

Dasar Pemrograman Bahasa C pada Modul DST-51

(1)

Perkenalan

Pada awal munculnya mikrokontroler 8051, ada suatu anggapan bahwa bahasa assembly adalah satu-satunya pilihan yang paling efisien untuk memprogram mikrokontroler ini. Seolah-olah, keterbatasan memori internal yang dimiliki 8051 baik memori program maupun memori data, tidak memungkinkan untuk pembangunan program menggunakan bahasa tingkat tinggi (High Level Language).

Walaupun demikian, pengembangan *compiler* C untuk 8051 telah dimulai sejak tahun 1985. Dan sejak saat itu, usaha yang berkesinambungan dilakukan untuk memperbaiki efisiensi *Compiler* C untuk 8051. Sebagai hasilnya, beberapa tahun kemudian muncullah beberapa *Compiler* C yang disebut-sebut mempunyai efisiensi hampir sama dengan *Assembler*. Benarkah demikian??? Kita mesti mencobanya, baru dapat mengatakan ya dan tidaknya ∅

Bertolak dari sini, artikel ini akan mencoba membahas dasar pemrograman C pada mikrokontroler 8051 beserta contoh-contoh aplikasinya. Disini akan dipakai *compiler* SDCC untuk mendemokan. *Compiler* ini free.

Jika kita telah pernah menggunakan bahasa C untuk mikrokontroler / mikroprosesor selain 8051, maka yang pertama kali perlu kita ketahui adalah struktur memori 8051. 8051 mempunyai memori yang terpisah untuk program dan data. Untuk memori program, pilihan kita adalah salah satu dari memori program external atau memori program internal. Pilihan tersebut direpresentasikan dengan pemasangan logic pada kaki EA (kaki nomer 31). Pemasangan logic “0” pada kaki ini membuat 8051 mengeksekusi program yang ada di memori eksternal. Sebaliknya, pemasangan logic “1” membuat 8051 mengeksekusi program yang ada di memori internalnya.

Sedangkan untuk memori data, kita bisa menggunakan hanya memori RAM internal saja atau kedu-duanya, memori RAM internal dan eksternal. Pemilihan ini dilakukan saat

pendeklarasian variabel pada program. Berikut contoh pendeklarasian data 1 byte pada memori internal:

```
unsigned char x;
```

Dan contoh berikut adalah pendeklarasian data 1 byte pada memori RAM eksternal:

```
xdata unsigned char x;
```

Tipe-tipe data yang disupport oleh SDCC adalah:

Tipe Data	Jumlah Bit	Jumlah Byte	Nilai
<code>signed char</code>	8	1	-128 .. 127
<code>Unsigned char</code>	8	1	0 .. 255
<code>Enum</code>	16	2	-32768 .. 32767
<code>signed short</code>	16	2	-32768 .. 32767
<code>unsigned short</code>	16	2	0 .. 65535
<code>signed int</code>	16	2	-32768 .. 32767
<code>Unsigned int</code>	16	2	0 .. 65535
<code>signed long</code>	32	4	-2147483648 .. 2147483648
<code>Unsigned long</code>	32	4	0 .. 4294967295
<code>Float</code>	32	1	$\pm 1.175494E-38$.. $\pm 3.402823E+38$
<code>Bit</code>	1		0 .. 1
<code>Sbit</code>	1		0 .. 1
<code>Sfr</code>	8	1	0 .. 255

Tipe-tipe variabel diatas bisa dikelompokkan bersama membentuk suatu variable array, structure maupun union. Hal ini sangat bermanfaat untuk pembuatan program berbasis data (*database*). Kita akan lihat kelak bahwa pembuatan program yang memerlukan database akan sangat mudah dengan menggunakan bahasa C.

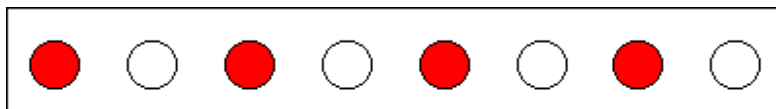
--- **** ---

Pada C, huruf kecil dan huruf besar sangat penting (*case sensitive*). Perlu diperhatikan bahwa semua kata kunci (*Keyword*) pada C harus huruf kecil. Awal program C adalah pernyataan:

```
main()  
{  
    .....  
    //Disini kita menulis program  
    .....  
}
```

Pernyataan ini memberitahu pada compiler bahwa disinilah program berawal. Tanda kurung () setelah kata **main** menunjukkan bahwa tidak ada argumen yang dilewatkan pada program ini. Sedangkan tanda '{' merupakan pernyataan awal program, dan tanda '}' merupakan pernyataan akhir program. Diantara tanda '{' dan tanda '}' inilah kita menuliskan program.

Misalkan kita akan membuat sebuah program sederhana untuk menyalakan 8 buah led yang terpasang pada port 1 dengan kombinasi:



P1.7 **P1.6**

P1.5 **P1.4** **P1.3** **P1.2** **P1.1** **P1.0**

Misalkan pemasangan led pada konfigurasi sink (LED menyala jika kaki port berlogika "0"), maka pertama kita cari kombinasi biner dari ke-8 LED tersebut, yakni: 10101010 biner atau AA heksa. Maka programnya:

```
1: sfr at 0x90 P1;  
2:  
3: void main(){
```

```
4:   P1 = 0xAA;  
5:   while (1);  
6: }
```

baris 1 pada potongan program diatas merupakan deklarasi variable bernama P1, bertipe sfr (*Special Function Register*) beralamat di 90 heksa. Tanpa pernyataan ini maka P1 tidak akan dikenali oleh *compiler*. Lalu timbul pertanyaan, apakah semua *special function Register* yang dimiliki 8051 (**P0, P2, P3, SCON, TI, RI, TH0, TL0**, dll) harus kita definisikan dulu sebelum kita pakai? Jawabnya adalah YA, tapi tidak harus kita yang mendefinisikannya, karena pada SDCC sudah tersedia file header 8051.h yang berisi deklarasi dari semua *Special Function Register* yang dimiliki 8051. Ketika file ini kita *include*-kan pada program kita, semua SFR 8051 akan dikenali tanpa harus kita deklarasikan sebagaimana contoh program diatas. Jadi baris 1 dari potongan program diatas bisa diganti :

```
1: #include <8051.h>;
```

Pernyataan pada baris 4 merupakan perintah untuk meng-outputkan logic 10101010 biner pada kaki port 1. Hal ini sepadan dengan perintah assembly:

```
Mov P1,#0AAh
```

dan baris 5 adalah perintah untuk berhenti di titik tersebut, sepadan dengan perintah assembly:

```
Sjmp $
```

Delta Electronic