BAB II

TINJAUAN PUSTAKA

2.1 Landasan teori

2.1.1 Struktur pemrograman assembley untuk AT89C51

2.1.1.1 Program sumber assembly

Program sumber assembly merupakan program yang ditulis oleh pembuat program berupa kumpualan baris-baris perintah dan biasanya di simpan dengan ektension.ASM. Program ini ditulis dengan menggunakan perangkat lunak teks editor seperti notepad atau Editor DOS.

Program assembly terdiri atas beberapa bagian yang dijelaskan seagai berikut:

- **Label**

  Label sangat berguna untuk memberi nama pada alamat-alamat yang dituju karena pemberian label pada suatu lebih bersifat relatif. Untuk membuat suatu label, ada beberapa persyaratan yang harus dipenuhi, antara lain :

  1. Harus diawali dengan huruf .
  2. Tidak diperbolehkan adanya label yang sama dalam suatu progaram.
  3. Maksimal 16 karakter.
  4. Tidak diperbolehkan adanya karakter spasi dalam label.

- **Mnenonic**

  Mnemonic atau juga biasanya disebut kode operasi (Opcode) adalah kode-kode yang akan dikerjakan pada program assembler yang ada pada komputer

atau pun mikrokontroler. Kode operasi yang dikerjakan oleh mikrokontroler merupakan perintah-perintah atau intruksi-intruksi yang sangat bergantung dengan jenis mikrokontroler yang digunakan. Contoh, untuk keluarga MCS-51 digunakan MOV,MOVX,MOVC,ADD dan lain-lain.

Sedangkan kode operasi yang dikerjakan oleh program assembler yang ada pada komputer sangat bergantung pada program assembler yang digunakan. Contoh, ORRG,EQU,BD dan lain-lain.

- **Operand**

Operand merupakan pelengkap dari mnemonic. Jumlah operand yang dibutuhkan oleh sebuah mnemonic tidak selalu sama, sebuah mnemonic tidak memiliki tiga, dua, satu atau bahkan tidak memiliki operand seperti contoh dibawah ini.

```
CJNE    A, B, Tak_sama      ; memerlukan 3 operand
MOV     R1, A               ; memerlukan 2 operand
INC     A                   ; memerlukan 1 operand
NOP                         ; tidak memerlukan operand
```

- **Komentar**

Bagian komentar tidak mutlak ada dalam sebuah program, namun bagian ini sering kali digunakan untuk menjelaskan proses-proses kerja atau pun catatan-catatan tertentu pada bagian-bagian program. Bahkan pembuat program sering kali membutuhkannya untuk mengingat kembali jalannya program rancagnannya.

Penggunaan komentar biasanya diawali denga tanda “;” dan dapat diletakkan pada bagian manapun dari suatu program.

2.1.1.2 Sistim Pengalamatan

Dalam pembuatan program pada mikrokontroler terdapat beberapa jenis sistem pengalamatan yang perlu diketahui, antara lain :

- **Pengalamatan Langsung**

  Proses pengalamatan ini terjadi pada sebuah perintah ketika nilai operand merupakan data yang akan diproses. Biasanya operand tersebut diawali dengan tanda ‘#’ seperti conyoh berikut :

  MOVA, #05H

- **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 merupakan salah satu I/O dari AT89C51 yang mempunyai alamat 20H. Perintah pada contoh diatas selain mengirimkan data ke akumulator port 0 juga merupakan perintah pemindahan data dari akumulator ke alamat 80H sehingga dapat juga ditulis MOV 80H, A.

- **Pengalamatan tidak Langsung**

  Proses pengalamatan ini terjadi pada sebuah perintah ketika salah satu operand merupakan register berisikan alamat data yang akan di isi atau dipindahkan. Pengalamatan jenis ini biasa digunakan untuk melakukan

penulisan, pemindahan atau pembacaan beberapa data dalam lokasi memori yang mempunyi urutan beraturan.

Jika proses ini dilakukan dengan menggunakan pengalamatan langsung maka jumlah baris program yang di perlukan akan cukup panjang. Contohnya pengalamatan data 08H pada alamat 50H sampai 53H.

```
MOV   50H, #08H
MOV   51H, #08H
MOV   52H, #08H
MOV   53H, #08H
```

Dengan digunakan pengalamatan tidak langsung, maka listing di atas dapat di rubah menjadi :

```
    MOV   R0, #08H
Loop:
    MOV   @R0, #08H
    INC     R0, #08H
    CJNE  R0, #53H, Loop
```

AT89C51 mempunyai sebuah register 16 bit (DPTR) dan dua buah register 8 bit (R0, R1) yang dapat digunakan untuk melakukan pengalamatan tidak langsung. Contoh-contoh pengalamatan tidak langsung :

```
MOV         @R0, A
MOV         A, R1
MOVX        @DPTR, A
MOVC        A, @A + DPTR
```

- **Pengalamtan Kode**

Pengalamatan kode merupakan pengalmatan ketika operand merupakan alamat dari intruksi jump dan call (ACALL, JMP, LJMP dan LCALL). Biasanya operand tersebut menunjuk ke suatu alamat yang diberi lebel sebelumnya seperti pada contoh berikut ini :

```
        ACALL           DELAY
        ..................................................
DELAY :
        MOV     R5, #05H
        MOV     R4, 0FFH
        DJNZ    R4, $
        DJNZ    R5, DEL1
        RET
```

Pada listing diatas, program ACALL DELAY mempunyai operand yang menuju ke Label Delay sehingga pada saat perintah ini dijalankan, program akan melompat kelokasi memori yang di beri Lebel Delay.

- **Pengalamatan Bit**

Proses pengalamatan ketika proses menuju ke alamat pada RAM Internal atau pun Register fungsi khusus yang mempunyai kemampuan pengalamatan secara bit (*bitaddressable*).

Berdasarkan penulisannya, pengalamatan ini terdiri atas beberapa macam yaitu sebagai berikut :

- ✓ **Langsung menunjuk ke alamat bit**

Contoh :

```
SETB    0B0H
```

Perintah ini memberikan logika 1 pada bit di alamat B0H dengan pengalamtan secara bit.

- ✓ **Menggunakan operand titik**

Contoh :

```
SETB    P3.0
```

Perintah ini memberikan logika 1 pada bit ke 0 dari port 3, bit tersebut terletak di alamat B0H dengan pengalamtan secara bit.

- ✓ **Menggunakan lambang Assembler secara standard**

Contoh :

```
SETB    RXD
```

Perintah ini memberikan logika 1 pada kai RXD yang terletak pada bit 0 dari port 3.

- ✓ **Menggunakan Assembler secara bebas**

Contoh :

```
PLAY    BIT   P1.7
```

Perintah ini memberikan logika 1 pada bit play yang sebelumnya didefinisikan sebagai bit P1.

2.1.1.3 Ekspresi Assembler

Dengan ekspresi assembler, sebuah operand dapat di ekspresikan dalam bermacam-macam bentuk sesuai keinginan pembuat program. Hal ini membuat pembuat program lebih mudah.

- **Basis Bilangan**

  Mikroprosesor membuat akhiran ‘B’ untuk ekspresi biner, ‘Q’ untuk ekspresi octal, ‘D’ untuk ekspresi desimal dan ‘H’ untuk ekspresi heksa.

- **Operan Logika**

  Ekspresi ini digunakan untuk mempermudah pembuatan program dalam pemberian nilai pada operand yang memerlulkan proses operasi logika terlebih dahulu. Operand-operand tersebut terdiri dari :

  AND untuk operasi logika AND

  OR untuk operasi OR

  NOT untuk operasi logika komplemen

  XOR untuk operasi EXOR

2.1.1.4 Assembler Directive

Pengarah assembler merupakan mnemonic yang akan di proses oleh program assembler. Berikut ini adalah pengarah assembler yang bisa digunakan untuk program-program assembler.

- **Kontrol Kondisi Assembler**
  - ✓ **ORG (Origin)**

    ORG digunakan untuk nenunjukan suatu lokasi memori tempat intruksi atau perintah yang ada dibawahnya disimpan. Bentuk ORG adalah:

    ORG ekspresi

    Contoh :

    ORG 2000H

```
MOV          DPTR, #4000H
```

Disini, perintah MOV DPTR, #4000H yang berada di bawah ORG 2000H akan terletak dialamat 2000H.

- **Definisi Lambang**
  - ✓ **EQU (Equate)**

EQU digunakan untuk mendefinisikan sebuah simbol atau lambang secara bebas.

Contoh :

```
Bilangan          EQU    50H
MOV               A, # Bilangan
```

Pada perintah diatas, akumulator diisi dengan konstanta ‘Bilangan’. Konstanta ini telah di definisikan sebelumnya dengan nilai 50H dengan menggunakan assembler directive EQU.

- ✓ **BIT**

Pengarah bit digunakan untuk mendefinisikan suatu lambang yang menunjuk ke suatu lokasi bit pada memori yang dapat dialamatkan secara pengalamatan bit.

Contoh :

```
Flag    Bit      0
```

Pada perintah diatas, lambang Flag menunjuk kelokasi 0 secara pengalamatan bit.

- **Pengarah Pilihan Segmen**
  - ✓ **CSEG (Code Segmen)**

Digunakan untuk memilih lokasi memori program.

- ✓ **DSEG (Data Segmen)**

  Digunakan untuk memilih lokasi memori RAM Internal.

- **Penyediaan Memori dan Pengarah Penyimpanan**

  - ✓ **DB (Define Byte)**

    DB digunakan untuk memberi nilai tertentu pada memori di lokasi tersebut.

    Contoh :

    ```
    ORG     3000H
    DB      50H, 51H
    ```

    Pengarah assembler directive BD terletak dibawah ORG 3000H oleh karena itu, data 50H dan 51H akan menempati lokasi dialamat 3000H dan 3001H.

  - ✓ **DS (Define String)**

    Pengarah assembler ini selalu diikuti dengan angka dan berfungsi untuk menyediakan tempat sebesar angka tersebut mulai dari alamat assembler directive ini berada. Tempat yang disediakan selalu terletak pada RAM Internal.

    Contoh :

    ```
    DSEG
    ORG         50H
    MEMORI :    DS   2
    ```

Pada contoh diatas, pengarah assembler ini terletak dialamat 50H dari RAM Internal. Oleh karena itu, mulai dari alamat 50H tersedia tempat sebesar 2 byte yang menempati alamat 50H dan 51H.

2.1.1.5 Organisasi Program

Sebuah program dapat ditelusuri dengan mudah sebaiknya dirancang secara konsisten berdasarkan struktur organisasi berikut :

- Kumpulan Pengarah EQU
- Perintah – perintah Inisialisasi
- Program Utama
- Subroutine – subroutine
- Definisi Konstanta data (DB, DW)
- Lokasi data RAM yang dinyatakan dengan pengarah DS

Kumpulan Pengarah EQU

Pengarah – pengarah EQU disertai komentar yang menjelaskan lebih detail mengenai lambang yang diarahkan oleh EQU tersebut, jika perlu.

Contoh :

ROM 4000H ; Alamat awal program dalam ROM ditentukan di alamat 4000H.

Perintah – Perintah Inisialisasi

Perintah – perintah ini biasanya digunakan untuk memberi nilai awal sebuah variable atau melakukan inisialisasi pada perangkat keras dan akan lebih mudah ditelusuri jika ditambahkan komentar yang menjelaskanproses inisialisasi tersebut.

Contoh :

```
P1.0        EQU   Relay
            CLR   Relay   ; relay diaktifkan ( aktif low )
```

Program Utama

Program Utama merupakan bagian besar dari program yang dirancang yang biasanya berupa sebuah putaran besar.

Contoh :

```
START :     Acall   Ambil keyboard      ; Ambil data dari keyboard
            Acall   Kirim LCD           ; Kirim data ke LCD
            Acall    Kirim Printer      ; Kirim data ke printer
            Ajmp   Start
```

Subroutine – subroutine

Sekelompok pernyataan yang sering kali dilakukan berulang – ulang, sebaiknya diletakkan dalam sebuah subroutine. Hal ini digunakan untuk memperingkas rancangan program yang dibuat.

Contoh :

Sub Routine untuk inisialisasi LCD

```
LCDINIT :
        CLR         P1.5
        CLR         TR1
        MOV         A, #38H
        CALL        CONTROLINIT
        MOV         A, #86H
```

```
        CALL    CONTROLINIT
LCDCLR :
        MOV     A, #0CH
        CALL    CONTROLINIT
        MOV     A, #01H
        CALL    CONTROLINIT
        MOV     A, #02H
        CALL    CONTROLINIT
        RET
```

Definisi Konstanta Data (DB atau DW)

Definisi Konstanta sebaiknya diletakkan terpisah dari subroutine ataupun program utama sehingga lebih mudah untuk mencarinya.

Lokasi Data RAM yang dinyatakan dengan Pengarah DS

Jika pada bagian – bagian yang dijelaskan sebelumnya terletak pada bagian Code Segment, bagian data yang terletak pada RAM Internal terletak pada bagian Data Segment. Bagian ini biasanya berisi variable – variable yang digunakan oleh program utama maupun subroutine.

Contoh :

```
ORG   30H
PENAMPUNG       DS      3
```

2.1.2 Operasi Timer

AT89C51 mempunyai 2 buah timer yaitu Timer 0 dan Timer 1 yang keduanya dapat berfungsi sebagai Counter maupun sebagai Timer. Secara fisik

sebetulnya timer juga merupakan rangkaian T Flip – flop yang dapat diaktifkan dan dinon – aktifkan setiap saat. Perbedaan terletak pada sumber clock dan aplikasinya. Jika timer mempunyai sumber clock dengan frekuensi tertentu yang sudah pasti sedangkan counter mendapat sumber clock dari pulsa yang hendak dihitung jumlahnya. Aplikasi dari counter atau penghitung biasa digunakan untuk aplikasi menghitung jumlah kejadian yang terjadi dalam periode tertentu sedangkan timer atau pewaktu biasa digunakan untuk aplikasi menghitung lamanya suatu kejadian yang terjadi.

Kedua Timer pada AT89C51 masing – masing mempunyai 16 bit Counter yang mampu diatur keaktifan maupun mode operasinya, direset dan diset dengan harga tertentu. Untuk mengatur Timer ini AT89C51 mempunyai 6 buah Special Function Register.

- **Timer Mode Register (TMOD)**

  Register TMOD berupa 8 bit yang terletak pada alamat 89H degan fungsi setiap bitnya adalah sebagai berikut:

  GATE : Timer akan berjalan jika bit ini di set dan di INT 0 (untuk timer 0) atau INT 1 (untuk timer 1) berkondisi high.

  C/T : 1 = Counter.
  2 = Timer

  M1 & M2 : Untuk melihat mode Timer.

- **THx dan TLx**

AT89C51 mempunyai 2 buah Timer, yaitu Timer 0 dan Timer 1 dimana setiap Timer terdiri atas 16 bit Timer yang masing-masing tersimpan dalam 2 buah register yaitu THx untuk Timer Haigh Byte dan TLx untuk Timer Low Byte.

TH 0 : Timer 0 High Byte terletak pada alamat 8AH

TL 0 : Timer 0 Low Byte terletak pada alamat 8BH

TH 1 : Timer 1 High Byte terletak pada alamat 8CH

TL 1 : Timer 1 Low Byte terletak pada alamat 8DH

- **Timer Control Register (TCON)**

Timer ini hanya mempunyai 4 bit saja, yaitu TCON.4, TCON.5, TCON.6 dan TCON.7 saja yag mempunyai fungsi berhubungan dengan Timer.

Register ini bersifat bit *Addressable* sehingga Bit TF1 dapat disebut TCON.7, TR 1 sebagai TCON.6, TF 0 sebagai TCON.5, TR 0 sebagai TCON.4, IE 1 sebagai TCON.3, IT 1 sebagai TCON.2, IE 0 sebagai TCON.1, IT 0 sebagai TCON.0

TCON.7 atau TF : Timer 1 oferflow Flag yang akan di set jika Timer oferflow. Bit ini dapat di *clear* oleh software atau oleh harware pada saat program menuju ke alamat yang ditunjuk oleh vektor interrupt.

TCON6 atau TR1 : 1 = Timer 1 aktif

0 = Timer 1 nonaktif

TCON5 atau TF0 : Sama dengan TF1.

TCON4 atau TR0 : Sama dengan TR1.

TCON3 hingga TCON 0 akan dibahas pada bagian Interrupt.

- **Mode Timer**

Timer AT89C51 mempunyai 4 buah mode timer dimana setiap mode mempunyai masing-masing fungsi. Penentuan mode kerja dari timer dilakukan dengan melakukan inisialisasi pada register TMOD seperti dijelaskan pada bagian berikut:

- Mode 0

Pada mode ini, timer bekerja dengan 13 bit timer ketika overflow terjadi saat terjadi perubahan kondisi dari 13 bit yang tersimpan diregister TLx dan THx (x=0 untuk timer 0 dan x=1 untuk timer 1) menjadi logika 0 setelah sebelumnya mencapai logika 1. Pada amplikasi sebagai counter hal ini terjadi saat counter kembali menghitung kembali dari awal. Bit TFx akan berlogika 1 pada saat kondisi overflow terjadi.

- Mode 1 (16 bit timer)

Pada mode 1, timer berfungsi sebagai 16 bit timer yang akan menghitung naik mulai dari 0000 H hingga FFFF H. Hasil dari perhitungan tersimpan pada register TLx untuk low byte dan THx untuk high byte.

Jika perhitungan sesudag mencapai FFFF H, timer akan kembali menghitung dari 0, pada saat ini timer flag (TFx) akan set.

- Mode 2

Pada mode ini, timer akan bekerja dalam mode 8 bit dimana nilai tersimpan dalam TLx. Register THx berisi nilai isi ulang (Reload Value) yang akan dikirim ke register TLx setiap terjadi over flow.

- Mode 3

Pada mode ini, AT8C51 bagaikan memiliki 3 buah timer. Timer 0 terpisah menjadi 2 buah bit timer yaitu TL0 dengan TF0 sebagai over flow flag dan TH0 dengan TF1 sebagai over flow flag. Sedangkan timer1 tetap berfungsi sebagai 16 bit timer.

Pada saat timer 1 berada pada mode3, timer ini akan berhenti hingga mode kerja timer1 diubah menjadi mode lain. Oleh karena bit TF1 digunakan oleh TH0 sebagai over flow flag, maka bit ini tidak dapat digunakan selama timer 0 masih berada pada mode3.

- Cara Kerja Timer

Operasi dari timer merupakan sumber clock yang didapat dari internal maupun eksternal. Jika timer menggunakan sumber clock dari eksternal, pin T0 (P3.4) berfungsi sebagai input clock. Untuk menjadikan sumber clock eksternal sebagai sumber clock timer maka bit C/T dari register TMOD harus diset atau berkondisi high. Jika bit C/T berkondisi high, saklar akan menghubungkan sumber clock timer ke pin Tx (T0 untuk timer 0 dan T1 untuk tomer 1).

Jika digunakan sumber clock internal, input clock tersebut berasal dari oscillator yang telah dibagi 12. Untuk ini bit C/T dari register TMOD harus

diclear atau berkondisi low sehingga saklar akan menghubungkan sumber clock timer ke oscillator yang telah dibagi 12.

Untuk mengaktifkan timer dapat dilakukan melalui hardware maupun software. Timer baru akan aktif setelah mendapat sumber clock dan saklar SPST yang terletak antara saklar yang dikontrol oleh C/T dan timer terhubung sehingga sinyal clock dari sumer clock akan mengalir ke timer. Sedangakn saklar tersebut terhubung jika mendapat logika high dari output gerbang AND. Sesuai dengan tabel kebenaran gergang AND, output dari gerbang AND hanya akan berlogika high jika kedua inputnya berlogika high pula. Jika ada salah satu inputannya yang berlogika low, output akan berlogika low pula. Untuk pengaturan timer melalui software, keaktifan timer hanya dilakukan oleh kondisi bit TR0 saja. Oleh karena itu output dari gerbang OR yang terhubung ke input yang lain dari gerbang AND harus berlogika high. Dengan demikian, jika TR0 berlogika high, output dari gerbang AND akan berlogika high dan timer akan aktif dan sebaliknya jika TR0 berlogika low maka output gerbang AND akan berlogika low dan timer akan berhenti.

Agar output dari gerbang OR berlogika high, sesuai dengan tabel kebenaran gerbang OR, cukup salah satu dari inputannya saja yang berlogika high. Dengan demikian, output akan berlogika high pula. Untuk pengaturan timer melalui sofware, bit gate harus berkondisi low sehingga hasil inversnya yang merupakan salah stu input dari gerbang OR berlogika high. Hal ini membuat output gerbang OR selalu berlogika high walau apapun yang terjadi pada pin INTx (INT 0 untuk timer 0 dan INT 1 untuk timer 1).

Untuk pengaturan Timer dengan hardware, Pin INTx berfungsi sebagai penentu. Oleh karena itu, Bit TRx harus berlogika high, agar pengaturan Timer ditentukan oleh output dari gerbang OR. Agar kondisi output dari Gerbang OR ditentukan oleh Pin INTx maka Bit Gate harus berkondisi high.

Jadi kesimpulannya untuk pengaturan Timer melalui software maka bit penentu keaktifan adalah TRx sedangkan kondisi dari Bit Gate harus berlogika 0. Untuk pengaturan Timer dengan hardware maka penentu keaktifan adalah INTx sedangkan kondisi Bit Gate dan TRx harus berlogika 1.

2.1.3 Operasi Interupsi

Interupsi adalah suatu kejadian atau peristiwa yang menyebabkan mikrikontroller berhenti sejenak untuk melayani interupsi tersebut. Program yang dijalankan pada saat melayani interupsi disebut *Interrupt Routine Service* (Rutin Layanan Interupsi). Analoginya adalah sebagai berikut, seseorang sedang mengetik laporan, mendadak telepon berderingdan menginterupsi orang tersebut sehingga menghentikan pekerjaan mengetik dan mengangkat telepon. Setelah pembicaraan dalam hal ini merupakan analogi dari *Interrupt Routine Service* selesai, orang tersebut kembali meneruskan pekerjaan mengetiknya.

Demikian pula pada system mikrokontroller yang sedang menjalankan programnya, saat terjadi interupsi, program akan berhenti sesaat, melayani interupsi tersebut dengan menajalankan program yang berada pada alamat yang ditunjuk oleh vektordari interupsi yang terjadi hingga selesai dan kembali meneruskan program yang terhenti oleh interupsi tadi.

- **Pengaktifan Interupsi**

Dalam suatu kondisi, dapat juga dibutuhkan suatu program yang sedang berjalan tidak boleh diinterupsi. Untuk itu, 89C51 mempunyai 5 buah interupsi yang masing-masing dapat ditetapkan enable ataupun disable satu per satu pengaturan keaktifan interupsi dilakukan pada *Interrupt Enable Register* (Register Pengaktif Interupsi) yang terletak pada alamat A8H.

EA : Menonaktifkan semua interupsi jika bit ini clear. Jika bit ini clear, maka apapun kondisi bit lain dalam register ini, semua interupsi tidak akan dilayani, oleh karena itu untuk mengaktifkan salah satu interupsi, bit ini harus set.

ES : Mengaktif/nonaktifkan interupsi port serial, set = aktif, clear = nonaktif.

Jika Interupsi Port Serial aktif, interupsi akan terjadi setiap data yang masuk ataupun keluar melalui serial port yang membuat Flag RI *(Receive Interrupt Flag)* ataupun TI *(Transmit Interrupt Flag)*.

ET1 : Mengaktif/nonaktifkan interupsi tiemr 1 Interrupt, set = aktif, clear = non aktif.

Jika interupsi ini aktif, interupsi akan terjadi pada saat Timer 1 overflow.

EX1 : Mengaktif/nonaktifkan interupsi eksternal 1, set = aktif, clear = non aktif

Jika interupsi ini enable, interupsi akan terjadi pada saat terjadi logika 0 pada INT1

ET0 : Mengaktif/nonaktifkan interupsi eksternal 0, set = aktif, clear = non aktif

Jika interupsi ini aktif, interupsi akan terjadi pada saat Timer 0 overflow.

ET0 : Mengaktif/nonaktifkan interupsi eksternal 0, set = aktif, clear = non aktif

Jika interupsi ini aktif, interupsi akan terjadi pada saat terjadi logika 0 pada INT0.

- **Prioritas Interupsi**

Pada aplikasi yang menggunakan lebih dari satu interupsi, dapat saja terjadi interupsi lain pada saat sebuah rutin interupsi sedang dijalankan. Untuk menentukan apakah sebuah interupsi dapat menginterupsi saat sebuah rutin interupsi lain sedang berjalan bergantung pada prioritas dari interupsi tersbut.

Prioritas interupsi pada kondisi standard selalu tersusun pada prioritas sesuai pada tabel 2.1

Tabel 2.1 Prioritas Interupsi

Prioritas	Jenis Interupsi
1	Interupsi Eksternal 0
2	Interupsi Timer 0
3	Interupsi Eksternal 1
4	Interupsi Timer 1
5	Interupsi Serial

Interupsi yang mempunyai prioritas lebih tinggi, tidak akan dapat diinterupsi oleh interupsi lain yang mempunyai prioritas lebih rendah.

Prioritas tersebut dapat diubah dengan melakukan inisialisasi pada *Register Interrupt Priority (IP)*.

PT2 : Priority Timer Interrupt 2, bit yang mengatur prioritas interupsi timer 2 (89C52).

PS : Priority Serial Interrupt, bit yang mengatur prioritas interupsi serial.

PT1 : Priority Timer Interrupt 1, bit yang mengatur prioritas interupsi timer 1.

PX1 : Priority External Interrupt 1, bit yang mengatur prioritas interupsi external 1.

PT0 : Priority Timer Interrupt 0, bit yang mengatur prioritas interupsi timer 0.

PX0 : Priority External Interrupt 0, bit yang mengatur prioritas interupsi external 0.

Prioritas yang dapat diatur oleh register ini adalah prioritas tinggi dan prioritas rendah. Untuk membuat sebuah interupsi menjadi prioritas tinggi maka bit pengatur prioritas interupsi tersebut harus diubah menjadi logika 1. Interupsi eksternal 0 yang mempunyai prioritas paling tinggi pada kondisi standard, akan mempunyai prioritas yang lebih rendah dari Interupsi Serial jika kondisi bit PX0 berlogika 0dan kondisi bit PS berlogika 1.

Jika kedua bit pengatur prioritas berlogika 1, prioritas interupsi akan tersusun berdasarkan prioritas pada kondisi standard, yaitu interupsi, eksternal 0 mempunyai prioritas lebih tinggi dari interupsi serial.

2.1.4 Perangkat Instruksi

Mikrokontroler Intel 89C51 memiliki 256 perangkat instruksi .seluruh instruksi dapat dikelompokkan menjadi 4 bagian yang meliputi instruksi 1 *byte* sampai 4 *byte*

Apabila *frekuwensi mikrokontroler* yang digunakan adalah 12 *MHZ*.kecepatan pelaksanaan instruksi akan bervariasi dari 1 hinga 4 *mikro* detik .Perangkat instruksi *mikrokontroler intel* 89C51 dapat dibagi menjadi 5 kelompok sebagai berikut :

- Instruksi *Transfer* Data

  Instruksi ini memindahkan data antara *register-register*,memori-memori,*register*-memori,antar muka-*register,*dan antar muka-memori

- Instruksi *Aritrmatika*

  Instruksi ini melakukan operasi *aritmatika* yang meliputi penjumlahan, pengurangan, penambahan satu *(increment),* pengurangan satu *(decrement),* perkalian dan pembagian

- Instruksi Logika dan Manipulasi Bit

  Melaksanakan operasi logika *AND, OR, XOR*, pertandingan,penggeseran dan komplemen data

- Instruksi Percabagan

  Instruksi ini mengubah urutan normal pelaksanaan suatu program.Dengan instruksi ini program yang sedang dilaksanakan akan mencabang ke suatu alamat tertentu ,instruksi percabangan dibedakan atas percabangan bersyarat dan percabagan tanpa syarat

- Instruksi *Stack, I/O,* dan Kontrol

  Instruksi ini mengatur penguanaan *stack,* membaca / menulis *port I/O*, serta pengontrolan-pengontrolan

- Perangkat Instruksi 89C51

  Perangkat instruksi mikrokontroler intel 89C51 secara lengkap adalah sebagai berikut :

  ACALL *Absolute Call Within 2K byte Page*

  Mnemonic : ACALL

  Operand : Code address

  Format : ACALL code address

  Keterangannya instruksi ini menyimpan isi yang telah dinaikkan dari pencacah program pada stack. Byte rendah PC ditempatkan ke stack terlebihdahulu. Alamat tujuan harus diantara 2K byte dari instruksi ACALL ini berada.

  ADD *Add Immediate Data*

  Mnemonic : ADD

  Operand : A Akumulator

  : data -256<= data <= +256

  Format : ADD A,# data

  Keterangan instruksi ini menambah 8 bit data langsung ke dalam isi akumulator dan meyimpan hasilnya pada akumulator.

  ADD *Add Indirect Data*

  Mnemonic : ADD

  Operand : A Akumulator

: Rr Register 0 <=r<=1

Format : ADD A, @Rr

Keterangan instruksi ini menambahkan isi data memori yang lokasinya di tunjukan oleh nilai register r ke isi akumulator dan menyimpan hasilnya dalam akumulator.

Contoh : ADD A, @R1

ADD *Add Register*

Mnemonic : ADD

Operand : A Akumulator

: Rr Register 0 sampai 7

Format : ADD A, Rr

Keterangan instruksi ini menambahkan isiregister r ke isi akumulator dan menyimpan hasilnya dalam akumulator.

Contoh : ADD A, R6

ADD *Add memori*

Mnemonic : ADD

Operand : A Akumulator

: Alamat data 0 <= alamat data,=256

Format : ADD A, alamat data

Keterangan instruksi ini menambahkan isi alamat data ke isi akumulator dan menyimpan hasilnya dalam akumulator.

Contoh :ADD A,32H

ADDC *Add Carry Plus Immediate Data to Accumulator*

Mnemonic : ADDC

Operand : A Akumulator

: data -256<= data <= +256

Format : ADD A, # data

Keterangan instruksi ini menambahkan isi carry flag (0 atau 1) kedalam isi akumulator. Dan langsung 8 bit ditambahkan ke akumulator.

Contoh :ADDC A, #0AFH

ADDC *Add Carry Plus Indirect Address to Accumulator*

Mnemonic : ADDC

Operand : A Akumulator

: Rr Register 0 <=r<=1

Format : ADD A, @Rr

Keterangan instruksi ini menambahkan isi carry flag (0 atau 1) dengan isi akumulator. Isi data memori pada lokasi yang di tunjukkan oleh register Rr ditambahkan dan hasinya disimpan di akumulator.

Contoh :ADDC A, @R1

ADDC *Add Carry Plus Register to Accumulator*

Mnemonic : ADDC

Operand : A Akumulator

: Rr Register 0<= r <=7

Format : ADDC A,Rr

Keterangan instruksi ini menambahkan isi carry flag dengan isi akumulator, isi register r ditambahkan dan hasilnya disimpan di akumulator.

Contoh :ADDC A, R7

ADDC *Add Carry Plus Indirect Address to Accumulator*

Mnemonic : ADDC

Operand : A Akumulator

: Alamat data 0 <= alamat data,=255

Format : ADD A, Alamat data

Keterangan instruksi ini menambahkan isi *carry flag* dengan isi akumulator, dari alamat data tertentu ditambahkan pula, dan hasilnya disimpan di akumulator.

Contoh :ADDC A, 25H

AJMP *Absolute Jump Within 2K Byte Page*

Mnemonic : AJMP

Operand : Alamat kode

Format : AJMP Alamat kode

Keterangan instruksi ini meletakan bagian bawah 11 bit dari pencacah program dengan 11 bit alamat yang dikodekan.

ANL *Logical AND Indirect Address to Accumulator*

Mnemonic : ANL

Operand : A Akumulator

: R Register 0 <=r<=1

Format : ANL A, @Rr

Keterangan instruksi ini meng-AND-kan isi memori yang lokasinya ditunjukan oleh isi register r dengan isi akumulator.

Contoh :ANL A, @Rr

ANL *Logical AND Immediate Data to Accumulator*

Mnemonic : ANL

Operand : A Akumulator

: data -256<= data <= +256

Format : ANL A, #data

Keterangan instruksi ini meng-AND-kan data 8 bit secara langsung dengan isi akumulator

Contoh :ANL A, #00001000B

ANL *Logical AND Register to Accumulator*

Mnemonic : ANL

Operand : A Akumulator

: R Register 0 <=Rr<=7

Format : ANL A, Rr

Keterangan instruksi ini meng-AND-kan i isi register r dengan isi akumulator.

ANL *Logical AND Memory to Accumulator*

Mnemonic : ANL

Operand : A Akumulator

: Alamat data 0 <= alamat data<,=255

Format : ANL A, Alamat data

Keterangan instruksi ini meng-AND-kan i isi alamat tertentu dengan isi akumulator.

2.2 Kerangka pemikiran

Variabel yang digunakan dalam tugas pendahuluan proyek akhir ini adalah intensitas cahaya. Tugas pendahuluan proyek akhir ini akan menggunakan intensitas cahaya yang telah ditetapkan sebagai masukan untuk menjalankan m*usic box* yang berbasis mikrokontroler, intensitas cahaya tersebut akan diterima oleh LDR yang kemudian akan diproses sebagai data dan dikirim ke *music box*. Penggunaan bahasa pemrograman assembley digunakan untuk memperoses jalannya masukan yang masuk kedalam mikrokontroler dan dikeluarkan sebagai data untuk pemilikan lagu di dalam *misic box.*