BAB 3 MODE AKSES MEMORI, PENGGUNAAN VARIABEL DAN *STACK*

Pengaksesan memori yang dibicarakan dalam bab ini meliputi memori internal prosesor yang biasa disebut sebagai register dan memori di luar prosesor. Pada bagian pertama akan dibicarakan beberapa mode pengaksesan atau pengalamat kemudian dilanjutkan dengan penggunaan variabel dalam Bahasa Assembler serta bagian akhir penjelasan tentang *Stack*.

3.1. MODE PENGALAMATAN ATAU PENGAKSESAN

Seperti diketahui bahwa instruksi-instruksi Assembler ada yang membutuhkan operan baik satu maupun dua. Operan-operan ini dibutuhkan oleh instruksi yang bersangkutan, misalnya `MOV AX,BX`, pada instruksi terlihat dibutuhkannya dua operan yaitu register **AX** dan **BX**.

Prosesor saat menjalankan suatu instruksi bisa memperoleh operan dari register, dalam instruksi itu sendiri, suatu lokasi memori atau port I/O (keluaran/masukan). Secara garis besar terdapat 7 (tujuh) macam mode pengalamatan, yaitu:

1. Pengalamatan register (*register addressing*);
2. Pengalamatan segera (*immediate addressing*);
3. Pengalamat langsung (*direct addressing*);
4. Pengalamatan tak-langsung register (*register indirect addressing*);
5. Pengalamatan relatif dasar (*base relative addressing*);
6. Pengalamatan langsung terindeks (*direct indexed addressing*); dan
7. Pengalamatan dasar terindeks (*base indexed addressing*).

Masing-masing mode tersebut masing-masing memiliki ciri khas tersendiri, perhatikan tabel 3.1. Untuk dua mode pengalamatan yang pertama, register dan segera, tidak membutuhkan segmen karena operannya berupa register atau data langsung. Lima mode terakhir (mode 3 hingga 7) merupakan mode pengalamatan memori.

Tabel 3.1. Mode pengalamatan 80x86

Mode pengalamatan	Format operan	Register segmen
Register	Register	Tidak ada
Segera	Data	Tidak ada
Langsung	Pergeseran Label	DS DS
Tak-langsung register	[BX] [BP] [DI] [SI]	DS SS DS DS
Relatif dasar	[BX] + pergeseran [BP] + pergeseran	DS SS

Tabel 3.1 (lanjutan). Mode pengalamatan 80x86

Mode pengalamatan	Format operan	Register segmen
Langsung terindeks	[DI] + pergeseran	DS
	[SI] + pergeseran	DS
Dasar terindeks	[BX][SI] + pergeseran	DS
	[BX][DI] + pergeseran	DS
	[BP][SI] + pergeseran	SS
	[BP][DI] + pergeseran	SS

3.1.1. MODE PENGALAMATAN REGISTER DAN SEGERA

Pada mode pengalamatan register, prosesor akan mengambil operan dari suatu register, misalnya instruksi:

```
MOV  AX,CX
```

digunakan untuk menyalin isi register **AX** (16-bit atau word) ke register **CX** (16-bit juga), dengan kata lain menyamakan register **AX** dan **CX**.

Sedangkan pada mode pengalamatan segera, prosesor akan mengambil operan langsung dari instruksi yang bersangkutan, yaitu berupa data atau konstanta byte maupun word, misalnya instruksi:

```
ADD  AX,0034h
```

digunakan untuk menambahkan isi register **AX** dengan data 34h dan hasilnya disimpan di **AX** kembali.

3.1.2. MODE PENGALAMATAN LANGSUNG

Dalam mode pengalamatan memori dikenal adanya istilah **alamat efektif** atau *effective address*. Alamat efektif ini merupakan jarak dalam byte dari awal segmen hingga lokasi memori atau operan yang bersangkutan. Alamat efektif ini juga disebut sebagai **offset** bersama dengan **segmen** membentuk alamat fisik dengan persamaan:

Alamat fisik = offset + (segmen x 16)

segmen yang dikalikan 16 artinya digeser ke kiri 4-bit, dengan demikian jika dituliskan suatu **alamat logik** CS:IP = 2340:0100, maka alamat fisiknya adalah:

```
23400 – segmen digeser ke kiri 4 bit atau 1 digit heksa
 0100 – offset
--------+
23500
```

Dengan mode pengalamatan langsung, offset atau alamat efektif ada di dalam instruksi yang bersangkutan (menjadi satu dengan instruksi tersebut, sama seperti

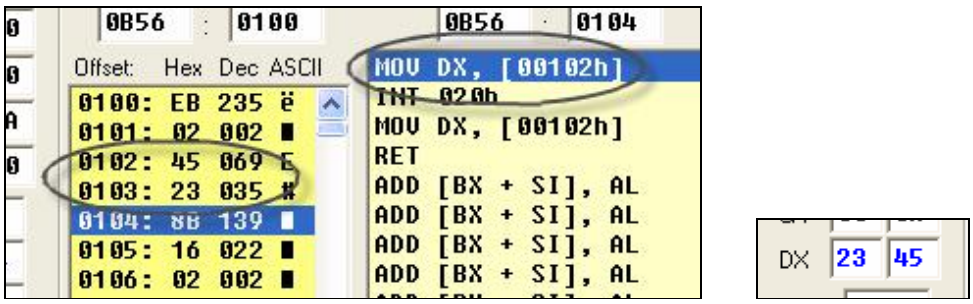
mode pengalamatan segera). Operan pengalamatan langsung biasanya berupa label, misalnya:

```
MOV  AX,TABEL
```

instruksi ini digunakan untuk menyalin isi lokasi memori yang ditunjuk oleh label TABEL (sebelumnya harus didefinisikan dulu) ke register AX. Perhatikan contoh program berikut ini:

```
; nama berkas: program0301.asm
; program contoh mode pengalamatan langsung
;
#make_COM#

ORG 100h

     JMP START ; mulai dari START

TABEL DW 2345h       ; definisikan TABEL

start:
     MOV DX,TABEL
     INT 20H
```

Ketik dan jalankan menggunakan emulator program ini. Perhatikan saat dijalankan per instruksi, **TABEL** diberikan lokasi **0102h**, sebagaimana ditunjukkan pada gambar 3.1, isi memori pada lokasi **0102h=45** dan **0103h=23**, dalam hal ini cara penyimpanannya menggunakan metode **Litle Endian**, artinya yang bobot digitnya kecil ditaruh belakangan (di lokasi berikutnya). Setelah selesai Anda jalankan, perhatikan bahwa sekarang register **DX** berisi data **2345h** (gambar 3.1 bagian kanan).



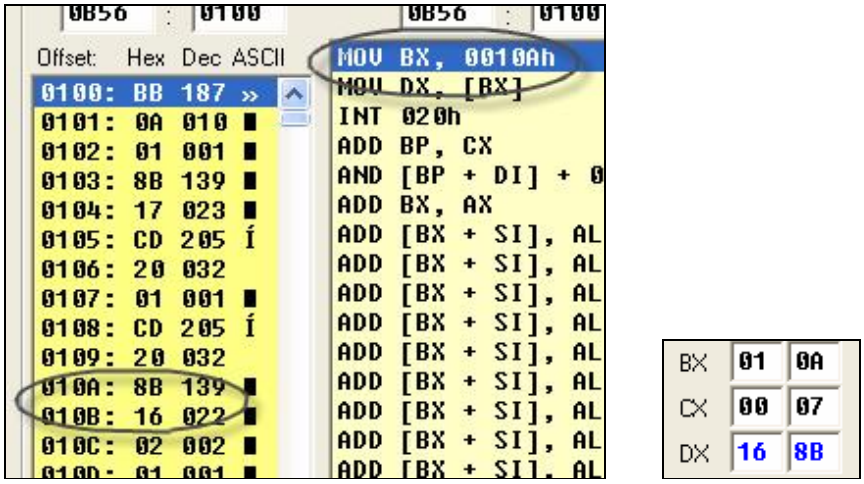
Gambar 3.1. Contoh mode pengalamatan langsung

3.1.3. MODE PENGALAMATAN TAK-LANGSUNG REGISTER

Pada mode pengalamatan tak-langsung register ini, alamat offset operan disimpan dalam register dasar BX, register penunjuk dasar BP atau register indeks SI atau DI. Perhatikan contoh program berikut ini:

```
; nama berkas: program0302.asm
; program contoh
; mode pengalamatan tak-langsung register
;
#make_COM#

ORG 100h

start:
     MOV BX,010AH          ; BX diisi dulu
     MOV DX,[BX]           ; mode tak-langsung register
     INT 20H
```

Perhatikan untuk contoh ini kita isi dulu register **BX** dengan nilai **010Ah**, kita tidak tahu pada lokasi **010Ah** tersebut berisi data atau nilai berapa kecuali setelah kita jalankan emulatornya, sebagaimana ditunjukkan pada gambar 3.2 sebelah kiri. Hasil eksekusi program ini berupa isi **DX** yang berubah menjadi isi memori lokasi **010Ah** yaitu 168Bh (gambar 3.2 bagian kanan).



Gambar 3.2. Contoh mode pengalamatan tak-langsung register

3.1.4. MODE PENGALAMATAN RELATIF DASAR

Pada mode pengalamatan ini, alamat efektif dihitung dengan cara menjumlahkan nilai pergerseran dengan isi register dasar **BX** atau **BP**. Perhatikan contoh program berikut ini:

```
; nama berkas: program0303.asm
; program contoh
; mode pengalamatan relatif dasar
;
#make_COM#

ORG 100h

start:
```

```
     MOV BX,010AH    ; BX diisi dulu
     MOV DX,[BX]+2   ; mode relatif dasar
     INT 20H
```

Seperti pada contoh sebelumnya, register **BX** diisi dengan nilai **010Ah** terlebih dahulu, kemudian lokasi memori yang ditunjuk oleh **[BX]+2** yaitu **010Ah+2** atau **010Ch** disalin ke register **DX**. Selain cara penulisan seperti pada contoh program tersebut, Anda bisa juga menuliskan dengan cara:

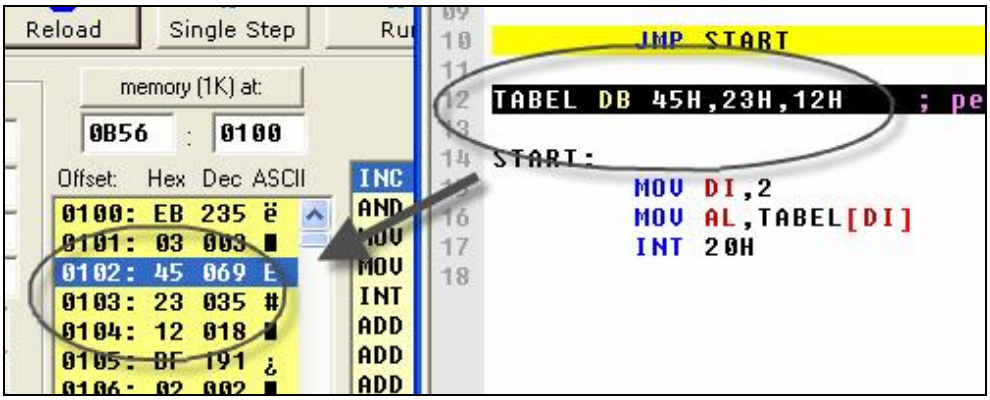
```
     MOV  DX,2[BX]        ; seperti ini atau
     MOV  DX,[BX+2]       ; seperti ini
```

3.1.5. MODE PENGALAMATAN LANGSUNG TERINDEKS

Dengan mode pengalamatan langsung terindeks, alamat efektif merupakan jumlah dari pergeseran dengan register indeks **SI** atau **DI**. Perhatikan contoh program berikut ini:

```
; nama berkas: program0304.asm
; program contoh
; mode pengalamatan langsung terindeks
;
#make_COM#

; COM file is loaded at CS:0100h
ORG 100h

     JMP START

TABEL DB 45H,23H,12H     ; pertama indeks-nya 0

START:
     MOV DI,2            ; disalin indeks ke-2
     MOV AL,TABEL[DI]    ; ke register AL
     INT 20H
```

Seperti pada contoh program yang pertama (**program0301.asm**), kita definisikan dulu sebuah data **TABEL**, yaitu **45h**, **23h** dan **12h**. Data pertama berindeks 0 dan akan disalin data yang ketiga atau berindeks 2 ke register **AL** (melalui register **DI** yang diisi 2). Melalui emulator terlihat bahwa TABEL diberikan lokasi 0102h hingga 0104h, sebagaimana ditunjukkan pada gambar 3.3. Hasil eksekusi program ini berupa isi register **AL** yang berubah menjadi **12h** (silahkan dicek pada emulator).



Gambar 3.3. Contoh mode pengalamatan langsung terindeks

3.1.6. MODE PENGALAMATAN DASAR TERINDEKS

Dalam mode pengalamatan ini (dasar terindeks), alamat efektif merupakan jumlahan register dasar, register indeks dan (opsional) pergeseran. Mode pengalamatan ini sangat cocok untuk akses data larik dua dimensi. Perhatikan contoh program berikut:

```
; nama berkas: program0305.asm
; program contoh
; mode pengalamatan dasar terindeks
;
#make_COM#

; COM file is loaded at CS:0100h
ORG 100h

     JMP START

TABEL DB 45H,23H,12H,56H,89H  ; pertama indeks-nya 0

START:
     MOV DI,2        ; disalin indeks ke-2
     MOV BX,1        ; salin indeks ke-1
     MOV AL,TABEL[DI+BX+1]    ; ke register AL
     INT 20H
```

Mirip dengan program sebelumnya (**program0304.asm**), sekarang **TABEL** kita perpanjang hingga 5 data (dari indeks 0 sampai 4). Register indeks yang digunakan adalah **DI** (dengan diisi 2), reister dasar yang digunakan adalah **BX** (dengan diisi 1) kemudian pergeseran diisi dengan 1, sehingga total indeks yang digunakan adalah 2+1+1 = 4 atau data ke-5. Setelah program dijalankan, perhatikan bahwa register **AL** akan berisi **89H** (perhatikan pada emulator).

3.2. VARIABEL DAN PENGGUNAANNYA

Sebagaimana dikenal dalam bahasa pemrograman lainnya (Pascal, BASIC, C, C++ dan lain-lain), variabel adalah nama dari suatu lokasi memori. Dari kacamata pemrogram, akan lebih mudah untuk menyimpan suatu nilai dalam sebuah variabel dengan nama, misalnya, **data01** daripada meyimpan dengan alamat langsung memori, misalnya, 5A88:1239, apalagi jika data yang akan disimpan dalam jumlah yang cukup banyak.

Emu8086 mendukung dua jenis variabel, yaitu **BYTE**, dengan lebar data 8-bit dan **WORD**, dengan lebar data 16-bit. Sintaks atau cara penulisan deklarasi variabel dalam Emu8086 sebagai berikut:

```
nama DB   nilai
nama DW   nilai
```

DB – kepanjangan dari **Define Byte**, dan
DW – kepanjangan dari **Define Word**.

nama – dapat sembarang kombinasi huruf atau angka, diawali dengan huruf. Dimungkinkan juga mendeklarasikan variabel yang tidak bernama (variabel jenis ini memiliki alamat tapi tidak memiliki nama).

nilai – dapat sembarang nilai numerik dalam berbagai macam format yang didukung (desimal, biner dan heksadesimal) atau gunakan simbol **?** untuk variabel-variabel yang belum akan diisi (tidak diinisialisasi).

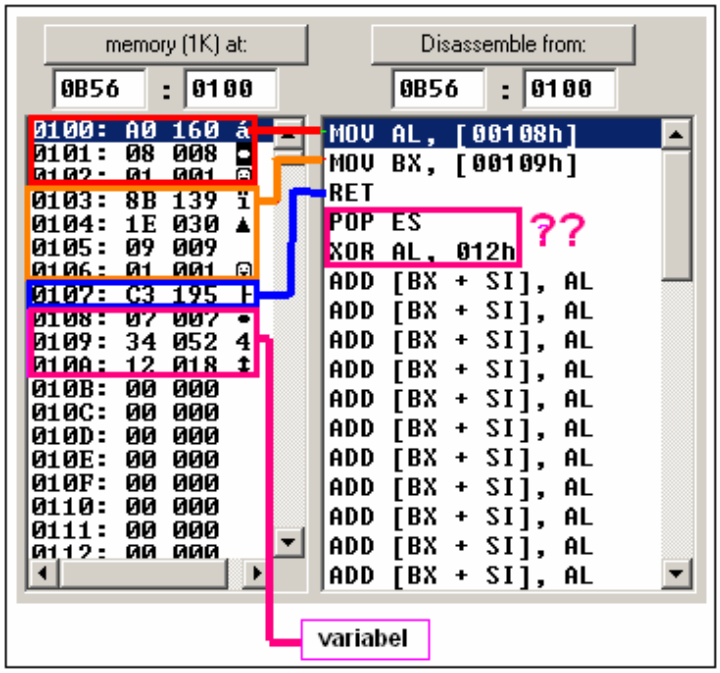
Untuk lebih jelasnya, perhatikan contoh program sebagai berikut:

```
; nama program : program0306.asm
; contoh program dengan deklarasi variabel
;
#make_COM#

; COM file is loaded at CS:0100h
ORG 100h

MOV AL, VAR1
MOV BX, VAR2

RET       ; menghentikan program

VAR1 DB 7
VAR2 DW 1234H
```

Kemudian lakukan emulasi, sebagaimana ditunjukkan pada gambar 3.4, seperti pada contoh-contoh sebelumnya, sekarang semua nama variabel diganti dengan lokasi memori yang sesungguhnya. Saat kompiler membuat kode-kode mesin, secara otomatis semua nama variabel diganti dengan **offset**-nya. Secara *default*, segmen disimpan di register **DS** (akan sama dengan **CS** jika digunakan tipe berkas

COM). Perhatikan bahwa DS terisi dengan **0B56H**, sedangkan **VAR1** (**Emu8086** tidak membedakan huruf kecil maupun besar) diberikan offset **0108H** dan alamat lengkapnya adalah **0B56:0108H**. Sedangkan **VAR2** diberikan offset **0109H** dan alamat lengkapnya **0B56:0108H**, variabel terakhir ini tipenya **word** dengan demikian membutuhkan ruang 2 (dua) byte, lokasi **0109H** dan **010AH**, dengan urutan penyimpanan **34H** kemudian diikuti dengan **12H** (datanya **1234H**).

Perhatikan juga pada tampilan memori pada emulator (gambar 3.4), terdapat dua instruksi setelah **RET**, yaitu **POP ES** dan **XOR AL,012H**, hal ini terjadi karena disasembler tidak tahu dimana data diawali, diterjemahkan begitu saja sebagai suatu instruksi 80x86.



Gambar 3.4. Contoh penggunaan variabel

Anda dapat menuliskan program yang sama hanya dengan pengarah **DB** saja, perhatikan contoh program berikut:

```
; nama program : program0307.asm
; contoh program dengan deklarasi variabel
; penulisan program langsung kode mesin-nya
;
#make_COM#

; COM file is loaded at CS:0100h
ORG 100h

DB 0A0h
DB 08h
DB 01h
```

```
DB 8Bh
DB 1Eh
DB 09h
DB 01h

DB 0C3h

DB 7

DB 34h
DB 12h
```

Setelah dilakukan kompilasi, perhatikan pada emulator, hasilnya sama dengan program yang sebelumnya (begitu juga dengan fungsinya). Sebagaimana dugaan Anda, proses kompilasi hanyalah proses menerjemahkan program sumber menjadi satuan-satuan byte yang dinamakan **kode mesin**, prosesor hanya mengenali kode-kode mesin ini dan menjalankannya.

Jika diperhatikan, kedua contoh program terakhir (**`program0306.asm`** dan **`program0307.asm`**) selalu diawali dengan pengarah **ORG 100H**, sebagaimana ciri-ciri dari program tipe **COM**. Pengarah ini penting sekali saat Anda bekerja dengan variabel. Pengarah ini akan memberitahukan kepada kompiler, bahwa program dimulai pada **offset 100H** (256 byte), dengan demikian kompiler akan melakukan perhitungan alamat dengan benar untuk semua variabel-variabel yang digunakan saat mengganti nama variabel dengan offset-nya masing-masing. Mengapa menempatkan program pada lokasi **100H**? Hal ini dikarenakan sistem operasi menyimpan beberapa data program pada 256 byte yang pertama dari segmen kode, seperti parameter perintah dan lain-lain. Hal ini hanya berlaku untuk jenis program **COM**, program jenis **EXE** bisa mengawali program pada lokasi **0000H**.

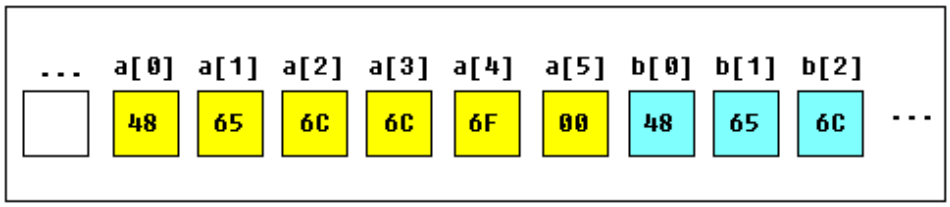
3.3. LARIK DAN PENGGUNAANNYA

Larik atau *array* dapat dilihat sebagai serangkaian variabel-variabel. String teks merupakan contoh larik byte, masing-masing karakter dinyatakan dalam kode ASCII (dalam ukuran byte dari 0 hingga 255).

Berikut ini contoh definisi larik:

```
a    DB   48h, 65h, 6Ch, 6Ch, 6Fh, 00h
b    DB   'Hello', 0
```

b merupakan salinan persis dari **a**, saat kompiler menemui suatu string di antara tanda petik tunggal, maka secara otomatis akan dikonversikan ke kode ASCII. Pada gambar 3.5 ditunjukkan bagian memori yang menyimpan larik yang telah dideklarasikan sebelumnya.



Gambar 3.5. Susunan memori yang menyimpan larik

Anda bisa mengakses sembarang nilai elemen dalam larik menggunakan kurung siku, misalnya:

```
MOV AL, a[3]
```

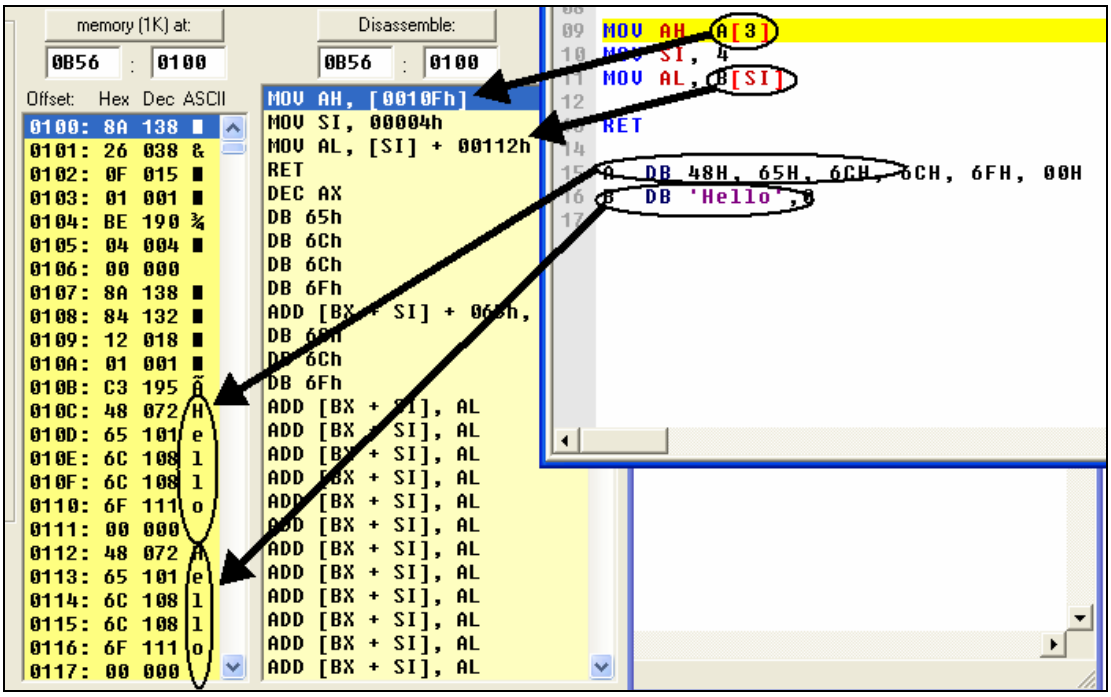
Selain dengan cara menuliskan langsung indeksnya, seperti contoh sebelumnya, Anda bisa juga menggunakan register indeks BX, SI, DI atau BP, misalnya:

```
MOV SI, 3
MOV AL, a[SI]
```

Perhatikan contoh program berikut ini:

```
; nama program : program0308.asm
; contoh program dengan deklarasi larik
;
#make_COM#

; COM file is loaded at CS:0100h
ORG 100h

MOV AH, A[3]
MOV SI, 4
MOV AL, B[SI]

RET

A  DB 48H, 65H, 6CH, 6CH, 6FH, 00H
B  DB 'Hello',0
```

Setelah dilakukan kompilasi, hasilnya ditunjukkan pada gambar 3.6. Larik **A** menempati lokasi offset **010CH** (lengkapnya **0B56:010CH**), indeks ke-3 atau data ke-4 menempati lokasi offset **010FH** (**010C + 3**). Dengan demikian variabel **A[3]** langsung diganti dengan offset **010FH**. Untuk larik **B** menempati lokasi offset **0112H** (lengkapnya **0B56:0112H**), karena akses ke elemen data menggunakan register indeks **SI**, maka hanya awal larik **B** saja yang diganti dengan offset **0112H**, kemudian ditambahkan dengan **SI**.



Gambar 3.6. Contoh penggunaan larik

Jika Anda ingin mendeklarasikan suatu larik yang besar, Anda bisa menggunakan operator **DUP**. Caranya:

```
angka DUP(nilai)
```

angka diisi dengan jumlah duplikasi yang ingn dibuat (sembarang angka), sedangkan **nilai** diisi dengan ekspresi yang akan diduplikasi, misalnya:

```
c DB 5 DUP(9)
```

artinya, menduplikasi angka 9 sebanyak 5 kali, sama dengan jika dituliskan:

```
c DB 9, 9, 9, 9, 9
```

Contoh lain:

```
d DB 5 DUP(1, 2)
```

sama dengan jika dituliskan:

```
d DB 1, 2, 1, 2, 1, 2, 1, 2, 1, 2
```

Tentu saja Anda bisa menggunakan **DW** selain **DB**, tetapi **DW** tidak bisa digunakan untuk mendeklarasikan string!

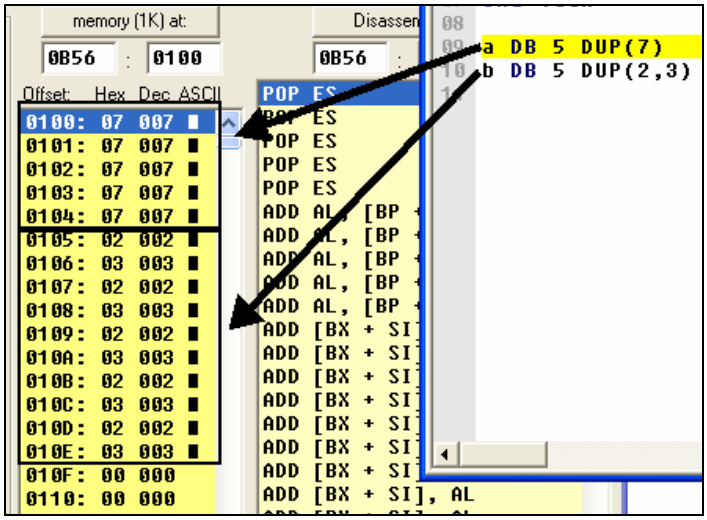
Ekspansi dengan operan **DUP** tidak boleh lebih dari 1020 karakter! Ekspansi yang terakhir hanya 12 karakter, dihitung setelah **DB**. Jika Anda ingin mendeklarasikan

suatu larik yang besar, buatlah deklarasinya menjadi 2 baris, toh pada akhirnya Anda akan mendapatkan sebuah larik besar sesuai keinginan.

Untuk lebih memperjelas, perhatikan contoh program berikut:

```
; nama program : program0309.asm
; contoh program dengan deklarasi dengan DUP
;
#make_COM#

; COM file is loaded at CS:0100h
ORG 100h

a DB 5 DUP(7)
b DB 5 DUP(2,3)
```

Program tersebut sengaja tidak ada satu instruksi-pun yang dituliskan, karena hanya untuk memberikan gambaran tentang penggunaan **DUP**. Setelah dilakukan kompilasi, hasilnya ditunjukkan pada gambar 3.7. Perhatikan bagaimana kompiler melakukan alokasi data sesuai dengan operan DUP dan sesuai dengan pembahasan sebelumnya.



Gambar 3.7. Contoh penggunaan **DUP**

3.4. MEMPEROLEH ALAMAT SUATU VARIABEL

Ada sebuah instruksi, yaitu **LEA** (***Load Effective Address***) dan operan alternatif **OFFSET**, yang dapat digunakan untuk memperoleh alamat offset dari suatu variabel.

Instruksi **LEA** lebih ampuh, karena membolehkan Anda mendapatkan alamat sari suatu variabel terindeks (seperti larik). Memperoleh alamat dari suatu variabel akan sangat bermanfaat dalam beberapa situasi atau kasus, misalnya, saat Anda ingin mengambil suatu parameter ke suatu prosedur.