklus dan pengulangan ke tiga sebesar 5x3 siklus sehingga total 752.800+765+15 = 653.800 siklus.

Jika menggunakan frekuensi kristal 12 MHz waktu yang dibutuhkan untuk menyelesaikan subrutin adalah 653.580 x 1 md = 653.580 md = 0,65 detik.

Untuk pewaktuan yang akurat bisa menggunakan timer yang akan dibahas pada bagian selanjutnya.

MIKROKONTROLER

CONTOH PORT PARAREL

SEVEN SEGMENT

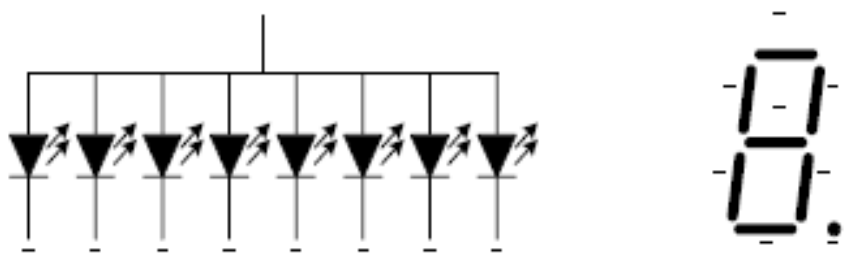
Aplikasi Seven Segment

Seven segment adalah tampilan angka yang terdiri dari 7 LED yang disusun membentuk angka 8 ditambah 1 LED sebagai titik (dot). Ada dua tipe 7 Segment yaitu : Common Anode dan Common Cathode

Common Anode (CA)

Pada 7 segment CA semua anoda LED dihubungkan menjadi satu dan disebut sebagai Common Anode, sementara katoda LED diberi nama a, b, c, d, e, f, g dan dp (dot/titik).

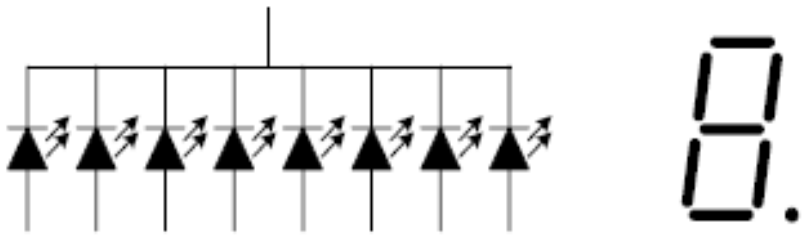
Tanda bar diatas menunjukkan bahwa pin tersebut adalah aktif low. Sebagai contoh untuk membentuk angka 2 maka pin common diberi tegangan + sedangkan pin a, c, d, f dan g diberi tegangan 0 volt.
Besarnya tegangan forward (Vf) LED dapat dilihat dari lembaran data sheet tiap produk seven segment.



Konfigurasi Seven Segment Common Anode

Common Cathode

Common Cathode (CC)
Pada 7 segment CC semua katoda LED dihubungkan menjadi satu dan disebut sebagai Common Cathode, sementara katoda LED diberi nama a, b, c, d, e, f, g dan dp (dot/titik). Sebagai contoh untuk membentuk angka 1 maka pin common diberi tegangan 0 volt sedangkan pin a dan b diberi tegangan +.

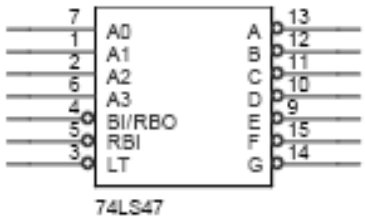


Konfigurasi Seven Segment Common Cathode

BCD To Seven Segment

BCD (Binary Coded Decimal) to Seven Segment adalah sebuah decoder yang dapat mengubah kode biner menjadi tampilan angka pada seven segment. Ada dua tipe BCD to 7 segment yaitu : 74LS47 dan 74LS48.

74LS47 (BCD to 7 Segment Common Anode)



74LS47

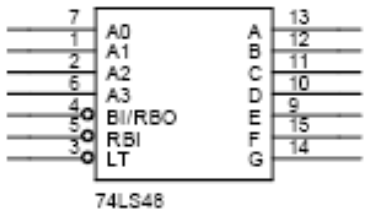
Nama Pin	Deskripsi
A0-A3	Input BCD
$\overline{RBI}$	Ripple Blanking Input (Aktif Low)
$\overline{LT}$	Lamp Test Input (Aktif Low)
$\overline{BI}$ / $\overline{RBO}$	Blanking Input/Ripple Blanking Output (Aktif Low)
$\overline{a}$ - $\overline{g}$	Output Segment

Tabel Kebenaran

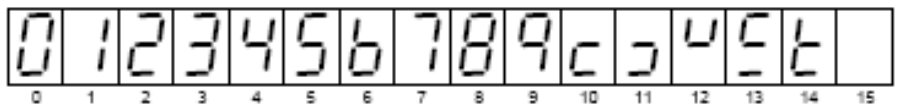
Tabel Kebenaran 74LS47

Tabel Kebenaran															
Desimal / Fungsi	Input							Output							
	$\overline{LT}$	$\overline{RBI}$	A3	A2	A1	A0	$\overline{BI}$ / $\overline{RBO}$	$\overline{a}$	$\overline{b}$	$\overline{c}$	$\overline{d}$	$\overline{e}$	$\overline{f}$	$\overline{g}$
0	1	1	0	0	0	0	1	0	0	0	0	0	0	1
1	1	X	0	0	0	1	1	1	0	0	1	1	1	1
2	1	X	0	0	1	0	1	0	0	1	0	0	1	0
3	1	X	0	0	1	1	1	0	0	0	0	1	1	0
4	1	X	0	1	0	0	1	1	0	0	1	1	0	0
5	1	X	0	1	0	1	1	0	1	0	0	1	0	0
6	1	X	0	1	1	0	1	1	1	0	0	0	0	0
7	1	X	0	1	1	1	1	0	0	0	1	1	1	1
8	1	X	1	0	0	0	1	0	0	0	0	0	0	0
9	1	X	1	0	0	1	1	0	0	0	1	1	0	0
10	1	X	1	0	1	0	1	1	1	1	0	0	1	0
11	1	X	1	0	1	1	1	1	1	0	0	1	1	0
12	1	X	1	1	0	0	1	1	0	1	1	1	0	0
13	1	X	1	1	0	1	1	0	1	1	0	1	0	0
14	1	X	1	1	1	0	1	1	1	1	0	0	0	0
15	1	X	1	1	1	1	1	1	1	1	1	1	1	1
$\overline{BI}$	X	X	X	X	X	X	0	1	1	1	1	1	1	1
$\overline{RBI}$	1	0	0	0	0	0	0	1	1	1	1	1	1	1
$\overline{LT}$	0	X	X	X	X	X	1	0	0	0	0	0	0	0

74LS48 (BCD to 7 Segment Common Cathode)



Nama Pin	Deskripsi
A0-A3	Input BCD
$\overline{RBI}$	Ripple Blanking Input (Aktif Low)
$\overline{LT}$	Lamp Test Input (Aktif Low)
$\overline{BI}$ / $\overline{RBO}$	Blanking Input/Ripple Blanking Output (Aktif Low)
a - g	Output Segment



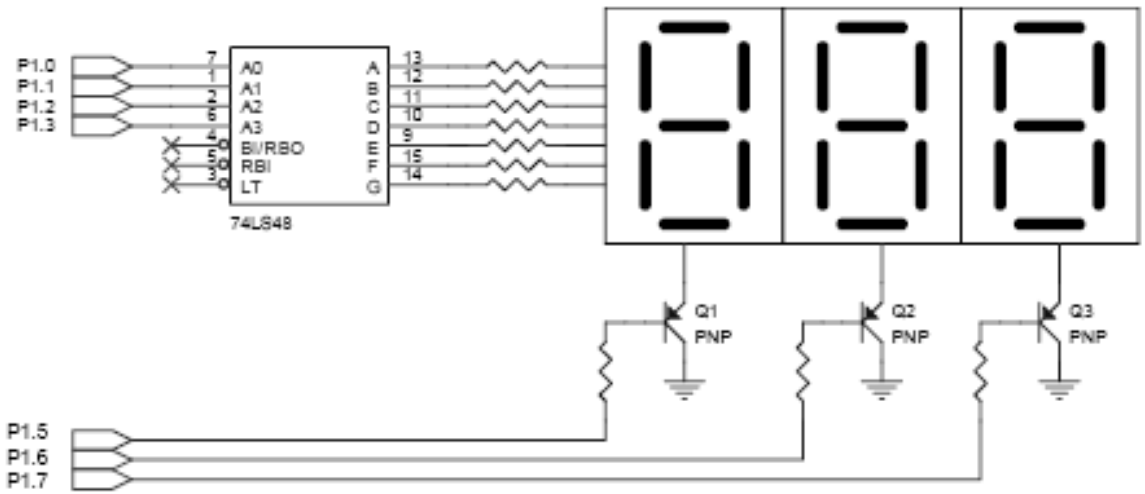
Tabel Kebenaran

Tabel Kebenaran 74LS48

Tabel Kebenaran														
Desimal / Fungsi	Input							Output						
	$\overline{LT}$	$\overline{RBI}$	A3	A2	A1	A0	$\overline{BI}$ / $\overline{RBO}$	a	b	c	d	e	f	g
0	1	1	0	0	0	0	1	1	1	1	1	1	1	0
1	1	X	0	0	0	1	1	0	1	1	0	0	0	0
2	1	X	0	0	1	0	1	1	1	0	1	1	0	1
3	1	X	0	0	1	1	1	1	1	1	1	0	0	1
4	1	X	0	1	0	0	1	0	1	1	0	0	1	1
5	1	X	0	1	0	1	1	1	0	1	1	0	1	1
6	1	X	0	1	1	0	1	0	0	1	1	1	1	1
7	1	X	0	1	1	1	1	1	1	1	0	0	0	0
8	1	X	1	0	0	0	1	1	1	1	1	1	1	1
9	1	X	1	0	0	1	1	1	1	1	0	0	1	1
10	1	X	1	0	1	0	1	0	0	0	1	1	0	1
11	1	X	1	0	1	1	1	0	0	1	1	0	0	1
12	1	X	1	1	0	0	1	0	1	0	0	0	1	1
13	1	X	1	1	0	1	1	1	0	0	1	0	1	1
14	1	X	1	1	1	0	1	0	0	0	1	1	1	1
15	1	X	1	1	1	1	1	0	0	0	0	0	0	0
$\overline{BI}$	X	X	X	X	X	X	0	0	0	0	0	0	0	0
$\overline{RBI}$	1	0	0	0	0	0	0	0	0	0	0	0	0	0
$\overline{LT}$	0	X	X	X	X	X	1	1	1	1	1	1	1	1

Metoda Scanning

Metoda scanning digunakan untuk melakukan penghematan jalur data yang diperlukan untuk mengendalikan seven segmen yang jumlahnya lebih dari satu buah seperti pada gambar dibawah ini.



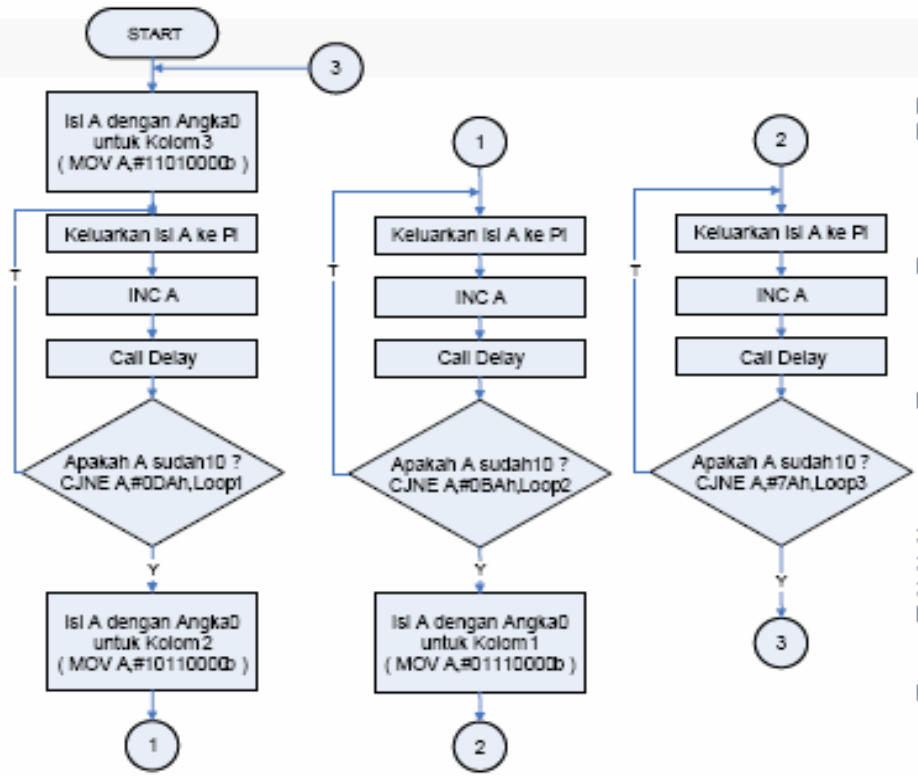
Dengan metoda scanning, semua output segmen dari 74LS48 (a sampai g) dihubungkan ke semua seven segmen. Dengan demikian data akan diterima oleh semua seven segmen secara bersamaan. Yang harus dilakukan selanjutnya adalah memilih common mana yang akan diaktifkan. Dengan mengaktifkan common secara bergantian dan dilakukan dalam frekuensi yang cepat (50 Hz) maka seolah-olah akan dilihat tiga digit angka yang menyala bersamaan.

Contoh Seven Segment

Contoh 1 :

Tiga digit seven segment didesain untuk menampilkan angka 0-9 pada tiap kolomnya secara bergantian (hanya satu kolom yang menyala). Dimulai dari kolom ketiga, kemudian kolom kedua dan kolom pertama, demikian berulang-ulang.

Flowchart Program :



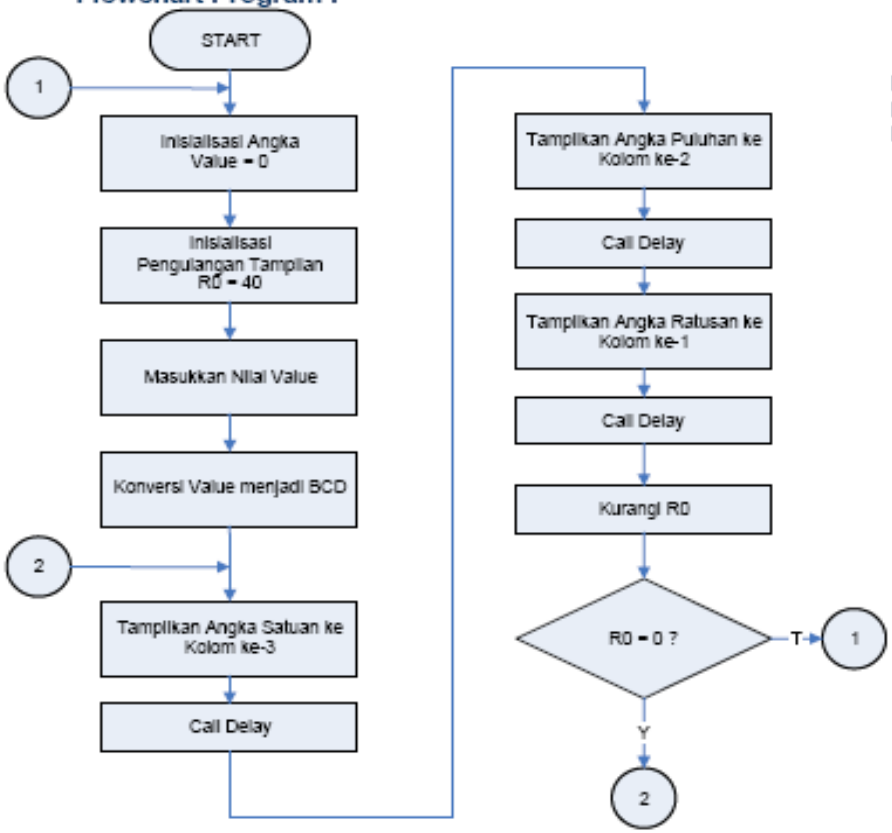
Listing Program :

```
                $mod51
                ORG     0000H
                LJMP    MULAI
                ORG     0100H
MULAI:          MOV     A,#11010000B
LOOP1:          MOV     P1,A
                INC     A
                ACALL   DELAY
                CJNE    A,#0DAH,LOOP1
                MOV     A,#10110000B
LOOP2:          MOV     P1,A
                INC     A
                ACALL   DELAY
                CJNE    A,#0BAH,LOOP2
                MOV     A,#01110000B
LOOP3:          MOV     P1,A
                INC     A
                ACALL   DELAY
                CJNE    A,#7AH,LOOP3
                SJMP    MULAI
;-------------------------------
; DELAY SUBROUTINE
;-------------------------------
DELAY:          MOV     R5,#0FFH
                MOV     R6,#0FFH
                MOV     R7,#5
DLY:            DJNZ    R5,DLY
                DJNZ    R6,DLY
                DJNZ    R7,DLY
                RET
                END
```

Contoh Seven Segment

Contoh 2 :
Tiga digit seven segment didesain untuk menampilkan angka 000 sampai 255 kemudian berulang kembali dari 000.

Flowchart Program :



Listing Program :

```
;---------- VARIABLES ----------
ANGKA1      EQU     30H
ANGKA2      EQU     31H
ANGKA3      EQU     32H
VALUE       EQU     33H
;---------- MAIN PROGRAM --------
            ORG     0000H
            LJMP    MULAI
            ORG     0100H
MULAI:      MOV     VALUE,#0
LOOP1:      MOV     R0,#40
LOOP2:      MOV     A,VALUE
            MOV     B,#0AH
            DIV     AB
            MOV     ANGKA3,B
            MOV     B,#0AH
            DIV     AB
            MOV     ANGKA2,B
            MOV     ANGKA1,A
            MOV     A,ANGKA3
            ANL     A,#0FH
            ORL     A,#70H
            MOV     P1,A
            CALL    DELAY
            MOV     A,ANGKA2
            ANL     A,#0FH
            ORL     A,#0B0H
            MOV     P1,A
            CALL    DELAY
            MOV     A,ANGKA1
            ANL     A,#0FH
            ORL     A,#0D0H
            MOV     P1,A
            CALL    DELAY
            DJNZ    R0,LOOP2
            INC     VALUE
            SJMP    LOOP1
```

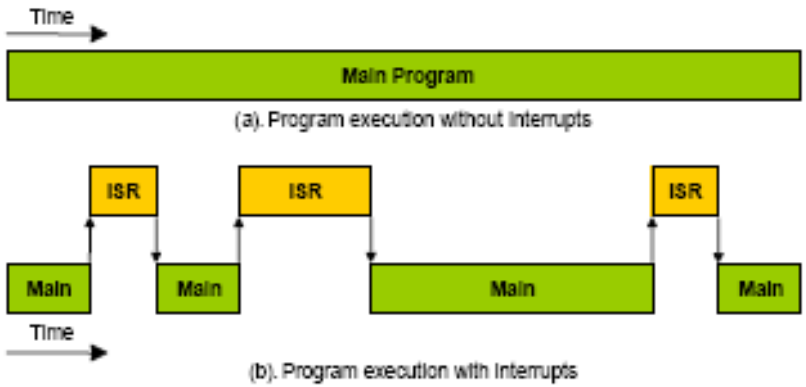
MIKROKONTROLER

INTERUPSI & PORT SERIAL

Operasi Interupsi

Interupsi adalah kondisi yang memaksa mikrokontroler menghentikan sementara eksekusi program utama untuk mengeksekusi rutin interrupt tertentu / ***Interrupt Service Routine (ISR)***

Setelah melaksanakan ISR secara lengkap, maka mikrokontroler akan kembali melanjutkan eksekusi program utama yang tadi ditinggalkan.



Pada AT89S51, ada 5 sumber interrupt yaitu

1. System reset
2. External 0
3. Timer 0
4. External 1
5. Timer 1
6. Serial Port.

Untuk mengatur kerja interrupt, dapat dilakukan pengaturan pada register ***Interrupt Enable (IE)*** dan ***Interrupt Priority (IP).***

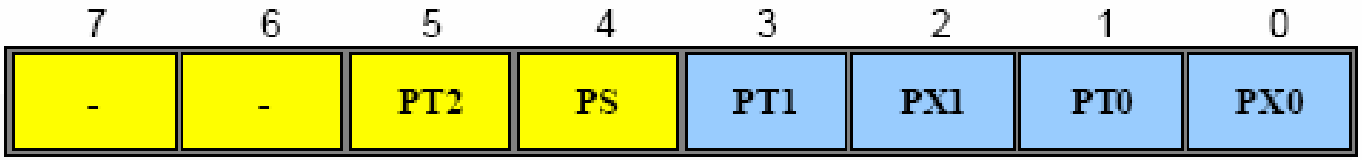
Register IE

IE / Interrupt Enable Special Function Register

Bit	Symbol	Fuction
7	EA	Enable interrupts bit Clear ke 0 oleh program untuk melumpuhkan semua interruptset 1 untuk mengaktifkan interrupt sesuai enable bit interrupt terkait
6	-	Tidak digunakan
5	ET2	Reserved for future use
4	ES	Enable serial port interrupt Set 1 oleh program untuk mengaktifkan serial port interrupt clear untuk melumpuhkan
3	ET1	Enable timer 1 overflow interrupt Set 1 oleh program untuk mengaktifkan timer1 overflow interrupt clear untuk melumpuhkan
2	EX1	Enable external interrupt1. Set 1 oleh program untuk mengaktifkan interrupt (INT1), clear untuk melumpuhkan
1	ET0	Enable timer 0 overflow interrupt Set 1 oleh program untuk mengaktifkan timer0 overflow interrupt clear untuk melumpuhkan
0	EX0	Enable external interrupt0. Set 1 oleh program untuk mengaktifkan interrupt0 (INT0), clear untuk melumpuhkan

Bit addressable as IE0 to IE.7

Register IP



IP / Interrupt Priority Special Function Register

Bit	Symbol	Fuction
7	-	Tidak digunakan
6	-	Tidak digunakan
5	PT2	Reserved for future use
4	PS	Priority of serial port interrupt
3	PT1	Priority of timer 1 overflow interrupt
2	PX1	Priority of external interrupt1.
1	PT0	Priority of timer 0 overflow interrupt
0	PX0	Priority of external interrupt0.

Bit addressable as IP.0 to IP.7

Serial Port Interrupt

Serial port interrupt terjadi jika **transmit interrupt flag (TI)** atau **receive interrupt flag (RI)** dalam kondisi set (1).

Transmit interrupt terjadi ketika mikrokontroler telah berhasil mengirim data secara lengkap.
Receive interrupt terjadi ketika mikrokontroler telah berhasil menerima data secara lengkap.

Flag TI dan RI harus di-clear (0) oleh program sebab kedua flag ini tidak otomatis clear secara hardware ketika ISR selesai dijalankan.

```
            ORG     0000h
            LJMP    MAIN
            ORG     0023h               ; serial port interrupt vector
            LJMP    SP_ISR              ; jump ke Serial Port ISR
            ORG     0030h
MAIN:       MOV     TMOD,#20h           ; Timer 1, mode 2
            MOV     TH1,#0FDh           ; 1200 baudrate
            SETB    TR1                 ; start Timer
            MOV     SCON,#42h           ; mode 1
            MOV     A,#20h              ; send ASCII space first
            MOV     IE,#90h             ; enable serial port interrupt
            SJMP    $                   ; do nothing
;
SP_ISR:     CJNE    A,#7F,SKIP          ; if finished ASCII set,
            MOV     A,#20h              ; reset to SPACE
SKIP:       MOV     SBUF,A              ; send char. to serial port
            INC     A                   ; increment ASCII code
            CLR     TI                  ; clear interrupt flag
            RETI
            END
```

External Interrupt

External interrupt terjadi jika ***interrupt external flag (IE0 atau IE1)*** pada TCON dalam kondisi set (1).
Interrupt external flag dalam kondisi set jika :

1. Terdapat sinyal low pada Pin INT0 atau INT1atau ;
2. Terdapat perubahan sinyal dari high ke low Pin INT0 atau INT1

Pengaturan kondisi sinyal interrupt ini dilakukan pada bit IT0 dan IT1 dalam register TCON. Jika ITx = 0 maka sinyal interrupt low, jika ITx=1 maka sinyal interrupt dari high ke low. Flag IEx akan kembali 0 jika subrutin interrupt dieksekusi. Untuk kembali dari subrutin interrupt digunakan instruksi RETI ***(RETurn from Interrupt)***

Reset

Reset termasuk interrupt yang tidak dapat dilumpuhkan, sebab reset dipicu secara hardware yaitu pada pin RST dan menyebabkan mikrokontroler mengeksekusi instruksi pada address 0000h. Ketika reset, nilai internal register menjadi sbb :

REGISTER	VALUE(HEX)	REGISTER	VALUE(HEX)
PC	0000	TCON	00
DPTR	0000	TMOD	00
A	00	TH0	00
B	00	TL0	00
SP	07	TH1	00
PSW	00	TL1	00
P0-3	FF	SCON	00
IP	xxx0000b	SBUF	xx
IE	0xx0000b	PCON	0xxxxxxxxb

Contoh Interrupt

Contoh 1. Interrupt 0

```
                $MOD51
                ORG     0000H
                LJMP    MULAI
                ORG     0003H
                LJMP    INT0_ISR
                ORG     0100H
MULAI:          MOV     IE,#81H     ; Interrupt 0 enable
                MOV     IP,#01H     ; Prioritaskan
LOOP:           MOV     P0,#00
                CALL    DELAY
                MOV     P0,#0FFH
                CALL    DELAY
                SJMP    LOOP
;-------------------------------
; INTERRUPT 0 SERVICE ROUTINE
;-------------------------------
INT0_ISR:       PUSH    05H
                PUSH    06H
                PUSH    07H
                MOV     R0,#16
                MOV     A,#01H
LOOP1:          MOV     P0,A
                CALL    DELAY
                RL      A
                DJNZ    R0,LOOP1
                POP     07H
                POP     06H
                POP     05H
                RETI
;
DELAY:          MOV     R5,#0FFH
                MOV     R6,#0FH
                MOV     R7,#05H
DLY:            DJNZ    R5,DLY
                DJNZ    R6,DLY
                DJNZ    R7,DLY
                RET
                END
```

Contoh 2. Interrupt 1

```
                $MOD51
                ORG     0000H
                LJMP    MULAI
                ORG     0013H
                LJMP    INT1_ISR
                ORG     0100H
MULAI:          MOV     IE,#84H     ; Interrupt 1 enable
                MOV     IP,#04H     ; Prioritaskan
LOOP:           MOV     P0,#00
                CALL    DELAY
                MOV     P0,#0FFH
                CALL    DELAY
                SJMP    LOOP
;-------------------------------
; INTERRUPT 0 SERVICE ROUTINE
;-------------------------------
INT1_ISR:       PUSH    05H
                PUSH    06H
                PUSH    07H
                MOV     R0,#16
                MOV     A,#80H
LOOP1:          MOV     P0,A
                CALL    DELAY
                RL      A
                DJNZ    R0,LOOP1
                POP     07H
                POP     06H
                POP     05H
                RETI
;
DELAY:          MOV     R5,#0FFH
                MOV     R6,#0FH
                MOV     R7,#05H
DLY:            DJNZ    R5,DLY
                DJNZ    R6,DLY
                DJNZ    R7,DLY
                RET
                END
```

Lokasi Memori Interrupt

Interrupt Service Routine / ISR ditempatkan pada address yang berbeda untuk tiap sumber interrupt. Berikut adalah tabel sumber interrupt, flag yang dipengaruhi dan vector address tiap sumber interrupt.

INTERRUPT	FLAG	VECTOR ADDRESS
System reset	RST	0000h
External 0	IE0	0003h
Timer 0	TF0	000Bh
External 1	IE1	0013h
Timer 1	TF1	001B
Serial Port	RI or TI	0023h

Komunikasi Data Serial

Komunikasi serial memiliki keuntungan dari segi efektifitasnya karena hanya membutuhkan 2 jalur komunikasi, jalur data dan clock. Data dikirim/diterima per bit secara bergantian. Pada MCS-51, data ditampung sementara dalam register SBUF ***(Serial Buffer)*** sebelum dikirim/diterima.
Untuk mengatur mode komunikasi data serial dilakukan oleh register SCON ***(Serial Control register)***.
Untuk mengatur baudrate dilakukan oleh register PCON ***(Power Control register)***.
Pada AT89S51, port serial terdapat pada P3.0(RXD) dan P3.1 (TXD)

Ada 4 mode komunikasi data serial yang bisa dilakukan mikrokontroler AT89S51 yang dapat dipilih pada bit SM0 an SM1 dalam SCON.

Dalam SCON terdapat flag TI *(Transmit Interrupt)* dan RI *(Receive Interrupt)* yang menandakan sedang terjadi pengiriman atau penerimaan data.

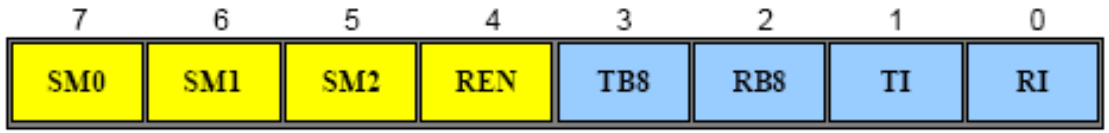
Pengiriman Data

Pengiriman data serial dimulai ketika sebuah byte data dikirimkan ke SBUF. TI akan 1 ketika data telah selesai dikirimkan.

Penerimaan Data

Penerimaan data serial dimulai ketika REN dalam SCON di set 1. RE akan 1 ketika data telah selesai diterima. Data tersebut kemudian disimpan di dalan SBUF

Register SCON



SCON / Serial Port Control Special Function Register

Bit	Symbol	Fuction
7	SM0	Serial port mode bit 0.
6	SM1	Serial port mode bit 1. SM1 SM0 Mode Description 0 0 0 Shift register, baud = f/12 0 1 1 8-bit UART, baud = variable 1 0 2 9-bit UART, baud = f/32 or f/64 1 1 3 9-bit UART, baud = variable
5	SM2	Multiprocessor communications bit Set/clear oleh program untuk mengaktifkan komunikasi multiprosesor pada mode2 dan 3. Jika di-set 1, sebuah interrupt akan dihasilkan jika bit ke9 dari data yang diterima adalah 1. Tidak ada interrupt yang dihasilkan jika bit ke9 adalah 0. Jika di-set 1 pada mode 1, tidak ada interrupt yang dihasilkan kecuali jika sebuah bit stop telah diterima Clear ke 0 jika mode 0 digunakan.
4	REN	Receive enable bit Set 1 untuk mengaktifkan penerimaan, clear ke 0 untuk melumpuhkan penerimaan
3	TB8	Transmitted bit 8. Set/clear oleh program pada mode2 dan 3.
2	RB8	Received bit 8. Bit ke-8 dari data yang diterima pada mode2 dan 3 ; stop bit pada mode 1. Tidak digunakan pada mode0.
1	TI	Transmit interrupt flag Set 1 pada akhir bit ke-7 pada mode 0, dan pada awal bit stop pada mode lain. Harus di-clear oleh program
0	RI	Receive interript flag Set 1 pada akhir bit ke-7 pada mode 0,dan "setengah jalan" pada bit stop pada mode lain. Harus di-clear oleh program

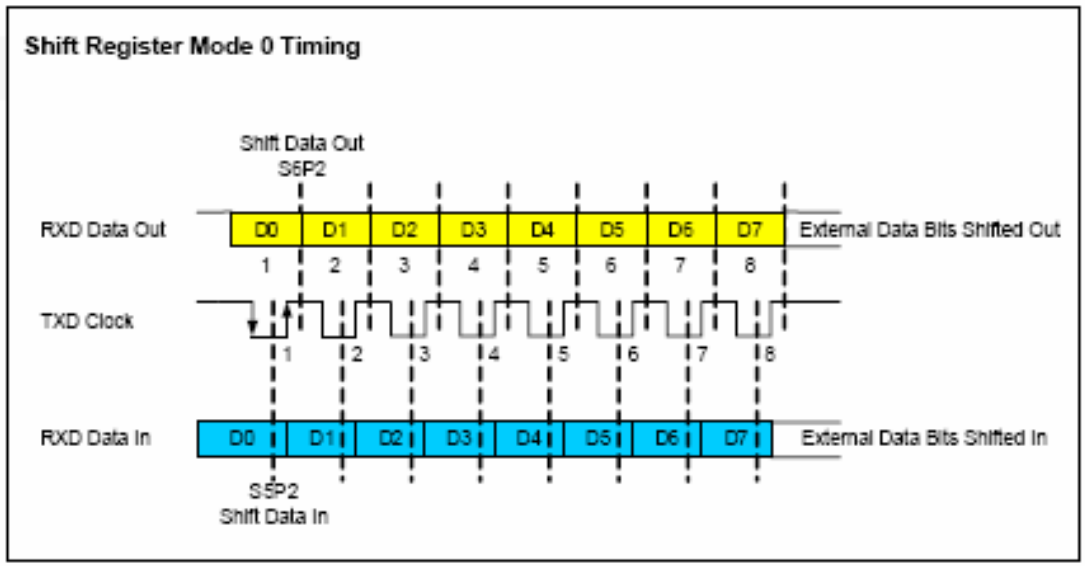
Register PCON

PCON / Power Mode Control Special Function Register

Bit	Symbol	Fuction
7	SMOD	Serial baud rate modify bit . Set 1 oleh program untuk menggandakan baud rate menggunakan timer 1 pada mode 1, 2 dan 3. Clear oleh program untuk menggunakan baud rate timer 1.
6-4	-	Tidak digunakan
3	GF1	General pupose user flag bit 1.
2	GF0	General pupose user flag bit 0.
1	PD	Power down bit . Set 1 oleh program untuk masuk konfigurasi power down .
0	IDL	Idle mode bit . Set 1 oleh program untuk masuk konfigurasi idle mode .

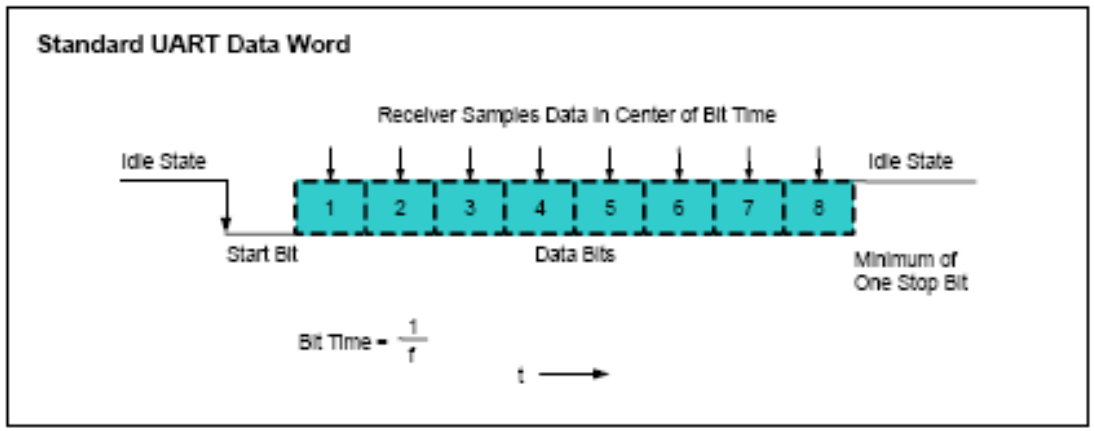
Mode Komunikasi Data Serial

Serial Data Mode 0. Shift Register



Jika bit SM1 & SM0 dalam SCON adalah 00, menyebabkan SBUF dapat menerima atau mengirim data 8 bit melalui pin RXD. Pin TXD digunakan sebagai jalur clock. Baudrate tetap yaitu 1/12 frekuensi osilator.
Ketika mengirim data, data digeser keluar pin RXD setelah satu pulsa clock. Data akan berubah ketika clock dalam fase falling edge / transisi dari haigh ke low.
Ketika menerima data dari pin RXD, data harus disinkronkan dengan pulsa clock yang dihasilkan pada TXD.

Serial Data Mode 1 Standard UART



Ketika SM0 dan SM1 adalah 01, SBUF menjadi 10-bit full-duplex receiver/transmitter yang dapat menerima dan mengirim data pada waktu yang sama. Pin RXD menerima semua data dan Pin TXD mengirim semua data.

Pengiriman data diawali dengan start bit, disusul dengan 8 bit data (Least Significant Bit / LSB terlebih dahulu) dan diakhiri dengan stop bit. Interrupt flag TI akan 1 setiap kali 10 bit dikirim.

Pengiriman data dimulai ketika start bit diterima, disusul dengan 8 bit data dan berakhir dengan diterimanya stop bit. Data 8 bit disimpan dalam SBUF dan stop bit disimpan pada RB8 dalam SCON. Interrupt flag RI akan 1 setiap kali 10 bit diterima.

Baud Rate Mode 1

Baudrate ditentukan dengan timer 1 yang dioperasikan dalam mode 8 bit autoreload dengan persamaan sbb :

$$f_{baud} = \frac{2^{SMOD}}{32d} x \frac{\text{oscillator frequency}}{12d \times [256d - (TH1)]}$$

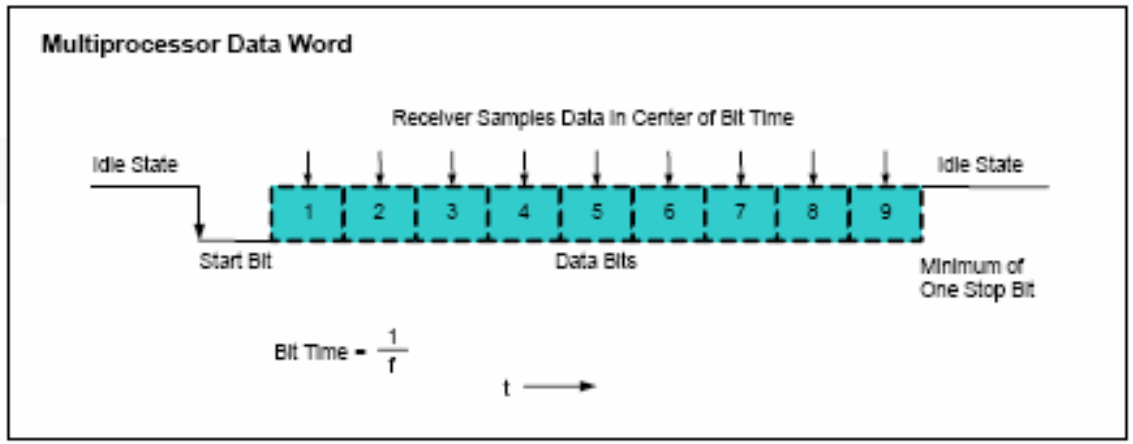
SMOD adalah control bit dalam PCON dan dapat bernilai 0 atau 1.
Jika timer 1 tidak bekerja pada mode 2, maka baudrate adalah :

$$f_{baud} = \frac{2^{SMOD}}{32d} x(\text{timer 1 overflow frequency})$$

Jika diinginkan baudrate-nya standar, maka harus menggunakan crystal dengan frekuensi 11.0592 MHz.
Untuk mendapatkan baudrate standar 9600 hertz maka nilai TH1 dapat dihitung dengan cara :

$$TH1 = 256d - \left(\frac{2^0}{32d} x \frac{11.0592 \times 10^6}{12 \times 9600d} \right) = 253d = 0FDh$$

Serial Data Mode 2 Multiprocesor Mode



Mode 2 sama dengan mode 1, tetapi jumlah data yang dikirim adalah 11 bit, dimulai dengan start bit, 9 bit data dan diakhiri dengan 1 bit stop. Data ke-9 disimpan pada TB8 dalam SCON ketika proses pengirman dan disimpan dalam RB8 ketika proses penerimaan.
Baudrate-nya adalah :

$$f_{baud} = \frac{2^{SMOD}}{64d} \times \text{oscillator frequency}$$

Serial Data Mode 3.

Mode 3 sama dengan mode 2 kecuali baudrate-nya sama dengan pada mode 1.

Contoh Data Serial

Contoh 1. Pengiriman Data

```
                $MOD51
                ORG     0000H
                LJMP    MULAI
                ORG     0100H
MULAI:          CALL    INITSERIAL
LOOP:           MOV     R0,#15
                MOV     DPTR,#TEXT
LOOP1:          CLR     A
                MOVC    A,@A+DPTR
                CALL    KIRIM
                CALL    DELAY
                DJNZ    R0,NEXT
                SJMP    LOOP
NEXT:           INC     DPTR
                SJMP    LOOP1
INITSERIAL:
                MOV     SCON,#52H
                MOV     TMOD,#21H
                MOV     TH1,#0FDH
                SETB    TR1
                RET
KIRIM:          JNB     TI,$
                CLR     TI
                MOV     SBUF,A
                RET
DELAY:          MOV     R5,#0FFH
                MOV     R6,#0FFH
                MOV     R7,#3
DLY:            DJNZ    R5,DLY
                DJNZ    R6,DLY
                DJNZ    R7,DLY
                RET
;-------- MESSAGES --------
;               012345678901234
TEXT:           DB      ' HELLO WORLD ! '
                END
```

Contoh 2. Penerimaan Data

```
                $MOD51
                ORG     0000H
                LJMP    MULAI
                ORG     0100H
MULAI:          CALL    INITSERIAL
LOOP:           CALL    TERIMA
                MOV     P1,A
                CALL    DELAY
                SJMP    LOOP
INITSERIAL:
                MOV     SCON,#52H
                MOV     TMOD,#21H
                MOV     TH1,#0FDH
                SETB    TR1
                RET
TERIMA:         JNB     RI,$
                CLR     RI
                MOV     A,SBUF
                RET
DELAY:          MOV     R5,#0FFH
                MOV     R6,#0FFH
                MOV     R7,#3
DLY:            DJNZ    R5,DLY
                DJNZ    R6,DLY
                DJNZ    R7,DLY
                RET
                END
```