Pemrograman Dengan Bahasa Assembly Edisi Online Versi 1.0

Penulis : S'to

Editor : Arif Nopi

KATA PENGANTAR

Walaupun bahasa tingkat tinggi terus berkembang dengan segala fasilitas dan kemudahannya, peranan bahasa pemrograman tingkat rendah tetap tidak dapat digantikan. Bahasa assembly mempunyai keunggulan yang tidak mungkin diikuti oleh bahasa tingkat apapun dalam hal kecepatan, ukuran file yang kecil serta kemudahan dalam manipulasi sistem komputer.

Buku ini disusun berdasarkan pengalaman dari penulis sendiri dalam menggunakan bahasa assembler. Oleh karenanya buku ini disusun dengan harapan bagi anda yang tidak tahu sedikitpun tentang assembly dapat belajar sehingga assembler akan tampak sama mudahnya dengan bahasa tingkat tinggi.

Setiap penjelasan pada buku ini akan disertai dengan contoh program yang sesederhana dan semenarik mungkin agar mudah dimengerti. Selain itu juga diberikan TIP-TIP dan TRIK dalam pemrograman !

EDISI ONLINE

Setelah melalui beberapa kali cetakan di PT Gramedia, buku ini tidak mengalami cetakan lanjutan lagi dan mempertimbangkan cukup banyak yang tertarik dengan buku ini, maka saya memutuskan untuk mempublikasikannya secara online dan memberikannya secara gratis sehingga perpanjangan percetakan ke Gramedia tidak akan dilakukan lagi untuk buku ini.

Edisi online ini bisa terlaksana berkat partisipasi dari teman kita melalui milist Jasakom **Arif Nopi** (nophiee@yahoo.com) yang biasa menggunakan nick **Ragila** yang telah meluangkan waktunya yang begitu banyak untuk mengedit versi buku online ini. Melalui ini saya ucapkan banyak terima kasih yang tak terhingga atas kesediaannya menjadi sukarelawan demi kemajuan anak-anak bangsa.

Anda diijinkan untuk mendistribusikan buku ini secara bebas asalkan tidak merubah sedikitpun isi yang ada di buku edisi online. Untuk mendistribuksikannya ataupun saran dan kritik anda bisa menghubungi penulis melalui email sto@poboxes.com atau sto@jasakom.com

S'to

25 Jun 2001
http://www.jasakom.com

DISKET PROGRAM

Buku ini disertai dengan disket program sehingga memudahkan anda yang ingin mencoba program-program didalam buku ini. Semua listing program dalam buku ini disimpan pada directory ASM. Listing program telah diuji semua oleh penulis dan anda dapat memperoleh hasil jadinya dalam bentuk COM maupun EXE pada directory COM.

Selain itu penulis juga menyadari bahwa bahasa assembler adalah bahasa yang rawan. Dengan sedikit kesalahan saja misalkan anda lupa mengakhiri program dengan suatu interupsi atau kesalahan dalam pemakaian interupsi, komputer akan menjadi "hang". Tentunya akan sangat membosankan bila anda harus selalu mem-boot ulang komputer setiap kali menjalankan program anda yang ternyata salah.

Untuk itulah penulis membuat sebuah program yang sangat sederhana dengan ukuran file sekecil mungkin untuk membantu anda. Program tersebut bisa anda dapatkan pada disket program dengan nama SV.COM. Program SV merupakan sebuah program residen (akan anda pelajari pada bab 24) yang akan membelokkan vektor interupsi 05h(PrtScr).

Sebelum anda mulai bereksperimen dengan program-program assembly, jalankanlah dahulu program SV.COM dengan cara:

```
C:\SV  <enter>
```

Setelah program SV dijalankan maka setiap kali program anda mengalami kemacetan anda tinggal menekan tombol PrtScr. Tombol PrtScr akan segera memaksa program tersebut segera kembali ke DOS sehingga anda tidak perlu mem-Boot ulang komputer anda.

Program SV dibuat dengan cara sesederhana dan sekecil mungkin (hanya 816 Byte) sehingga anda tidak perlu khawatir akan kekurangan memory. Listing program SV sengaja tidak disertakan karena diharapkan setelah anda membaca bab 24(tentang residen) anda sudah bisa membuat program semacam ini. Bila anda ingin melihat listing dari program SV ini anda bisa menggunakan program seperti SR.EXE(khusus untuk melihat listing), DEBUG.EXE atau TD.EXE(Bab 27).

UCAPAN TERIMA KASIH

Atas diselesaikannya buku ini, penulis ingin mengucapkan terima kasih kepada Suriyanto, Rudi, Pieter, Harianto, Adi dan Sentosa yang telah meminjamkan buku-buku bacaan, Aripin yang telah meminjamkan printer HP, Wandy, To-je, Aliang, Aminandar, Petrick, Suwangdi dan Weng yang selalu mendukung, serta teman- teman seperjuangan di STMIK BINA NUSANTARA, terutama yang sering menandatangani absen saya.

Diluar itu semua, saya juga sangat berterima kasih kepada Sdr. AriSubagijo yang telah banyak membantu, Staf serta pimpinan dari ELEX MEDIA KOMPUTINDO.

Jakarta, 27 November 1994 4

S'to

BAB I
BILANGAN

1.1. BERBAGAI JENIS BILANGAN

Didalam pemrograman dengan bahasa assembler, bisa digunakan berbagai jenis bilangan. Jenis bilangan yang bisa digunakan, yaitu: Bilangan biner, oktaf, desimal dan hexadesimal. Pemahaman terhadap jenis-jenis bilangan ini adalah penting, karena akan sangat membantu kita dalam pemrograman yang sesungguhnya.

1.1.1. BILANGAN BINER

Sebenarnya semua bilangan, data maupun program itu sendiri akan diterjemahkan oleh komputer ke dalam bentuk biner. Jadi pendefinisisan data dengan jenis bilangan apapun(Desimal, oktaf dan hexadesimal) akan selalu diterjemahkan oleh komputer ke dalam bentuk biner.

Bilangan biner adalah bilangan yang hanya terdiri atas 2 kemungkinan(Berbasis dua), yaitu 0 dan 1. Karena berbasis 2, maka pengkorversian ke dalam bentuk desimal adalah dengan mengalikan suku ke-N dengan 2^N. Contohnya: bilangan biner $0111_2 = (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) = 7_{10}$.

1.1.2. BILANGAN DESIMAL

Tentunya jenis bilangan ini sudah tidak asing lagi bagi kita semua. Bilangan Desimal adalah jenis bilangan yang paling banyak dipakai dalam kehidupan sehari-hari, sehingga kebanyakan orang sudah akrab dengannya.

Bilangan desimal adalah bilangan yang terdiri atas 10 buah angka(Berbasis 10), yaitu angka 0-9. Dengan basis sepuluh ini maka suatu angka dapat dijabarkan dengan perpangkatan sepuluh. Misalkan pada angka $123_{10} = (1 \times 10^2) + (2 \times 10^1) + (1 \times 10^0)$.

1.1.3. BILANGAN OKTAL

Bilangan oktal adalah bilangan dengan basis 8, artinya angka yang dipakai hanyalah antara 0-7. Sama halnya dengan jenis bilangan yang lain, suatu bilangan oktal dapat dikonversikan dalam bentuk desimal dengan mengalikan suku ke-N dengan 8^N. Contohnya bilangan $12_8 = (1 \times 8^1) + (2 \times 8^0) = 10_{10}$.

1.1.4. BILANGAN HEXADESIMAL

Bilangan hexadesimal merupakan bilangan yang berbasis 16. Dengan angka yang digunakan berupa:

0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

Dalam pemrograman assembler, jenis bilangan ini boleh dikatakan yang paling banyak digunakan. Hal ini dikarenakan mudahnya pengkonversian bilangan ini dengan bilangan yang lain, terutama dengan bilangan biner dan desimal. Karena berbasis 16, maka 1 angka pada hexadesimal akan menggunakan 4 bit.

1.2. BILANGAN BERTANDA DAN TIDAK

Pada assembler bilangan-bilangan dibedakan lagi menjadi 2, yaitu bilangan bertanda dan tidak. Bilangan bertanda adalah bilangan yang mempunyai arti plus(+) dan minus(-), misalkan angka 17 dan -17. Pada bilangan tidak bertanda, angka negatif(yang mengandung tanda '-') tidaklah dikenal. Jadi angka -17 tidak akan akan dikenali sebagai angka -17, tetapi sebagai angka lain.

Kapan suatu bilangan perlakukan sebagai bilangan bertanda dan tidak? Assembler akan selalu melihat pada Sign Flag, bila pada flag ini bernilai 0, maka bilangan akan diperlakukan sebagai bilangan tidak bertanda, sebaliknya jika flag ini bernilai 1, maka bilangan akan diperlakukan sebagai bilangan bertanda.

Pada bilangan bertanda bit terakhir (bit ke 16) digunakan sebagai tanda plus(+) atau minus(-). Jika pada bit terakhir bernilai 1 artinya bilangan tersebut adalah bilangan negatif, sebaliknya jika bit terakhir bernilai 0, artinya bilangan tersebut adalah bilangan positif(Gambar 1.1).

>>>> Bilangan <<<<		
Biner	Tidak Bertanda	Bertanda
0000 0101	+ 5	+ 5
0000 0100	+ 4	+ 4
0000 0011	+ 3	+ 3
0000 0010	+ 2	+ 2
0000 0001	+ 1	+ 1
0000 0000	0	0

1111 1111	+ 255	- 1
1111 1110	+ 254	- 2
1111 1101	+ 253	- 3
1111 1100	+ 252	- 4
1111 1011	+ 251	- 5
1111 1010	+ 250	- 6

Gambar 1.1. Bilangan Bertanda dan Tidak

BAB II

M E M O R I

Memori dengan komputer memiliki hubungan yang tak dapat dipisahkan, karena setiap komputer memerlukan memori sebagai tempat kerjanya. Memori ini dapat berfungsi untuk memuat program dan juga sebagai tempat untuk menampung hasil proses.

Yang perlu kita perhatikan bahwa memori untuk menyimpan program maupun hasil dari pekerjaan bersifat volatile yang berarti bahwa data yang disimpan cuma sebatas adanya aliran listrik. Jadi bila listrik mati maka hilang pulalah semua data yang ada di dalamnya. Hal ini mengakibatkan diperlukannya media penyimpan kedua yang biasanya berupa disket maupun hard disk.

2.1. Microprocessor

Pada IBM-PC terdapat suatu bagian penting yang disebut microprocessor atau yang sering disebut processor saja. Processor ini berfungsi untuk menangani keseluruhan dari kerja komputer kita. Pada processor inilah segala hal yang berhubungan dengan kerja komputer diatur dan dibagi prioritasnya dengan baik agar tidak terjadi kesalahan yang kemudian akan menyebabkan kacaunya informasi yang diperoleh.

Lama kelamaan tugas komputer tentu saja makin bertambah baik dari segi kuantitas maupun kerumitannya. Sejalan dengan itu processor juga makin dikembangkan. Processor yang baru sebenarnya hanyalah perbaikan dan pengembangan dari yang versi lama sehingga semua instruksi yang berlaku di processor lama dapat pula dikerjakan oleh yang baru dengan tentu saja beberapa keunggulan.

Adapun processor yang kini banyak beredar di pasaran :

- 8088 & 8086 :

Ini merupakan processor IBM-PC yang pertama sekali atau yang sering disebut XT. Processor 8088 menggunakan jalur bus data 8 bit sedangkan 8086 menggunakan 16 bit. Perbedaan jalur bus ini menyebabkan perbedaan jumlah data yang dikirim pada satu saat dan secara langsung mengakibatkan speed 8086 berada di atas 8088. Baik 8088 maupun 8086 mampu mengalamatkan memori hingga 1 MB.

- 80286 :

Versi pengembangan dari 8086. Pada 80286 ini beberapa instruksi baru ditambahkan. Selain itu dengan jalur bus yang sama dengan 8086, 80286 dirancang mempunyai speed di atas 8086. Selain itu 80286 dapat bekerja pada 2

mode yaitu mode real dan protected.

Mode real pada 80286 dapat beroperasi sama seperti 8088 dan 8086 hanya terdapat perbedaan dalam hal speed. Mode real ini dimaksudkan agar semua software yang dapat dioperasikan pada 8088/8086 dapat pula dioperasikan dengan baik di 80286. Pada mode protected 80286 mampu mengalamatkan sampai 16 MB memori.

- 80386 :

Processor 80386 merupakan sesuatu yang sangat baru dibanding 80286 sebab bus data yang digunakan di sini sudah 32 bit sehingga speednya juga jauh di atas 80286. Selain itu pada 80386 ditambahkan pula sebuah mode pemrograman baru yaitu mode virtual. Pada mode virtual ini 80386 mampu mengalamatkan sampai 4 GB memori. Sama seperti 80286, mode real dimaksudkan untuk kompatibilitas dengan 8088/8086 dan mode protected untuk menjaga kompatibilitas dengan 80286.

2.2. Organisasi Memori Pada PC

Memori yang ada pada komputer perlu diatur sedemikian rupa sehingga mudah dalam pengaksesannya. Oleh sebab itu dikembangkanlah suatu metode yang efektif dalam pengorganisasiannya. Pada bagian ini akan dibahas mengenai pengorganisasian memori ini.

2.3. Pembagian Memori

Memori komputer terbagi atas 16 blok dengan fungsi-fungsi khusus yang sebagian besar adalah sebagai RAM (Random Access Memory) yang berfungsi sebagai penyimpan bagi hasil pengolahan pada komputer itu sendiri. Untuk lebih jelasnya diberikan pembagian fungsi pada blok memori ini secara kasar pada gambar 2.1.

block	fungsi
0	RAM
1	RAM
2	RAM
3	RAM
4	RAM
5	RAM
6	RAM
7	RAM

8	RAM
9	RAM
A	EXTENDED VIDEO MEMORI
B	EXTENDED VIDEO MEMORY
C	PERLUASAN ROM
D	FUNGSI LAIN
E	FUNGSI LAIN
F	BIOS & BASIC

Gambar 2.1. Pembagian blok memori IBM PC

2.4. Pengalamatan Memori Dengan Segment Offset

Sudah kita bahas bersama bahwa baik 8086 maupun mode real 80286 dapat mengalamatkan sampai 1 MB memori. Tetapi sebenarnya baik 8086 maupun 80286 adalah procesor 16 bit. Banyaknya memori yang dapat dicatat atau dialamatkan oleh procesor 16 bit adalah maksimal 2^{16} byte (=64 KB). Jadi bagaimana 8086 dan mode real 80286 mampu mengalamatkan sampai 1 MB memori ?.

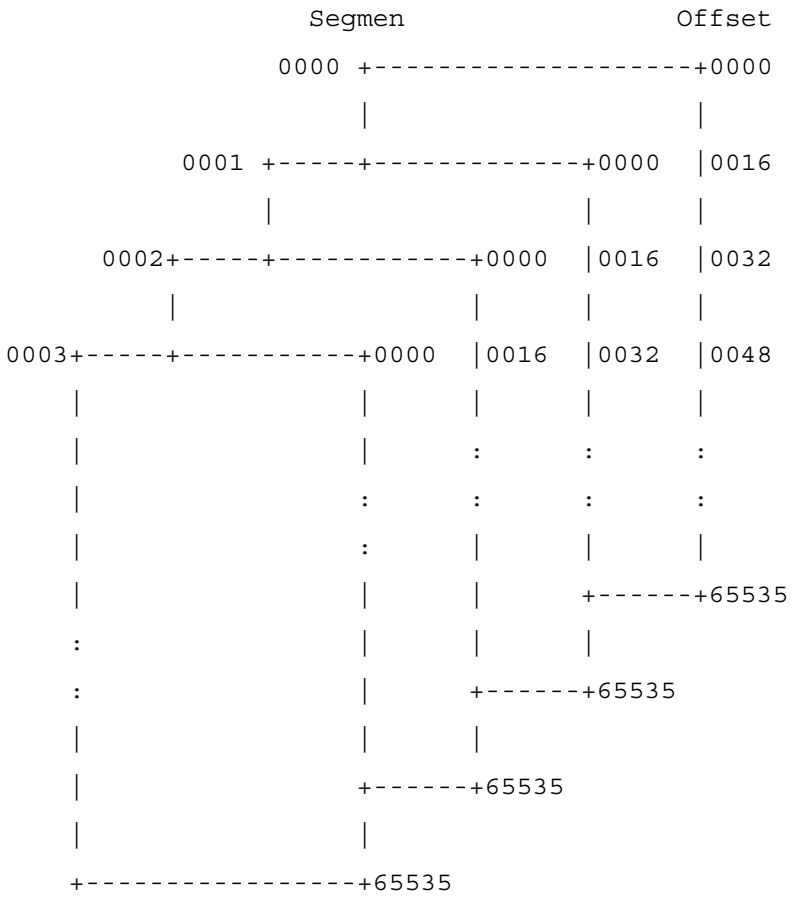
Hal ini dapat dimungkinkan dengan adanya pengalamatan yang menggunakan sistem 20 bit walaupun sebenarnya procesor itu hanya 16 bit. Dengan cara ini dapat dialamatkan 2^{20} byte (=1 MB) memori.

Tetapi masih tetap ada satu kendala dalam pengalamatan 20 bit ini. Yaitu bahwa sesuai dengan tipenya procesor ini hanya mampu mengakses 16 bit data pada satu kali akses time. Sebagai
pemecahannya dikembangkanlah suatu metode pengalamatan 20 bit
yang dimasukkan ke dalam format 16 bit.

Pada metode pengalamatan ini baik 8086 maupun mode real
80286 membagi ruang memori ke dalam segmen-segmen di mana besar 1 segmen adalah 64 KB (=2^{16} byte). Jadi pada segmen 0000h(Tanda "h" menunjukkan hexadesimal) terdapat 64 KB data, demikian pula dengan segmen 0001h dan seterusnya.

Sekarang bagaimana caranya agar setiap data yang tersimpan dalam satu segmen yang besarnya 64 KB itu dapat diakses secara individual. Cara yang dikembangkan adalah dengan membagi-bagi setiap segmen menjadi bagian-bagian yang disebut offset. Dalam satu segmen terdapat 2^{16} offset yang diberi nomor dari 0000h sampai FFFFh. Nomor offset selalu diukur relatif dari awal suatu segmen.

Sekarang kita lihat bagaimana sebenarnya letak suatu segmen dalam memori komputer kita. Segmen 0000h berawal dari lokasi memori 0 hingga 65535 (64 KB). Segmen 0001h berawal dari lokasi memori 16 (0010h) hingga 65551 (65535 + 16). Segmen 0002h berawal dari lokasi 32 hingga 65567. Demikian seterusnya. Kita lihat bahwa sistem penempatan segmen semacam ini akan menyebabkan ter-



Gambar 2.2. Peta Overlapping Segmen

jadinya overlapping (tumpang-tindih) di mana lokasi offset 0010h bagi segmen 0000h akan merupakan offset 0000h bagi segmen 0001h. Demikian pula offset 0011h bagi segmen 0000h akan merupakan offset 0001h bagi segmen 0001h. Dalam pembahasan selanjutnya akan kita lihat bahwa ada banyak nilai segmen:offset yang dapat digunakan untuk menyatakan suatu alamat memori tertentu disebabkan adanya overlapping ini. Untuk lebih jelasnya dapat kita lihat pada gambar 2.2.

2.5. Konversi Alamat

Alamat yang menggunakan sistem segmen:offset ini disebut sebagai alamat relatif karena sifat offset yang relatif terhadap segmen. Sedangkan alamat memori yang sebenarnya disebut alamat absolut. Berikut kita lihat cara pengkonversian alamat relatif ke absolut.

Pengkonversian dapat dilakukan dengan menggeser nilai segmen 4 bit ke kiri dan kemudian dijumlahkan dengan nilai offset. Atau yang lebih sederhana adalah dengan mengalikan nilai segmen dengan 2^4 (=10h) dan kemudian dijumlahkan dengan nilai offset. Cara ini dikembangkan dari besarnya selisih segmen yang satu dengan yang berikutnya yang sebesar 16 bit (=10h).

```
Alamat relatif :   1357h:2468h          1356h:2478h

                     13570                13560
                      2468                 2478
                    -------              -------
Alamat absolut :    159D8h               159D8h
```

Pada kedua contoh di atas terlihat jelas alamat relatif 1357h:2468h sebenarnya menunjukkan lokasi yang sama dalam memori dengan alamat relatif 1356h:2478h yang disebut overlapping.

Alamat yang overlapping ini menyebabkan sebuah alamat absolute dapat dinyatakan dengan alamat segmen:offset yang bervariasi sebanyak 2 pangkat 12 atau sebanyak 4096 variasi.

Variasi untuk alamat absolute :

```
0  - 15      dapat dinyatakan dengan 1 variasi
16 - 31      dapat dinyatakan dengan 2 variasi
32 - 48      dapat dinyatakan dengan 3 variasi
   :
   :
65520 keatas  dapat dinyatakan dengan 4096 variasi.
```

BAB III
INTERRUPT

3.1. PENGERTIAN INTERRUPT

Interupsi adalah suatu permintaan khusus kepada mikroposesor untuk melakukan sesuatu. Bila terjadi interupsi, maka komputer akan menghentikan dahulu apa yang sedang dikerjakannya dan melakukan apa yang diminta oleh yang menginterupsi.

Pada IBM PC dan kompatibelnya disediakan 256 buah interupsi yang diberi nomor 0 sampai 255. Nomor interupsi 0 sampai 1Fh disediakan oleh ROM BIOS, yaitu suatu IC didalam komputer yang mengatur operasi dasar komputer. Jadi bila terjadi interupsi dengan nomor 0-1Fh, maka secara default komputer akan beralih menuju ROM BIOS dan melaksanakan program yang terdapat disana. Program yang melayani suatu interupsi dinamakan Interrupt Handler.

3.2. VEKTOR INTERUPSI

Setiap interrupt akan mengeksekusi interrupt handlernya masing-masing berdasarkan nomornya. Sedangkan alamat dari masing- masing interupt handler tercatat di memori dalam bentuk array yang besar elemennya masing-masing 4 byte. Keempat byte ini dibagi lagi yaitu 2 byte pertama berisi kode offset sedangkan 2 byte berikutnya berisi kode segmen dari alamat interupt handler yang bersangkutan. Jadi besarnya array itu adalah 256 elemen dengan ukuran elemen masing-masing 4 byte. Total keseluruhan memori yang dipakai adalah sebesar 1024 byte (256 x 4 = 1024) atau 1 KB dan disimpan dalam lokasi memori absolut 0000h sampai 3FFh. Array sebesar 1 KB ini disebut Interupt Vector Table (Table Vektor Interupsi). Nilai-nilai yang terkandung pada Interupt Vector Table ini tidak akan sama di satu komputer dengan yang lainnya.

Interupt yang berjumlah 256 buah ini dibagi lagi ke dalam 2 macam yaitu:

- Interupt 00h - 1Fh (0 - 31) adalah interrupt BIOS dan standar di semua komputer baik yang menggunakan sistem operasi DOS atau bukan. Lokasi Interupt Vector Table-nya ada di alamat absolut 0000h-007Fh.
- Interupt 20h - FFh (32 - 255) adalah interrupt DOS. Interrupt ini hanya ada pada komputer yang menggunakan sistem operasi DOS dan Interupt Handler-nya di-load ke memori oleh DOS pada saat DOS digunakan. Lokasi Interupt Vector Table-nya ada di alamat absolut 07Fh-3FFh.

Nomor Interupt	Nama Interupt	Nomor Interupt	Nama Interupt
*00h	Divide By Zero	10h	Video Service
*01h	Single Step	11h	Equipment Check
*02h	Non MaskableInt(NMI)	12h	Memory Size
*03h	Break point	13h	Disk Service
04h	Arithmatic Overflow	14h	Communication (RS-232)
05h	Print Screen	15h	Cassette Service
06h	Reserved	16h	Keyboard Service
07h	Reserved	17h	Printer Service
08h	Clock Tick(Timer)	18h	ROM Basic
09h	Keyboard	19h	Bootstrap Loader
0Ah	I/O Channel Action	1Ah	BIOS time & date
0Bh	COM 1 (serial 1)	1Bh	Control Break
0Ch	COM 2 (serial 2)	1Ch	Timer Tick
0Dh	Fixed Disk	1Dh	Video Initialization
0Eh	Diskette	1Eh	Disk Parameters
0Fh	LPT 1 (Parallel 1)	1Fh	Graphics Char

Gambar 3.1. BIOS Interrupt

* Interrupt ini telah dipastikan kegunaannya oleh sistem untuk keperluan yang khusus , tidak boleh dirubah oleh pemrogram seperti yang lainnya.

- DEVIDE BY ZERO : Jika terjadi pembagian dengan nol maka proses akan segera dihentikan.
- SINGLE STEP : Untuk melaksanakan / mengeksekusi intruksi satu persatu.
- NMI : Pelayanan terhadap NMI (Non Maskable Interrupt) yaitu interupsi yang tak dapat dicegah.
- BREAK POINT : Jika suatu program menyebabkan overflow flag menjadi 1 maka interrupt ini akan melayani pencegahannya dan memberi tanda error.

Nomor Interrupt	Nama Interrupt
20h	Terminate Program
21h	DOS Function Services
22h	Terminate Code
23h	Ctrl-Break Code
24h	Critical Error Handler
25h	Absolute Disk Read
26h	Absolute Disk Write
27h	Terminate But Stay Resident

Gambar 3.2. DOS Interrupt

Didalam pemrograman dengan bahasa assembler kita akan banyak sekali menggunakan interupsi untuk menyelesaikan suatu tugas.

BAB IV
REGISTER

4.1.PENGERTIAN REGISTER

Dalam pemrograman dengan bahasa Assembly, mau tidak mau anda harus berhubungan dengan apa yang dinamakan sebagai Register. Lalu apakah yang dimaksudkan dengan register itu sebenarnya ?.

Register merupakan sebagian memori dari mikroprosesor yang dapat diakses dengan kecepatan yang sangat tinggi. Dalam melakukan pekerjaannya mikroprosesor selalu menggunakan register-register sebagai perantaranya, jadi register dapat diibaratkan sebagai kaki dan tangannya mikroprosesor.

4.2.JENIS-JENIS REGISTER

Register yang digunakan oleh mikroprosesor dibagi menjadi 5 bagian dengan tugasnya yang berbeda-beda pula, yaitu :

4.2.1. Segmen Register.

Register yang termasuk dalam kelompok ini terdiri atas register CS,DS,ES dan SS yang masing-masingnya merupakan register 16 bit. Register-register dalam kelompok ini secara umum digunakan untuk menunjukkan alamat dari suatu segmen.

Register **CS**(Code Segment) digunakan untuk menunjukkan tempat dari segmen yang sedang aktif, sedangkan register **SS**(Stack Segment) menunjukkan letak dari segmen yang digunakan oleh stack. Kedua register ini sebaiknya tidak sembarang diubah karena akan menyebabkan kekacauan pada program anda nantinya.

Register **DS**(Data Segment) biasanya digunakan untuk menunjukkan tempat segmen dimana data-data pada program disimpan. Umumnya isi dari register ini tidak perlu diubah kecuali pada program residen. Register **ES**(Extra Segment), sesuai dengan namanya adalah suatu register bonus yang tidak mempunyai suatu tugas khusus. Register ES ini biasanya digunakan untuk menunjukkan suatu alamat di memory, misalkan alamat memory video.

Pada prosesor 80386 terdapat tambahan register segment 16 bit, yaitu FS<Extra Segment> dan GS<Extra Segment>.

4.2.2. Pointer dan Index Register.

Register yang termasuk dalam kelompok ini adalah register SP,BP,SI dan DI yang masing-masing terdiri atas 16 bit. Register- register dalam kelompok

ini secara umum digunakan sebagai penunjuk atau pointer terhadap suatu lokasi di memory.

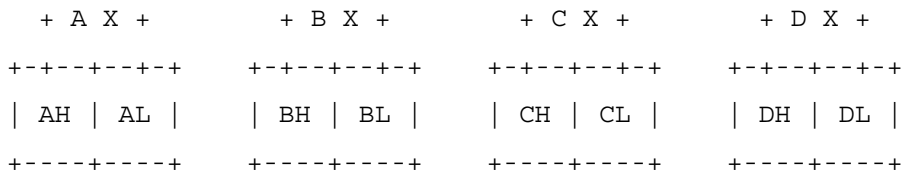
Register **SP**(Stack Pointer) yang berpasangan dengan register segment SS(SS:SP) digunakan untuk mununjukkan alamat dari stack, sedangkan register **BP**(Base Pointer)yang berpasangan dengan register SS(SS:BP) mencatat suatu alamat di memory tempat data.

Register **SI**(Source Index) dan register **DI**(Destination Index) biasanya digunakan pada operasi string dengan mengakses secara langsung pada alamat di memory yang ditunjukkan oleh kedua register ini.

Pada prosesor 80386 terdapat tambahan register 32 bit, yaitu ESP,EBP,ESI dan EDI.

4.2.3. General Purpose Register.

Register yang termasuk dalam kelompok ini adalah register AX,BX,CX dan DX yang masing-masing terdiri atas 16 bit. Register- register 16 bit dari kelompok ini mempunyai suatu ciri khas, yaitu dapat dipisah menjadi 2 bagian dimana masing-masing bagian terdiri atas 8 bit, seperti pada gambar 4.1. Akhiran **H** menunjukkan High sedangkan akhiran **L** menunjukkan Low.



Gambar 4.1. General purpose Register

Secara umum register-register dalam kelompok ini dapat digunakan untuk berbagai keperluan, walaupun demikian ada pula penggunaan khusus dari masing-masing register ini yaitu :

Register **AX**, secara khusus digunakan pada operasi aritmatika terutama dalam operasi pembagian dan pengurangan.

Register **BX**, biasanya digunakan untuk menunjukkan suatu alamat offset dari suatu segmen.

Register **CX**, digunakan secara khusus pada operasi looping dimana register ini menentukan berapa banyaknya looping yang akan terjadi.

Register **DX**, digunakan untuk menampung sisa hasil pembagian 16 bit.

Pada prosesor 80386 terdapat tambahan register 32 bit, yaitu EAX,EBX,ECX dan EDX.

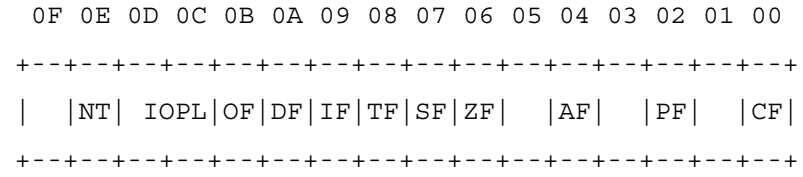
4.2.4. Index Pointer Register

Register IP berpasangan dengan CS(CS:IP) menunjukkan alamat dimemory tempat dari intruksi(perintah) selanjutnya yang akan dieksekusi. Register IP juga merupakan register 16 bit. Pada prosesor 80386 digunakan register EIP yang merupakan register 32 bit.

4.2.5. Flags Register.

Sesuai dengan namanya Flags(Bendera) register ini menunjukkan kondisi dari suatu keadaan< ya atau tidak >. Karena setiap keadaan dapat digunakan 1 bit saja, maka sesuai dengan jumlah bitnya, Flags register ini mampu memcatat sampai 16 keadaan. Adapun flag yang terdapat pada mikroprosesor 8088 keatas adalah :

- OF <OverFlow Flag>. Jika terjadi OverFlow pada operasi aritmatika, bit ini akan bernilai 1.
- SF <Sign Flag>. Jika digunakan bilangan bertanda bit ini akan bernilai 1
- ZF <Zero Flag>. Jika hasil operasi menghasilkan nol, bit ini akan bernilai 1.
- CF <Carry Flag>. Jika terjadi borrow pada operasi pengurangan atau carry pada penjumlahan, bit ini akan bernilai 1.



Gambar 4.2. Susunan Flags Register 8088

- PF <Parity Flag>. Digunakan untuk menunjukkan paritas bilangan. Bit ini akan bernilai 1 bila bilangan yang dihasilkan merupakan bilangan genap.
- DF <Direction Flag>. Digunakan pada operasi string untuk menunjukkan arah proses.
- IF <Interrupt Enable Flag>. CPU akan mengabaikan interupsi yang terjadi jika bit ini 0.
- TF <Trap Flag>. Digunakan terutama untuk Debugging, dengan operasi step by step.

- AF <Auxiliary Flag>. Digunakan oleh operasi BCD, seperti pada perintah AAA.
- NT <Nested Task>. Digunakan pada prosesor 80286 dan 80386 untuk menjaga jalannya interupsi yang terjadi secara beruntun.
- IOPL <I/O Protection level>. Flag ini terdiri atas 2 bit dan digunakan pada prosesor 80286 dan 80386 untuk mode proteksi.

Adapun susunan dari masing-masing flag didalam flags register dapat anda lihat pada gambar 4.2. Pada prosesor 80286 dan 80386 keatas terdapat beberapa tambahan pada flags register, yaitu :

- PE <Protection Enable>. Digunakan untuk mengaktifkan mode proteksi. flag ini akan bernilai 1 pada mode proteksi dan 0 pada mode real.
- MP <Monitor Coprosesor>. Digunakan bersama flag TS untuk menangani terjadinya intruksi WAIT.
- EM <Emulate Coprosesor>. Flag ini digunakan untuk mensimulasikan coprosesor 80287 atau 80387.
- TS <Task Switched>. Flag ini tersedia pada 80286 keatas.
- ET <Extension Type>. Flag ini digunakan untuk menentukan jenis coprosesor 80287 atau 80387.
- RF <Resume Flag>. Register ini hanya terdapat pada prosesor 80386 keatas.
- VF <Virtual 8086 Mode>. Bila flag ini bernilai 1 pada saat mode proteksi, mikroprosesor akan memungkinkan dijalankannya aplikasi mode real pada mode proteksi. Register ini hanya terdapat pada 80386 keatas.

BAB V

MEMULAI DENGAN ASSEMBLY

5.1. TEXT EDITOR

Untuk menuliskan source file untuk program assembly bisa anda gunakan berbagai editor, misalkan SideKick, WordStar dan Word Perfect. Source file yang diketikkan harus berupa file ASCII, file ini bisa anda hasilkan melalui WordStar dengan file 'NON DOCUMEN', atau dengan SideKick.

Untuk meyakinkan bahwa source file yang anda buat adalah file ASCII, bisa anda coba ketikkan perintah Type pada A>. Bila file yang terlihat dengan perintah type sama persis dengan yang anda ketikkan pada editor, tanpa tambahan karakter-karakter yang acak, maka file tersebut adalah file ASCII. Source file untuk assembly harus berektensi .ASM.

5.2. COMPILER

Source file ASCII yang telah anda ketikkan perlu dicompile kebentuk file object dengan extensi .OBJ, dari file object inilah nantinya dapat dijadikan kebentuk file .EXE atau .COM.

Untuk mengcompile source file, misalnya file COBA.ASM menjadi file object dengan extensi .OBJ bisa anda gunakan file TASM.EXE dengan mengetikkan:

```
C:\>tasm coba
Turbo Assembler  Version 2.0  Copyright (c) 1988,
1990 Borland International

Assembling file:   coba.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  307k
C:\>dir coba.*

 Volume in drive C is S'to
 Directory of C:\

COBA     OBJ           128 08-12-94  10:42p
COBA     ASM           128 08-12-94  10:41p
        2 file(s)             246 bytes
                      1,085,952 bytes free
```

5.3. LINGKING

File object yang telah terbentuk dengan TASM, belum dapat dieksekusi secara langsung. Untuk membuat file object ke bentuk file yang dapat dieksekusi(ektensi .COM atau .EXE) bisa anda gunakan file TLINK.EXE.

Bila source program yang anda buat dalam bentuk EXE maka untuk membentuk

file dengan ektensi EXE bisa anda ketikkan :

```
C:\>tlink coba
Turbo Link  Version 3.0 Copyright (c) 1987,
1990 Borland International
```

Bila source program yang dibuat adalah file COM, maka bisa anda ketikkan:

```
C:\>tlink/t coba
Turbo Link  Version 3.0 Copyright (c) 1987,
1990 Borland International
```

5.4. PERBEDAAN PROGRAM COM DAN EXE

Program dengan ektensi COM dan EXE mempunya berbagai perbedaan yang menyolok, antara lain :

PROGRAM COM :
- Lebih pendek dari file EXE
- Lebih cepat dibanding file EXE
- Hanya dapat menggunakan 1 segmen
- Ukuran file maksimum 64 KB (ukuran satu segment)
- sulit untuk mengakses data atau procedure yang terletak pada segment yang lain.
- 100h byte pertama merupakan PSP(Program Segment Prefix) dari program tersebut.
- Bisa dibuat dengan DEBUG

PROGRAM EXE :
- Lebih panjang dari file COM
- Lebih lambat dibanding file COM
- Bisa menggunakan lebih dari 1 segmen
- Ukuran file tak terbatas sesuai dengan ukuran memory.
- mudah mengakses data atau procedure pada segment yang lain.
- Tidak bisa dibuat dengan DEBUG

5.5. BENTUK ANGKA

Assembler mengizinkan penggunaan beberapa bentuk angka , yaitu :

1. DESIMAL

Untuk menuliskan angka dalam bentuk desimal, bisa digunakan tanda 'D' pada akhir angka tersebut atau bisa juga tidak diberi tanda sama sekali, contoh : 298**D** atau 298 saja.

2. BINER

Untuk menuliskan angka dalam bentuk biner(0..1), harus ditambahkan tanda 'B' pada akhir angka tersebut, contoh : 01100111**B**.

3. HEXADESIMAL

Untuk menuliskan angka dalam bentuk hexadesimal(0..9,A..F), harus ditambahkan tanda 'H' pada akhir angka tersebut. Perlu diperhatikan bahwa bila angka pertama dari hexa berupa karakter(A..F) maka angka nol harus ditambahkan didepannya. Bila hal ini tidak dilakukan, assembler akan menganggapnya sebagai suatu label, bukannya sebagai nilai hexa. Contoh penulisan yang benar: **0**A12**H**, 2A02**H**.

4. KARAKTER

Penulisan karakter atau string diapit oleh tanda petik dua (**"**) atau tanda petik satu(**'**), Contoh: **'** Ini adalah karakter **'**.

5.6. LABEL

Label bisa anda definisikan dengan ketentuan akhir dari nama label tersebut harus berupa tanda titik dua (**:**). Pemberian nama label bisa digunakan:

- Huruf : A..Z (Huruf besar dan kecil tidak dibedakan)
- Angka : 0..9
- Karakter khusus : @ . _ $

Nama pada label tidak boleh terdapat spasi dan didahului oleh angka, Contoh dari penulisan label yang benar: **mulai:** MOV CX,7. Nama label terpanjang yang dapat dikenali oleh assembler adalah 31 karakter.

5.7. KOMENTAR

Untuk memberikan komentar pada source file digunakan tanda '**;**'. Apapun yang dtuliskan dibelakang tanda '**;**' akan dianggap sebagai komentar, Contoh : mulai: MOV BX,7 **; berikan nilai 7 pada BX**

5.8. PERINTAH MOV

Perintah MOV digunakan untuk mengcopy nilai atau angka menuju suatu register,variabel atau memory. Adapun syntax untuk perintah MOV ini adalah :

MOV Tujuan,Asal

Sebagai contohnya :
```
MOV AL,9  ; masukkan nilai 9 pada AL.
MOV AH,AL ; nilai AL=9 dan AH=9
MOV AX,9  ; AX=AH+AL hingga AH=0 dan AL:=9
```

Pada baris pertama(**MOV AL,9**), kita memberikan nilai 9 pada register AL. Kemudian pada baris kedua(**MOV AH,AL**) kita mengcopykan nilai register AL untuk AH. Jadi setelah operasi ini register AL akan tetap bernilai 9, dan register

AH akan sama nilainya dengan AL atau 9. Pada baris ketiga(**MOV AX,9**), kita memberikan register AX nilai 9. Karena AX terdiri atas AH dan AL, maka register AH akan bernilai 0, sedangkan AL akan bernilai 9.

Perintah MOV akan mengcopykan nilai pada sumber untuk dimasukan ke Tujuan, nilai sumber tidaklah berubah. Inilah sebabnya MOV(E) akan kita terjemahkan disini dengan mengcopy, dan bukannya memindahkan.

5.9. PERINTAH INT

Didalam pemrograman assambler, kita akan banyak sekali menggunakan interupsi untuk membantu kita dalam mengerjakan suatu pekerjaan. Untuk menghasilkan suatu interupsi digunakan perintah INT dengan syntax:

INT NoInt

Dengan NoInt adalah nomor interupsi yang ingin dihasilkan. Sebagai contohnya bila kita ingin menghasilkan interupsi 21h, bisa dituliskan dengan: **INT 21h**, maka interupsi 21h akan segera terjadi.

BAB VI
MEMBUAT PROGRAM COM

6.1. MODEL PROGRAM COM

Untuk membuat program .COM yang hanya menggunakan 1 segment, bisa anda buat dengan model program seperti gambar 6.1. Bentuk yang digunakan disini adalah bentuk program yang dianjurkan(Ideal). Dipilihnya bentuk program ideal dalam buku ini dikarenakan pertimbangan dari berbagai keunggulan bentuk program ideal ini seperti, prosesnya lebih cepat dan lebih mudah digunakan oleh berbagai bahasa tingkat tinggi yang terkenal(Turbo Pascal dan C).

```
-----------------------------------------------------------
          .MODEL SMALL
          .CODE
           ORG 100H

   Label1 : JMP Label2
            +---------------------+
            | TEMPAT DATA PROGRAM |
            +---------------------+

   Label2 : +---------------------+
            | TEMPAT   PROGRAM    |
            +---------------------+

            INT 20H
    END     Label1
-----------------------------------------------------------
```

Gambar 6.1. Model Program COM

Supaya lebih jelas bentuk dari program ideal, marilah kita telusuri lebih lanjut dari bentuk program ini.

6.1.1 .MODEL SMALL

Tanda directive ini digunakan untuk memberitahukan kepada assembler bentuk memory yang digunakan oleh program kita. Supaya lebih jelas model-model yang bisa digunakan adalah :

- TINY

Jika program anda hanya menggunakan 1 segment seperti program COM. Model ini disediakan khusus untuk program COM.

- SMALL

Jika data dan code yang digunakan oleh program kurang dari ukuran 1 segment atau 64 KB.

- MEDIUM

Jika data yang digunakan oleh program kurang dari 64 KB tetapi code yang digunakan bisa lebih dari 64 KB.

- COMPACT

Jika data yang digunakan bisa lebih besar dari 64 KB tetapi codenya kurang dari 64 KB.

- LARGE

Jika data dan code yang dipakai oleh program bisa lebih dari 64 KB.

- HUGE

Jika data, code maupun array yang digunakan bisa lebih dari 64 KB.

Mungkin ada yang bertanya-tanya mengapa pada program COM yang dibuat digunakan model SMALL dan bukannya TINY ? Hal ini disebabkan karena banyak dari compiler bahasa tingkat tinggi yang tidak bisa berkomunikasi dengan model TINY, sehingga kita menggunakan model SMALL sebagai pemecahannya.

6.1.2 .CODE

Tanda directive ini digunakan untuk memberitahukan kepada assembler bahwa kita akan mulai menggunakan Code Segment-nya disini. Code segment ini digunakan untuk menyimpan program yang nantinya akan dijalankan.

6.1.3. ORG 100h

Pada program COM perintah ini akan selalu digunakan. Perintah ini digunakan untuk memberitahukan assembler supaya program pada saat dijalankan(diload ke memory) ditaruh mulai pada offset ke 100h(256) byte. Dapat dikatakan juga bahwa kita menyediakan 100h byte kosong pada saat program dijalankan. 100h byte kosong ini nantinya akan ditempati oleh PSP(Program Segment Prefix) dari program tersebut. PSP ini digunakan oleh DOS untuk mengontrol jalannya program tersebut.

6.1.4. JMP

Perintah JMP(JUMP) ini digunakan untuk melompat menuju tempat yang ditunjukkan oleh perintah JUMP. Adapun syntaxnya adalah:

JUMP Tujuan .

Dimana tujuannya dapat berupa label seperti yang digunakan pada bagan diatas. Mengenai perintah JUMP ini akan kita bahas lebih lanjut nantinya.

Perintah JUMP yang digunakan pada bagan diatas dimaksudkan agar melewati tempat data program, karena jika tidak ada perintah JUMP ini maka data program akan ikut dieksekusi sehingga kemungkinan besar akan menyebabkan program anda menjadi Hang.

6.1.5. INT 20h

Perintah **INT** adalah suatu perintah untuk menghasilkan suatu interupsi dengan syntax:

INT NoInt

Interupsi 20h berfungsi untuk mengakhiri program dan menyerahkan kendali sepenuhnya kepada Dos. Pada program COM cara ini bukanlah satu-satunya tetapi cara inilah yang paling efektif untuk digunakan. Bila anda lupa untuk mengakhiri sebuah program maka program anda tidak akan tahu kapan harus selesai, hal ini akan menyebabkan komputer menjadi hang.

BAB 7
MENCETAK HURUF

7.1. MENCETAK HURUF

Bila dihasilkan interupsi 21h apa yang akan dikerjakan oleh komputer ?. Jawabnya, ada banyak sekali kemungkinan. Pada saat terjadi interupsi 21h maka pertama-tama yang dilakukan komputer adalah melihat isi atau nilai apa yang terdapat pada register AH. Misalkan bila nilai AH adalah 2 maka komputer akan mencetak sebuah karakter, berdasarkan kode ASCII yang terdapat pada register DL. Bila nilai pada register AH bukanlah 2, pada saat dilakukan interupsi 21h maka yang dikerjakaan oleh komputer akan lain lagi.

Dengan demikian kita bisa mencetak sebuah karakter yang diinginkan dengan meletakkan angka 2 pada register AH dan meletakkan kode ASCII dari karakter yang ingin dicetak pada register DL sebelum menghasilkan interupsi 21h.

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~;
;   PROGRAM : A0.ASM             ;
;   FUNGSI  : MENCETAK KARATER   ;
;             'A' DENGAN INT 21  ;

;=========================S'to=;

         .MODEL SMALL
         .CODE
         ORG 100h
Proses :
         MOV  AH,02h    ; Nilai servis ntuk mencetak karakter
         MOV  DL,'A'    ; DL = Karakter ASCII yang akan dicetak
         INT  21h       ; Cetak karakter !!

         INT  20h       ; Selesai ! kembali ke DOS
END      Proses
```

Program 7.1. Mencetak sebuah karakter

Setelah program 7.1. anda ketik compile-lah dengan menggunakan TASM.EXE dengan perintah :

```
C:\>tasm a0
Turbo Assembler  Version 2.0  Copyright (c) 1988, 1990
Borland International

Assembling file:   a0.ASM
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  306k

C:\>dir a0.*

Volume in drive C is S'to
Directory of C:\

A0       ASM           506 08-14-94   3:56p
A0       OBJ           179 08-14-94  11:24p
     2 file(s)             685 bytes
```

```
                1,267,200 bytes free
```

Sampai disini sudah dihasilkan suatu file object dari KAL.ASM yang siap dijadikan file COM(karena kita membuat file dengan format program COM). Untuk itu lakukanlah langkah kedua, dengan perintah :

```
C:\>tlink/t a0
Turbo Link  Version 3.0 Copyright (c) 1987, 1990
Borland International

C:\>tlink/t a0
Turbo Link  Version 3.0 Copyright (c) 1987, 1990
Borland International

C:\>dir a0.*

Volume in drive C is S'to
Directory of C:\

A0       ASM           506 08-14-94   3:56p
A0       OBJ           179 08-14-94  11:26p
A0       MAP           229 08-14-94  11:26p
A0       COM             8 08-14-94  11:26p
        4 file(s)            922 bytes
                      1,266,176 bytes free
```

Setelah kedua proses itu selesai maka dihasilkanlah suatu program COM yang sudah siap untuk dijalankan. File-file yang tidak digunakan bisa anda hapus. Bila program 7.1. dijalankan maka pada layar akan ditampilkan

```
C:\>A0
  A
```

Kita lihat disini bahwa karakter yang tercetak adalah yang sesuai dengan kode ASCII yang ada pada register DL. Sebagai latihan cobalah anda ubah register DL dengan angka 65 yang merupakan kode ASCII karakter 'A'. Hasil yang didapatkan adalah sama.

Pelajarilah fungsi ini baik-baik, karena akan banyak kita gunakan pada bab-bab selanjutnya.

7.2. MENCETAK KARAKTER BESERTA ATRIBUT

Sebuah karakter disertai dengan warna tentunya akan lebih menarik. Untuk itu anda bisa menggunakan interupsi ke 10h dengan aturan pemakaiannya :

```
INPUT
   AH = 09h
   AL = Kode ASCII dari karakter yang akan dicetak
   BH = Nomor halaman(0 untuk halaman 1)
   BL = Atribut atau warna dari karakter yang akan dicetak
   CX = Banyaknya karakter tersebut akan dicetak
```

Setelah semua register dimasukkan nilainya maka lakukanlah interupsi 10h. Perlu anda perhatikan bahwa interupsi ini mencetak karakter tanpa menggerakkan kursor.

```
;================================;
;   PROGRAM : A1.ASM             ;
;   FUNGSI  : MENCETAK KARATER 'A';
;             BESERTA ATRIBUTNYA  ;
;             DENGAN INT 10h      ;
;============================S'to=;

         .MODEL SMALL
         .CODE
         ORG 100h
Proses :
         MOV  AH,09h    ; Nilai servis untuk mencetak karakter
         MOV  AL,'A'    ; AL = Karakter yang akan dicetak
         MOV  BH,00h    ; Nomor Halaman layar
         MOV  BL,93h    ; Warna atau atribut dari karakter
         MOV  CX,03h    ; Banyaknya karakter yang ingin dicetak
         INT  10h       ; Laksanakan !!!

         INT  20h       ; Selesai ! kembali ke DOS
END      Proses
```

Program 7.2. Mencetak karakter beserta atributnya

Bila program 7.2. dieksekusi maka akan ditampilkan huruf 'A' sebanyak 3 kali dengan warna dasar biru kedip dan warna tulisan Cyan(sesuai dengan nilai register BL). Mengenai halaman layar dan atribut(warna) akan kita bahas lebih lanjut pada bagian yang lain.

```
B:\>A1

AAA
```

Anda bisa merubah-rubah register AL dan BL untuk merubah karakter dan warna yang ingin dicetak.

7.3. PENGULANGAN DENGAN LOOP

Perintah LOOP digunakan untuk melakukan suatu proses yang berulang-ulang. Adapun syntax dari perintah ini adalah :

LOOP Tujuan

Tujuan dapat berupa suatu label yang telah didefinisikan, contoh:

```
         MOV  CX,3      ; Banyaknya pengulangan yang dilakukan
Ulang :  INT  10h       ; Tempat terjadinya pengulangan
         LOOP Ulang     ; Lompat ke label 'Ulang'
```

Pada proses pengulangan dengan perintah LOOP, register CX memegang suatu peranan yang khusus dimana register ini dijadikan sebagai counter/penghitung terhadap banyaknya looping yang boleh terjadi. Setiap ditemui perintah LOOP, maka register CX akan dikurangi dengan 1 terlebih dahulu, kemudian akan dilihat apakah CX sudah mencapai 0. Proses looping akan selesai bila nilai

pada register CX mencapai nol. Seperti pada contoh diatas, maka interupsi 10h akan dihasilkan sebanyak 3 kali(sesuai dengan nilai CX).

Perlu diperhatikan bahwa jangan sampai anda menaruh CX kedalam proses LOOP karena hal ini akan menyebabkan nilai CX diSET terus sehingga proses looping tidak bisa berhenti.

TRICK:

Bila anda menetapkan nilai CX menjadi nol pada saat pertama kali sebelum dilakukan loop, maka anda akan mendapatkan proses looping yang terbanyak. Hal ini dikarenakan proses pengurangan 0 dengan 1 akan menghasilkan nilai FFFFh(-1), Contoh :

```
       MOV  CX,00
Ulang: LOOP Ulang
```

7.4. MENCETAK BEBERAPA KARAKTER

Untuk mencetak beberapa karakter, bisa anda gunakan proses looping. Sebagai contoh dari penggunaan looping ini bisa dilihat pada program 7.3.

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~;
;   PROGRAM : ABC0.ASM          ;
;   FUNGSI  : MENCETAK 16 BUAH  ;
;             KARAKTER  DENGAN  ;
;             INT 21h SERVIS 02 ;
;==========================S'to=;

         .MODEL SMALL
         .CODE
         ORG 100h
Proses :
         MOV  AH,02h    ; Nilai servis
         MOV  DL,'A'    ; DL=karakter 'A' atau DL=41h
         MOV  CX,10h    ; Banyaknya pengulangan yang akan
 Ulang :
         INT  21h       ; Cetak karakter !!
         INC  DL        ; Tambah DL dengan 1
         LOOP Ulang     ; Lompat ke Ulang

         INT  20h
END      Proses
```

Program 7.3. Pengulangan dengan perintah LOOP

Bila program 7.3. dijalankan maka akan ditampilkan :

```
ABCDEFGHIJKLMNOP
```

Perintah **INC DL** akan menambah register DL dengan 1, seperti intruksi , DL:=DL+1 dalam Pascal. Lebih jauh mengenai operasi aritmatika(INC) ini akan dibahas pada bab selanjutnya.

BAB VIII
OPERASI ARITMATIKA

8.1. OPERASI PERNAMBAHAN

8.1.1. ADD

Untuk menambah dalam bahasa assembler digunakan perintah **ADD** dan **ADC** serta **INC**. Perintah ADD digunakan dengan syntax :

ADD Tujuan,Asal

Perintah ADD ini akan menambahkan nilai pada Tujuan dan Asal. Hasil yang didapat akan ditaruh pada Tujuan, dalam bahasa pascal sama dengan instruksi **Tujuan:=Tujuan + Asal**. Sebagai contohnya :

```
MOV AH,15h   ; AH:=15h
MOV AL,4     ; AL:=4
ADD AH,AL    ; AH:=AH+AL, jadi AH=19h
```

Perlu anda perhatikan bahwa pada perintah ADD ini antara Tujuan dan Asal harus mempunyai daya tampung yang sama, misalnya register AH(8 bit) dan AL(8 bit), AX(16 bit) dan BX(16 bit).

Mungkin ada yang bertanya-tanya, apa yang akan terjadi bila Tujuan tempat hasil penjumlahan disimpan tidak mencukupi seperti pertambahan 1234h dengan F221h.

```
 1234 h        Biner -->  0001 0010 0011 0100
 F221 h        Biner -->  1111 0010 0010 0001
---------- +             -------------------- +
10455 h                 1 0000 0100 0101 0101
```

Pada pertambahan diatas dapat dilihat bahwa pertambahan bilangan 1234 dengan F221 akan menghasilkan nilai 10455. Supaya lebih jelas dapat anda lihat pada pertambahan binernya dihasilkan bit ke 17, padahal register terdiri atas 16 bit saja. Operasi pertambahan yang demikian akan menjadikan carry flag menjadi satu, Contoh :

```
MOV AX,1234h   ; NIlai AX:=1234h dan carry=0
MOV BX,0F221h    ; Nilai BX:=F221h dan carry=0
ADD AX,BX        ; Nilai AX menjadi 0455h dan carry=1
```

8.1.2. ADC

Perintah ADC digunakan dengan cara yang sama pada perintah ADD, yaitu :

ADC Tujuan,Asal

Perbedaannya pada perintah ADC ini Tujuan tempat menampung hasil pertambahan Tujuan dan Asal ditambah lagi dengan carry flag (Tujuan:=Tujuan+Asal+Carry). Pertambahan yang demikian bisa memecahkan masalah seperti yang pernah kita kemukakan, seperti pertambahan pada bilangan 12345678h+9ABCDEF0h.

Seperti yang telah kita ketahui bahwa satu register hanya mampu menampung 16 bit, maka untuk pertambahan seperti yang diatas bisa anda gunakan perintah ADC untuk memecahkannya, Contoh:

```
MOV AX,1234h     ; AX = 1234h  CF = 0
MOV BX,9ABCh     ; BX = 9ABCh  CF = 0
```

```
        MOV CX,5678h      ; BX = 5678h  CF = 0
        MOV DX,0DEF0h     ; DX = DEF0h  CF = 0
        ADD CX,DX         ; CX = 3568h  CF = 1
        ADC AX,BX         ; AX = AX+BX+CF  = ACF1
```

Hasil penjumlahan akan ditampung pada register AX:CX yaitu ACF13568h.

Adapun flag-flag yang terpengaruh oleh perintah ADD dan ADC ini adalah CF,PF,AF,ZF,SF dan OF.

8.1.3. INC

Perintah INC(Increment) digunakan khusus untuk pertambahan dengan 1. Perintah INC hanya menggunakan 1 byte memory, sedangkan perintah ADD dan ADC menggunakan 3 byte. Oleh sebab itu bila anda ingin melakukan operasi pertambahan dengan 1 gunakanlah perintah INC. Syntax pemakainya adalah :

INC Tujuan

Nilai pada tujuan akan ditambah dengan 1, seperti perintah Tujuan:=Tujuan+1 dalam Turbo Pascal. Tujuan disini dapat berupa suatu register maupun memory. Contoh : perintah INC AL akan menambah nilai di register AL dengan 1. Adapun flag yang terpengaruh oleh perintah ini adalah OF,SF,ZF,AF dan PF.

8.1.4. PROGRAM PENAMBAHAN DAN DEBUG

Setelah apa yang telah kita pelajari, marilah sekarang kita menjadikannya sebuah program dengan semua contoh yang telah diberikan.

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~;
;   PROGRAM : TAMBAH.ASM         ;
;   FUNGSI  : MELIHAT PENAMBAHAN ;
;             YANG DILAKUKAN     ;
;             OLEH BERBAGAI      ;
;             PERINTAH           ;
;============================S'to=;

          .MODEL SMALL
          .CODE
          ORG 100h
 Proses :
          MOV AH,15h      ; AH:=15h
          MOV AL,4        ; AL:=4
          ADD AH,AL       ; AH:=AH+AL, jadi AH=19h

          MOV AX,1234h    ; NIlai AX:=1234h dan carry=0
          MOV BX,0F221h   ; Nilai BX:=F221h dan carry=0
          ADD AX,BX       ; AX:=AX+BX, jadi nilai AX=0455h

          MOV AX,1234h    ; AX = 1234h  CF = 0
          MOV BX,9ABCh    ; BX = 9ABCh  CF = 0
          MOV CX,5678h    ; BX = 5678h  CF = 0
          MOV DX,0DEF0h   ; DX = DEF0h  CF = 0
          ADD CX,DX       ; CX = 3568h  CF = 1
          ADC AX,BX       ; AX = AX+BX+CF = ACF1

          INC AL          ; AL:=AL+1, nilai pada AL ditambah 1

          INT  20h
END       Proses
```

Program 8.1. Operasi penambahan

Setelah anda selesai mengetikkan program 8.1., jadikanlah program COM dengan tasm dan tlink/t.Setelah itu cobalah untuk melihat kebenaran dari apa yang sudah diberikan dengan menggunakan debug. Pertama-tama ketikkanlah :

```
C:\>debug Tambah.com
-r      < tekan enter >
AX=0000 BX=0000 CX=0030 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=3597 ES=3597 SS=3597 CS=3597 IP=0100  NV UP EI PL NZ NA PO NC
3597:0100 B415          MOV     AH,15
-t      < tekan enter >
```

Penekanan "**r**" pada saat pertama kali digunakan untuk melihat nilai pada semua register. Pada baris pertama dapat anda lihat register yang dinamakan sebagai general purpose(AX,BX,CX dan DX). Register SP yang digunakan pada operasi stack menunjukkan nilai FFFE(akhir dari Segment), jadi operasi stack nantinya akan ditaruh pada posisi tersebut.

Pada baris kedua dapat anda lihat keempat register segment, yaitu DS,ES,SS dan CS. Keempat register segment menunjukkan nilai yang sama yaitu 3597(mungkin berbeda pada komputer anda). Hal ini dikarenakan program kita adalah program com yang hanya menggunakan 1 segment. Pada baris kedua dapat juga anda lihat register IP bernilai 100h. Register IP menunjukkan bahwa kita sekarang sedang berada pada offset ke 100h dari segment aktif(CS:IP atau 3597:100).

Pada baris ketiga dapat anda lihat 3597:0100, nilai ini menunjukkan pasangan dari CS:IP. Setelah itu dapat anda lihat nilai B415 yang menujukkan isi dari alamat 3597:0100 adalah B4 sedangkan isi dari alamat 3597:1001 adalah 15. Nilai B415 ini sebenarnya merupakan suatu bahasa mesin untuk instruksi MOV AH,15. Jadi bahasa mesin untuk perintah **"MOV AH,nilai"** adalah B4 disertai nilai tersebut. Dari nilai B415 ini dapat diketahui bahwa perintah MOV akan menggunakan 2 byte di memory.

Setelah itu tekanlah 't' untuk mengeksekusi intruksi yang terdapat pada alamat yang ditunjukkan CS:IP(MOV AH,15). Setelah anda menekan 't' maka akan ditampilkan hasil setelah intruksi "MOV AH,15" dieksekusi :

```
AX=1500 BX=0000 CX=0030 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=3597 ES=3597 SS=3597 CS=3597 IP=0102  NV UP EI PL NZ NA PO NC
3597:0102 B004          MOV     AL,04
-t        < enter >
```

Terlihat bahwa nilai AX berubah dari 0000 menjadi 1500 setelah mendapat perintah MOV AH,15. Tekanlah 't' disertai enter untuk melihat perubahan nilai pada register-register yang bersangkutan.

```
AX=1504 BX=0000 CX=0030 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=3597 ES=3597 SS=3597 CS=3597 IP=0104  NV UP EI PL NZ NA PO NC
3597:0104 02E0          ADD     AH,AL
-t        < enter >

AX=1904 BX=0000 CX=0030 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
```

```
DS=3597 ES=3597 SS=3597 CS=3597 IP=0106  NV UP EI PL NZ NA PO NC
3597:0106 B83412       MOV     AX,1234
-t        < enter >

AX=1234 BX=0000 CX=0030 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=3597 ES=3597 SS=3597 CS=3597 IP=0109  NV UP EI PL NZ NA PO NC
3597:0109 BB21F2       MOV     BX,F221
-t        < enter >

AX=1234 BX=F221 CX=0030 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=3597 ES=3597 SS=3597 CS=3597 IP=010C  NV UP EI PL NZ NA PO NC
3597:010C 03C3         ADD     AX,BX
-t        < enter >

AX=0455 BX=F221 CX=0030 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=3597 ES=3597 SS=3597 CS=3597 IP=010E  NV UP EI PL NZ NA PE CY
3597:010E B83412       MOV     AX,1234
-t        < enter >

AX=1234 BX=F221 CX=0030 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=3597 ES=3597 SS=3597 CS=3597 IP=0111  NV UP EI PL NZ NA PE CY
3597:0111 BBBC9A       MOV     BX,9ABC
-t        < enter >

AX=1234 BX=9ABC CX=0030 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=3597 ES=3597 SS=3597 CS=3597 IP=0114  NV UP EI PL NZ NA PE CY
3597:0114 B97856       MOV     CX,5678
-t        < enter >

AX=1234 BX=9ABC CX=5678 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=3597 ES=3597 SS=3597 CS=3597 IP=0117  NV UP EI PL NZ NA PE CY
3597:0117 BAF0DE       MOV     DX,DEF0
-t        < enter >

AX=1234 BX=9ABC CX=5678 DX=DEF0 SP=FFFE BP=0000 SI=0000 DI=0000
DS=3597 ES=3597 SS=3597 CS=3597 IP=011A  NV UP EI PL NZ NA PE CY
3597:011A 03CA         ADD     CX,DX
-t        < enter >

AX=1234 BX=9ABC CX=3568 DX=DEF0 SP=FFFE BP=0000 SI=0000 DI=0000
DS=3597 ES=3597 SS=3597 CS=3597 IP=011C  NV UP EI PL NZ NA PO CY
3597:011C 13C3         ADC     AX,BX
-t        < enter >

AX=ACF1 BX=9ABC CX=3568 DX=DEF0 SP=FFFE BP=0000 SI=0000 DI=0000
DS=3597 ES=3597 SS=3597 CS=3597 IP=011E  NV UP EI NG NZ AC PO NC
3597:011E FEC0         INC     AL
-t        < enter >

AX=ACF2 BX=9ABC CX=3568 DX=DEF0 SP=FFFE BP=0000 SI=0000 DI=0000
DS=3597 ES=3597 SS=3597 CS=3597 IP=0120  NV UP EI NG NZ NA PO NC
3597:0120 CD20          INT    20
-Q        < enter >
```

Pengetikan "Q" menandakan bahwa kita ingin keluar dari lingkungan debug dan akan kembali ke a:\>.

8.2. OPERASI PENGURANGAN

8.2.1. SUB

Untuk Operasi pengurangan dapat digunakan perintah SUB dengan syntax:

SUB Tujuan,Asal

Perintah SUB akan mengurangkan nilai pada Tujuan dengan Asal. Hasil yang didapat akan ditaruh pada Tujuan, dalam bahasa pascal sama dengan instruksi **Tujuan:=Tujuan-Asal**.

Contoh :

```
MOV AX,15   ; AX:=15
MOV BX,12   ; BX:=12
SUB AX,BX   ; AX:=15-12=3
SUB AX,AX   ; AX=0
```

Untuk menolkan suatu register bisa anda kurangkan dengan dirinya sendiri seperti SUB AX,AX.

8.2.2. SBB

Seperti pada operasi penambahan, maka pada operasi pengurangan dengan bilangan yang besar(lebih dari 16 bit), bisa anda gunakan perintah SUB disertai dengan SBB(Substract With Carry). Perintah SBB digunakan dengan syntax:

SBB Tujuan,Asal

Perintah SBB akan mengurangkan nilai Tujuan dengan Asal dengan cara yang sama seperti perintah SUB, kemudian hasil yang didapat dikurangi lagi dengan Carry Flag(Tujuan:=Tujuan-Asal-CF).

```
;================================;
;   PROGRAM : KURANG.ASM         ;
;   AUTHOR  : S'to               ;
;   FUNGSI  : MENGURANGKAN ANGKA ;
;             122EFFF-0FEFFFF    ;
;                                ;
;================================;

          .MODEL SMALL
          .CODE
          ORG 100h
TData :
          JMP   Proses           ; Lompat ke Proses
          ALo   EQU 0EFFFh
          AHi   EQU 122h
          BLo   EQU 0FFFFh
          Bhi   EQU 0FEh
          HslLo DW  ?
          HslHi DW  ?
Proses :
          MOV  AX,ALo        ; AX=EFFFh
          SUB  AX,Blo        ; Kurangkan EFFF-FFFF, jadi AX=F000
          MOV  HslLO,AX      ; HslLo bernilai F000
          MOV  AX,AHi        ; AX=122h
          SBB  AX,BHi        ; Kurangkan 122-FE-Carry, AX=0023
          MOV  HslHi,AX      ; HslHi bernilai 0023
```

```
        INT 20h             ; Kembali ke DOS
END     TData
```

Program 8.2. Mengurangkan angka yang menyebabkan Borrow

Disini dapat kita lihat begaimana caranya mendefinisikan suatu nilai constanta(nilai yang tidak dapat dirubah) dan variabel dengan :

```
ALo   EQU 0EFFFh
AHi   EQU 122h
BLo   EQU 0FFFFh
Bhi   EQU 0FEh
HslLo DW  ?
HslHi DW  ?
```

Perintah EQU digunakan untuk mendefisisikan suatu yang constan(Tetap), data yang telah didefinisikan dengan perintah EQU tidak dapat dirubah. Dengan perintah EQU kita mendefinisikan bahwa ALo = 0EFFF, AHi=122, BLo=FFFF dan BHi=0FE. Untuk menampung hasil dari pengurangan A-B(122EFFF-FEFFF) nantinya, kita definisikan suatu tempat untuk menyimpannya dengan nama HslLo dan HslHi. Tanda '?' digunakan untuk menyatakan bahwa tempat yang kita pesan sebanyak sebanyak 1 word(**DW**) tidak diberikan data awal yang akan terdapat pada varibel tersebut(HslLo dan HslHi). Jadi data yang akan terdapat pada HslLo dan HslHi bisa apa saja dan kita tidak mengetahuinya. Tempat data program kita lompati dengan perintah JMP supaya komputer tidak mengeksekusi data program sebagai perintah.

```
MOV  AX,ALo
SUB  AX,Blo
MOV  HslLO,AX
```

Untuk mengurangkan angka 122EFFF dengan 0FEFFFF kita dapat mengurangkan word rendah dari angka tersebut dahulu, yaitu EFFF- FFFF. Hal ini dikarenakan daya tampung register yang hanya 16 bit. Dapat anda lihat bahwa pengurangan EFFF-FFFF akan menyebabkan terjadinya peminjaman(Borrow), hasil word rendah(F000) yang didapatkan kemudian kita simpan pada variabel HslLo.

```
 122 EFFF
  FE FFFF
---------- -
 023 F000
```

Sampai saat ini kita sudah selesai mendapatkan nilai pada word rendahnya, yang disimpan pada variabel HslLo.

```
MOV  AX,AHi
SBB  AX,BHi
MOV  HslHi,AX
```

Langkah selanjutnya adalah menghitung word tingginya yaitu pengurangan 122-FE-Carry dengan menggunakan perintah SBB maka masalah tersebut dengan mudah terpecahkan. Akhirnya kita akan mendapatkan hasil pengurangan dari 122EFFF-0FEFFFF yaitu 23F000 yang tersimpan pada pasangan HslHi:HslLo(0023F000).

8.2.3. DEC

Perintah DEC(Decrement) digunakan khusus untuk pengurangan dengan 1. Perintah DEC hanya menggunakan 1 byte memory, sedangkan perintah SUB dan SBB menggunakan 3 byte. Oleh sebab itu bila anda ingin melakukan operasi pengurangan dengan 1 gunakanlah perintah DEC. Syntax pemakaian perintah dec ini adalah:

DEC Tujuan

Nilai pada tujuan akan dikurangi 1, seperti perintah Tujuan:=Tujuan-1 dalam Turbo Pascal. Tujuan disini dapat berupa suatu register maupun memory. Contoh : perintah DEC AL akan mengurangi nilai di register AL dengan 1.

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~;
;   PROGRAM : CBA0.ASM          ;
;   FUNGSI  : MENCETAK KARAKTER ;
;             "Z".."A" DENGAN   ;
;             INT 21h SERVIS 02 ;
;==========================S'to=;

         .MODEL SMALL
         .CODE
         ORG 100h
Proses :
         MOV  AH,02h    ; Nilai servis
         MOV  DL,'Z'    ; DL=5Ah
         MOV  CX,26     ; Banyaknya pengulangan yang akan
                        ; dilakukan
  Ulang:
         INT  21h       ; Cetak karakter !!
         DEC  DL        ; Kurang DL dengan 1
         LOOP Ulang     ; Lompat ke Ulang

         INT  20h
END      Proses
```

Program 8.3. Mencetak karakter "Z".."A"

Program 8.3. bila dijalankan akan mencetak karakter "Z" sampai "A" sebagai berikut :

ZYXWVUTSRQPONMLKJIHGFEDCBA

8.3. OPERASI PERKALIAN

Untuk perkalian bisa digunakan perintah MUL dengan syntax:

MUL Sumber

Sumber disini dapat berupa suatu register 8 bit(Mis:BL,BH,..), register 16 bit(Mis: BX,DX,..) atau suatu varibel. Ada 2 kemungkinan yang akan terjadi pada perintah MUL ini sesuai dengan jenis perkalian 8 bit atau 16 bit.

Bila Sumber merupakan 8 bit seperti **MUL BH** maka komputer akan mengambil nilai yang terdapat pada BH dan nilai pada AL untuk dikalikan. Hasil yang didapat akan selalu disimpan pada register AX. Bila sumber merupakan 16 bit

seperti **MUL BX** maka komputer akan mengambil nilai yang terdapat pada BX dan nilai pada AX untuk dikalikan. Hasil yang didapat akan disimpan pada register DX dan AX(DX:AX), jadi register DX menyimpan Word tingginya dan AX menyimpan Word rendahnya.

```
;================================;
;   PROGRAM : KALI.ASM           ;
;   AUTHOR  : S'to               ;
;   FUNGSI  : MENGALIKAN BILANGAN;
;             16 BIT, HASIL      ;
;             PADA DX:AX         ;
;================================;

        .MODEL SMALL
        .CODE
        ORG 100h
TData :
         JMP   Proses           ; Lompat ke Proses
         A     DW  01EFh
         B     DW  02FEh
         HslLo DW  ?
         HslHi DW  ?
Proses:
        MOV  AX,A         ; AX=1EF
        MUL  B            ; Kalikan 1FH*2FE
        MOV  HslLo,AX     ; AX bernilai C922 sehingga HslLo=C922
        MOV  HslHi,DX     ; DX bernilai 0005 sehingga HslHi=0005

        INT 20h           ; Kembali ke DOS
END     TData
```

Program 8.4. Proses perkalian dengan MUL

Pada program 8.4. kita mendefinisikan angka untuk variabel 'A'=1EF dan 'B'=2FE dengan DW. Karena tidak digunakan EQU, maka varibel 'A' dan 'B' dapat dirubah bila diinginkan.

8.4. PEMBAGIAN

Operasi pada pembagian pada dasarnya sama dengan perkalian. Untuk operasi pembagian digunakan perintah DIV dengan syntax:

DIV Sumber

Bila **sumber** merupakan operand 8 bit seperti **DIV BH**, maka komputer akan mengambil nilai pada register AX dan membaginya dengan nilai BH. Hasil pembagian 8 bit ini akan disimpan pada register AL dan sisa dari pembagian akan disimpan pada register AH.

Bila **sumber** merupakan operand 16 bit seperti **DIV BX**, maka komputer akan mengambil nilai yang terdapat pada register DX:AX dan membaginya dengan nilai BX. Hasil pembagian 16 bit ini akan disimpan pada register AX dan sisa dari pembagian akan disimpan pada register DX.

```
;================================;
```

```
;   PROGRAM : BAGI.ASM           ;
;   AUTHOR  : S'to               ;
;   FUNGSI  : MEMBAGI BILANGAN   ;
;             16 BIT, HASIL      ;
;             PADA DX:AX         ;
;================================;

         .MODEL SMALL
         .CODE
         ORG 100h
TData :
        JMP   Proses           ; Lompat ke Proses
        A     DW  01EFh
        B     DW  2
        Hsl   DW  ?
        Sisa  DW  ?
Proses:
        SUB  DX,DX       ; Jadikan DX=0
        MOV  AX,A        ; AX=1EF
        DIV  B           ; Bagi 1EF:2
        MOV  Hsl,AX      ; AX bernilai 00F7 sehingga Hsl=00F7
        MOV  Sisa,DX     ; DX berisi 0001 sehingga Sisa=0001

        INT 20h          ; Kembali ke DOS
END     Tdata
```

Program 8.5. Proses pembagian bilangan 16 bit

Cobalah anda trace dengan debug untuk melihat hasil yang didapat pada register AX dan DX.

BAB IX
POINTER

9.1. PENDAHULUAN

Pada program-program sebelumnya(pengurangan,perkalian dan pembagian) dapat anda lihat bahwa hasil dari operasi aritmatika disimpan dalam 2 variabel dimana 1 variabel untuk menampung hasil dari word tingginya dan 1 word untuk menampung word rendahnya. Bukankah hal ini akan tampak menjadi aneh, karena bila kita ingin melihat nilai tersebut maka nilai tersebut harus disatukan barulah dapat dibaca. Apakah ada cara lain supaya hasilnya dapat disimpan pada satu variabel saja ? YA!!, tetapi untuk itu anda harus menggunakan pointer untuk mengaksesnya. Bila anda tidak menggunakan pointer maka tipe data penampung harus sesuai dengan registernya. Tanpa pointer untuk memindahkan data dari suatu variabel ke register 8 bit, maka variabel tersebut haruslah 8 bit juga yang dapat didefinisikan dengan **DB**, demikian juga untuk register 16 bit dengan variabel yang didefinisikan dengan **DW**. Contoh :

```
A  DB 17    ; DB=8 bit jadi A=8 bit
B  DW 35    ; DW=16 bit jadi B=16 bit
   :
MOV AL,A    ; 8 bit dengan 8 bit
MOV AX,B    ; 16 bit dengan 16 bit.
```

Seperti pada contoh diatas anda tidak bisa menggunakan perintah MOV AX,A karena kedua operand tidak mempunyai daya tampung yang sama(16 dan 8 bit). Bila anda melakukan pemindahan data dari operand yang berbeda tipe data penampungnya maka akan ditampikan "****Error** BAGI.ASM(20) Operand types do not match**". Dengan menggunakan pointer hal ini bukanlah masalah. Sebelum itu marilah kita lihat dahulu berbagai tipe data yang terdapat pada assembler.

9.2. TIPE DATA

Didalam assembler kita bisa menyimpan data dengan berbagai tipe data yang berbeda-beda. Kita dapat memberikan nama pada data tersebut, untuk memudahkan dalam pengaksesan data tersebut. Adapun tipe data yang terdapat pada assembler dapat anda lihat pada gambar 9.1.

NAMA	UKURAN

```
DB<Define Byte>           1 BYTE
DW<Define Word>           2 BYTE
DD<Define DoubleWord>     4 BYTE
DF<Define FarWords>       6 BYTE
DQ<Define QuadWord>       8 BYTE
DT<Define TenBytes>      10 BYTE
-----------------------------------------
```

Gambar 9.1. Berbagai Tipe Data

Sebagai contohnya lihatlah bagaimana tipe data pada gambar 9.1. digunakan :

```
          .MODEL SMALL
          .CODE
          ORG 100h

TData :
             JMP Proses
             A   DB  4           ; 1 byte, nilai awal='4'
             B   DB  4,4,4,2,?   ; 1*5 byte, nilai awal=4,4,4,2,?
             C   DB  4 DUP(5)    ; 1*4 byte, nilai awal='5'
             D   DB  'HAI !!'    ; 6 byte berisi 6 karakter
             E   DW  ?           ; 1 word tidak diketahui isinya
             F   DW  ?,?,?       ; 3 word tidak diketahui isinya
             G   DW  10 DUP(?)   ;10 word tidak diketahui isinya
             H   DD  ?           ; 1 DoubleWord tanpa nilai awal
             I   DF  ?,?         ; 2 FarWord tanpa nilai awal
             J   DQ  0A12h       ; 1 QuadWord, nilai awal='0A12'
             K   DT  25*80       ; 1 TenBytes, nilai awal='2000'
             L   EQU 666         ; Konstanta, L=666
             M   DB  '123'         ; String '123'
             N   DB  '1','2','3'  ; String '123'
             O   DB   49,50,51    ; String '123'

Proses :     ;
             ;
             ;
END         Tdata
```

Pada baris pertama("**A DB 4**") kita mendefinisikan sebanyak satu byte untuk variabel dengan nama "**A**", variabel ini diberi nilai "4".

Pada baris kedua("**B DB 4,4,4,2,?**") kita mendefinisikan sebanyak 5 byte yang berpasangan untuk variabel dengan nama "**B**". Tiga byte pertama pada variabel "B" tersebut semuanya diberi nilai awal "4", byte ke empat diberi nilai awal 2 sedangkan byte ke lima tidak diberi nilai awal.

Pada baris ketiga("**C DB 4 DUP(5)**") kita mendefinisikan sebanyak 4 byte data yang diberi nilai awal "5" semuanya (DUP=Duplikasi). Jadi dengan perintah DUP kita dapat mendefinisikan suatu Array.

Pada baris keempat("**D DB 'HAI !! '**") kita mendefinisikan suatu string

dengan DB. Untuk mendefinisikan string selanjutnya akan selalu kita pakai tipe data DB. Bila kita mendefinisikan string dengan DW maka hanya 2 karakter yang dapat dimasukkan, format penempatan dalam memorypun nantinya akan membalikkan angka tersebut.

Pada baris kelima("**E DW ?**") kita mendefinisikan suatu tipe data Word yang tidak diberi nilai awal. Nilai yang terdapat pada variabel "E" ini bisa berupa apa saja, kita tidak perduli.

Pada baris keduabelas("**L EQU 666**") kita mendefinisikan suatu konstanta untuk variabel "L", jadi nilai pada "L" ini tidak dapat dirubah isinya.

Pada variabel M, N, O kita mendefinisikan suatu string "123" dalam bentuk yang berbeda. Ketiganya akan disimpan oleh assembler dalam bentuk yang sama, berupa angka 49, 50 dan 51.

Pada program-program selanjutnya akan dapat anda lihat bahwa kita selalu melompati daerah data("**TData:JMP Proses**"), mengapa demikian ? Bila kita tidak melompati daerah data ini maka proses akan melalui daerah data ini. Data-data program akan dianggap oleh komputer sebagai suatu intruksi yang akan dijalankan sehingga apapun mungkin bisa terjadi disana. Sebagai contohnya akan kita buat sebuah program yang tidak melompati daerah data, sehingga data akan dieksekusi sebagai intruksi. Program ini telah diatur sedemikian rupa untuk membunyikan speaker anda, pada akhir data diberi nilai CD20 yang merupakan bahasa mesin dari intruksi INT 20h.

```
;====================================;
;      PROGRAM : BHSMESIN.ASM        ;
;      AUTHOR  : S'to                ;
;      FUNGSI  : MEMBUNYIKAN SPEAKER ;
;                DENGAN DATA PROGRAM ;
;                                    ;
;====================================

     .MODEL SMALL
     .CODE
     ORG 100h

Tdata:DB 0E4h,61h,24h,0FEh,0E6h,61h,0B9h,0D0h,7h,0BBh,9Ah
      DB 2h,8Bh,0D1h,51h,34h,2h,0E6h,61h,0D1h,0C3h,73h,6h
      DB 83h,0C1h,0h,0EBh,0Bh,90h,52h,2Bh,0D1h,87h,0D1h,5Ah
      DB 81h,0C1h,34h,8h,0E2h,0FEh,59h,0E2h,0E2h,0CDh,20h
END   Tdata
```

Program 9.1. Program Yang Mengeksekusi Daerah Data

9.3. PENYIMPANAN DATA DALAM MEMORY

Sebelum kita melangkah lebih jauh, marilah kita lihat dahulu bagaimana komputer menyimpan suatu nilai didalam memory. Untuk itu ketikkanlah program

9.2. ini yang hanya mendefinisikan data.

```
          .MODEL SMALL
          .CODE
          ORG 100h

TData :  JMP Proses
             A   DB  12h,34h
             B   DW  0ABCDh
             C   DD  56789018h
             D   DB  40 DUP(1)

END      Tdata
```

Program 9.2. Mendefinisikan Data

Setelah program 9.2. diketikkan dan dijadikan COM dengan TASM.EXE dan TLINK.EXE, pakailah debug untuk melihat bagaimana data tersebut disimpan didalam memory komputer.

```
C:\>debug data.com     A     B        C           D
-d                   +-+-+ +-+-+ +----+----+ +----+--------
3001:0100  EB 31 90 12 34 CD AB 18-90 78 56 01 01 01 01 01
3001:0110  01 01 01 01 01 01 01 01-01 01 01 01 01 01 01 01
3001:0120  01 01 01 01 01 01 01 01-01 01 01 01 01 01 01 01
3001:0130  01 01 01 E0 AC 91 51 AD-8B C8 25 0F 00 8B D9 B1
3001:0140  04 D3 EB D1 E3 26 03 1E-64 01 8B 17 06 1F BF 04
3001:0150  00 57 BF FA 05 E8 83 0A-73 03 E8 63 0A 26 89 15
3001:0160  B9 FF FF EB 18 8E 06 82-01 2B DB 26 02 1C 7D 09
3001:0170  46 80 EB 80 8A FB 26 8A-1C 46 E8 16 DA 48 7D E5
-q
```

Gambar 9.2. Data program 9.2.

Ketiga byte pertama pada gambar 9.1. adalah bahasa mesin dari perintah **"JUMP PROSES"** dan **"NOP"**. Pada byte ke 4 dan ke 5 ini adalah data dari variabel **"A"**, dapat kita lihat bahwa data dari variabel **"A"**(1234) yang didefinisikan dengan **"DB"** disimpan didalam memory komputer sesuai dengan yang didefinisikan.

Dua byte selanjutnya(byte ke 6 dan 7), merupakan data dari variabel **C** yang telah kita definisikan dengan **"DW(2 byte)"**. Ternyata kedua byte dari variabel **"C"**(ABCD) disimpan didalam memory dalam urutan yang terbalik(CDAB) !. Mengapa demikian ?. Hal ini dikarenakan penyimpanan dimemory yang menyimpan nilai tingginya pada alamat tinggi. Anda dapat lihat pada ke 4 byte selanjutnya, yaitu data dari variabel **"D"** juga disimpan dengan susunan yang terbalik(56789018 menjadi 18907856).

9.4. MENGGUNAKAN POINTER

Kini kita sudah siap untuk melihat bagaimana memindahkan data dari variabel maupun register yang berbeda tipe datanya, dengan menggunakan pointer. Untuk itu digunakan perintah PTR dengan format penulisan :

TipeData PTR operand

Supaya lebih jelas, marilah kita lihat penggunaanya didalam program.

```
;================================;
;   PROGRAM : PTR.ASM            ;
;   AUTHOR  : S'to               ;
;   FUNGSI  : MEMINDAHKAN DATA   ;
;             ANTAR TIPE DATA YANG;
;             BERBEDA !!!        ;
;================================;

         .MODEL SMALL
         .CODE
         ORG 100h

TData :
        JMP   Proses           ; Lompat ke Proses
          A     DW  01EFh      ; 2 Byte
          B     DW  02FEh      ; 2 Byte
          D     DD  ?          ; 4 Byte
Proses:
        MOV  AL,BYTE PTR A     ; AL=EF, AX=?EF
        MOV  AH,BYTE PTR A+1   ; AH=01, AX=01EF

        MOV  BX,B              ; BX=02FE
        MOV  WORD PTR D,AX     ; D=??01EF
        MOV  WORD PTR D+2,BX   ; D=02FE01EF

        INT  20h               ; Kembali ke DOS
END     TData
```

Program 9.3. Menggunakan Pointer

Pada awalnya kita mendefinisikan variabel "A" dan "B" dengan tipe data word(16 bit) yang mempunyai nilai awal 01EF dan 02FE, serta variabel "C" dengan tipe data DoubleWord(32 bit) yang tidak diinialisasi.

```
MOV  AL,BYTE PTR A
```

```
MOV  AH,BYTE PTR A+1
```

Pada kedua perintah tersebut, kita memindahkan data dari variabel "A" ke register AX dengan byte per byte. Perhatikanlah bahwa kita harus menyesuaikan pemindahan data yang dilakukan dengan kemampuan daya tampungnya. Oleh sebab itu digunakan **"BYTE"** PTR untuk memindahkan data 1 byte menuju register 8 bit, dengan demikian untuk memindahkan data 16 bit harus digunakan **"WORD"** PTR. Pada baris pertama kita memindahkan byte rendah dari variabel **"A"** (EF) menuju register AL, kemudian pada baris kedua kita memindahkan byte tingginya(01) menuju register AH. Lihatlah kita menggunakan **"BYTE PTR A"** untuk nilai byte rendah dan **"BYTE PTR+1"** untuk byte tinggi dari variabel **"A"** dikarenakan penyimpanan data dalam memory komputer yang menyimpan byte tinggi terlebih dahulu(Lihat bagian 9.3.).

```
MOV  BX,B
```

```
MOV  WORD PTR D,AX
```

MOV WORD PTR D+2,BX

Pada bagian ini akan kita coba untuk memindahkan data dari 2 register 16 bit menuju 1 variabel 32 bit. Pada baris pertama **"MOV BX,B"** tentunya tidak ada masalah karena kedua operand mempunyai daya tampung yang sama. Pada baris kedua **"MOV WORD PTR D,AX"** kita memindahkan nilai pada register AX untuk disimpan pada variabel "D" sebagai word rendahnya. Kemudian pada baris ketiga **"MOV WORD PTR D+2,BX"** kita masukkan nilai dari register BX pada variabel "D" untuk word tingginya sehingga nilainya sekarang adalah BX:AX=02FE01EF. Perhatikanlah pada baris ketiga kita melompati 2 byte(WORD PTR+2) dari variabel "D" untuk menyimpan word tingginya.

Kini dengan menggunakan pointer ini kita bisa menyimpan hasil perkalian 16 bit didalam 1 varibel 32 bit. Untuk itu lihatlah program 9.4.

```
;================================;
;   PROGRAM : KALIPTR.ASM        ;
;   AUTHOR  : S'to               ;
;   FUNGSI  : MENGALIKAN BILANGAN;
;             16 BIT, HASIL      ;
;             PADA DX:AX         ;
;================================;

         .MODEL SMALL
         .CODE
         ORG 100h
TData :
        JMP   Proses            ; Lompat ke Proses
        A     DW  01EFh       ; 2 Byte
        B     DW  02FEh       ; 2 Byte
        Hsl   DD  ?           ; 4 Byte
Proses:
        MOV  AX,A               ; AX=1EF
        MUL  B                  ; Kalikan 1FH*2FE
        MOV  WORD PTR Hsl,AX    ; AX bernilai C922, Hsl=??C922
        MOV  WORD PTR Hsl+2,DX ; DX bernilai 0005, Hsl=0005C922

        INT 20h                 ; Kembali ke DOS
END     TData
```

Program 9.4. Menampung nilai 2 register dalam 1 variabel

BAB X

MANIPULASI BIT DAN LOGIKA

10.1. GERBANG NOT

Operator NOT akan menginvers suatu nilai seperti yang terlihat pada gambar 10.1.

A	Not (A)
0	1
1	0

Gambar 10.1. Tabel Operator NOT

Operasi Not di dalam assembler, digunakan dengan syntax :

NOT Tujuan,Sumber

Hasil dari operasi not ini akan disimpan pada **Tujuan**, sebagai contoh, instruksi **NOT AL,3Fh** akan menghasilkan nilai C0h bagi AL. Mungkin masih ada pembaca yang bingung dengan operasi ini. Baiklah untuk lebih jelasnya kita lihat operasi di atas secara per bit.

```
                3     F
             +--+-++--+-+
Bilangan  :  0011  1111
                C     0
             +--+-++--+-+
Not       :  1100  0000
```

10.2. GERBANG AND

Operator AND akan menghasilkan nilai nol bila salah satu operandnya bernilai nol. Dan hanya akan bernilai satu bila kedua operandnya bernilai satu.

A	B	A AND B
0	0	0

0	1	0
1	0	0
1	1	1

Gambar 10.2. Tabel Operator AND

Operasi AND di dalam assembler, digunakan dengan syntax :

AND Tujuan,Sumber

Hasil dari operasi AND ini akan disimpan pada **Tujuan**, sebagai contoh, instruksi :

```
MOV AL,3Fh
MOV BL,1Ah
AND AL,BL
```

Perintah diatas akan menghasilkan nilai 1A bagi register AL. **Ingatlah :**

Setiap bit yang di **AND** dengan 0 pasti menghasilkan bit 0 juga, sedangkan setiap bit yang di **AND** dengan 1 akan menghasilkan bit itu sendiri.

10.3. GERBANG OR

Operator logik OR akan menghasilkan nilai nol bila kedua operannya bernilai nol dan satu bila salah satunya bernilai satu.

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Gambar 10.3. Tabel Operator OR

Operasi OR di dalam assembler, digunakan dengan syntax :

OR Tujuan,Sumber

Hasil dari operasi OR ini akan disimpan pada **Tujuan**, sebagai contoh, instruksi :

```
MOV AL,3Fh
MOV BL,1Ah
```

OR AL,BL

Hasil operasi OR diatas akan menghasilkan nilai 3F bagi register AL.

Ingatlah :

Setiap bit yang di **OR** dengan 0 pasti menghasilkan bit itu sendiri, sedangkan setiap bit yang di **OR** dengan 1 pasti menghasilkan bit 1.

10.4. GERBANG XOR

Operator XOR akan menghasilkan nol untuk dua nilai yang sama nilainya dan satu untuk yang berbeda.

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Gambar 10.4. Tabel Operator XOR

Operasi XOR di dalam assembler, digunakan dengan syntax :

XOR Tujuan,Sumber

Hasil dari operasi XOR ini akan disimpan pada **Tujuan**, sebagai, contoh instruksi :

MOV AX,0A12h

XOR AX,AX

Hasil operasi XOR diatas pasti akan menghasilkan nilai 0 bagi register AX.

Ingatlah: Setiap bilangan yang di **XOR** dengan bilangan yang sama pasti menghasilkan bilangan 0.

10.5. TEST

Perintah Test digunakan untuk mengetahui nilai pada suatu bit, dengan syntax :

TEST Operand1,Operand2

Perintah test akan meng**AND** kedua nilai operand, tetapi hasil yang didapatkan tidak akan berpengaruh terhadap nilai kedua operand tersebut.

Setelah perintah Test dilaksanakan yang akan terpengaruh adalah Flags, sehingga perintah ini sering diikuti dengan perintah yang berhubungan dengan kondisi flags. Adapun flags yang terpengaruh adalah CF,OF,PF,ZF,SF dan AF.

```
TEST AX,0Fh
JNZ  Proses     ; Lompat jika Zerro flag 0
```

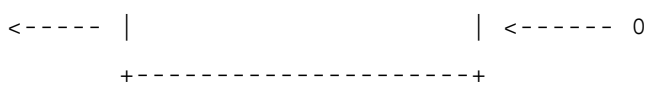
Pada perintah diatas komputer akan menuju ke label Proses bila ada satu bit atau lebih dari AX yang sama dengan 0Fh. Bila diikuti dengan perintah **JC Proses**, maka komputer akan menuju ke label proses bila keempat byte rendah pada AL semuanya 1(?F).

10.6. SHL (Shift Left)

Operator SHL akan menggeser operand1 ke kiri sebanyak operand2 secara per bit. Kemudian bit kosong yang telah tergeser di sebelah kanannya akan diberi nilai nol. Operator SHL digunakan dengan syntax :

SHL Operand1,Operand2

Supaya lebih jelas bisa anda lihat pada gambar 10.5. Operand2 harus digunakan register CL bila pergeseran yang dilakukan lebih dari satu kali.



Gambar 10.5. Operasi SHL

Instruksi :
```
MOV AX,3Fh
MOV CL,3
SHL AX,CL    ; Geser 3 bit ke kiri
```
Akan menghasilkan nilai F8h pada register AX. Operasi detilnya dapat dilihat di bawah ini.

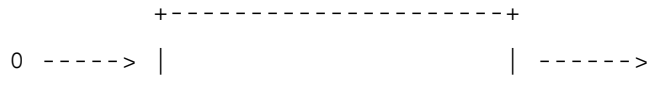
```
3Fh     :  0011 1111
SHL 1   :  0111 1110 (=7Eh)
SHL 2   :  1111 1100 (=FCh)
SHL 3   :  1111 1000 (=F8h)
```

10.7. SHR (Shift Right)

Operator SHR akan menggeser operand1 ke kanan sebanyak operand2 secara per bit dan menambahkan nilai nol pada bit yang tergeser seperti halnya pada operator SHL. Operator SHR digunakan dengan syntax :

SHR Operand1,Operand2

Supaya lebih jelas anda bisa lihat pada gambar 10.6. Operand2 harus digunakan register CL bila pergeseran yang dilakukan lebih dari satu kali.



Gambar 10.6. Operasi SHR

Instruksi : **MOV AX,3Fh**

MOV CL,3

SHR AX,CL ; Geser 3 bit ke kanan

Akan menghasilkan nilai 07h pada register AX. Operasi detilnya dapat dilihat di bawah ini.

3Fh : 0011 1111

SHR 1 : 0001 1111 (=1Fh)

SHR 2 : 0000 1111 (=0Fh)

SHR 3 : 0000 0111 (=07h)

BAB XI

ADDRESSING MODES

11.1. PENDAHULUAN

Pada bab-bab sebelumnya kita telah lihat, bagaimana perintah "MOV" mengcopykan suatu nilai kepada suatu register atau variabel. Kita bisa mengcopykan nilai pada suatu register, variabel ataupun lokasi memory dengan berbagai cara. Secara umum banyaknya cara yang dapat digunakan dapat dibagi menjadi 7, seperti pada gambar 11.1.

ADDRESSING MODE	FORMAT	SEGMENT REGISTER
1. Immediate	Data	Tidak Ada
2. Register	Register	Tidak Ada
3. Direct	Displacement	DS
	Label	DS
4. Register Indirect	[BX]	DS
	[BP]	SS
	[SI]	DS
	[DI]	DS
5. Base Relative	[BX]+Displacement	DS
	[BP]+Displacement	SS
6. Direct Indexed	[DI]+Displacement	DS
	[SI]+Displacement	DS
7. Base Indexed	[BX][SI]+Displacement	DS
	[BX][DI]+Displacement	DS
	[BP][SI]+Displacement	SS
	[BP][DI]+Displacement	SS

Gambar 11.1. Addressing Modes

Perlu anda perhatikan bahwa ada juga pengcopyan data yang terlarang, yaitu :

1. Pengcopyan data antar segment register, seperti:

```
MOV  DS,ES
```

Untuk memecahkan hal ini, anda bisa menggunakan register general purpose

sebagai perantara, seperti:

```
MOV  AX,ES
MOV  DS,AX
```

Selain dengan cara diatas, anda bisa juga menggunakan stack sebagai perantara, seperti:

```
PUSH  ES
POP   DS
```

2. Pemberian nilai untuk segment register(DS, ES, CS, SS) secara langsung, seperti:

```
MOV   ES,0B800h
```

Untuk memecahkan hal ini, anda bisa menggunakan register general purpose sebagai perantara, seperti:

```
MOV   AX,0B800h
MOV   ES,AX
```

3. Pengcopyan data langsung antar memory, seperti:

```
MOV  MemB,MemA
```

Untuk memecahkan hal ini, anda bisa menggunakan register general purpose sebagai perantara, seperti:

```
MOV  AX,MemA
MOV  MemB,AX
```

4. Pengcopyan data antar register yang berbeda tipenya(8 bit dengan 16 bit) tanpa menggunakan pointer, seperti:

```
MOV  AL,BX
```

Pelajarilah:

bagian ini dengan baik, karena addressing modes merupakan dasar bagi programmer bahasa assembly yang harus dikuasai.

11.2. IMMEDIATE ADDRESSING

Pengcopyan data yang tercepat ialah yang dinamakan dengan Immediate Addressing dan Register Addressing. Immediate Addressing adalah suatu pengcopyan data untuk suatu register 8,16 atau 32(80386) bit langsung dari suatu angka. Contohnya :

```
MOV AX,50h
MOV EAX,11223344h   <80386>
```

Immediate Addressing dapat juga mendapatkan nilainya melalui suatu constanta yang telah didefinisikan dengan perintah EQU, seperti :

```
A EQU 67h
  ;
  ;
MOV AX,A
```

Perintah diatas akan mengcopykan nilai 67h untuk register AX.

11.3. REGISTER ADDRESSING

Register Addressing adalah suatu proses pengcopyan data antar register. Pengcopyan antar register ini harus digunakan register yang berukuran sama, seperti AL dan BH, CX dan AX. Contah perintahnya:

```
MOV AX,CX
```

Register Addressing dapat juga dapat juga hanya terdiri atas sebuah register seperti pada perintah **INC CX**.

```
;/========================\;
;   PROGRAM : ADDR1.ASM    ;
;   AUTHOR  : S'to         ;
;   FUNGSI  : PERKALIAN    ;
;             DENGAN 80386 ;
;\========================/;

          .MODEL SMALL
          .386              ; Untuk prosesor 80386
          .CODE
          ORG 100h
Proses :
          MOV EAX,12345678h ; Immediate Addressing
          MOV EDX,33112244h ; Immediate Addressing
          MOV EBX,EDX       ; Register  Addressing
          MUL EBX           ; Register  Addressing

END       Proses
```

Program 11.1. Perkalian pada 80386

Aturan perkalian pada pada 80386 ini sama dengan perkalian yang telah kita bicarakan didepan. Pada prosesor 80386 digunakan register-register 32 bit, seperti EAX ,EBX dan EDX pada contoh program 11.1. Untuk menggunakan kelebihan pada komputer 80386 ini kita harus menambahkan directive **.386**.

11.4. DIRECT ADDRESSING

Secara umum Direct Addressing ialah suatu pengcopyan data pada suatu register dan simbol.

Contoh:

```
TData : JMP Proses
        A DB 12h
        B DB 59h

Proses :  MOV AL,A   ; Direct Addressing
          MOV AH,B   ; Direct Addressing
```

Perintah diatas akan mengcopykan data variabel A dan B pada register AL dan AH.

11.5. REGISTER INDIRECT ADDRESSING

Register Indirect Addressing biasanya digunakan untuk mengakses suatu data yang banyak dengan mengambil alamat efektive dari data tersebut. Register-register yang bisa digunakan pada addressing ini adalah BX,BP,SI dan DI. Tetapi bila anda memrogram pada prosesor 80386(Dengan menambahkan directive .386) maka semua register general purpose bisa dipakai.

Untuk mendapatkan alamat efektive dari suatu data bisa digunakan perintah LEA(Load Effective Addres) dengan syntax :

LEA Reg,Data

Untuk mengakses data yang ditunjukkan oleh register **Reg**, setelah didapatkannya alamat efektive harus digunakan tanda kurung siku ('**[]**').

Jika pada perintah pengaksesannya tidak disebutkan segmennya, maka yang digunakan adalah segment default. Seperti bila digunakan register BX sebagai penunjuk offset, maka segment DS yang digunakan. Sebalikkan bila digunakan register BP sebagai penunjuk offset, maka segment SS yang digunakan.

```
;/============================\;
;   PROGRAM : RID.ASM          ;
;   AUTHOR  : S'to             ;
;   FUNGSI  : MENGAKSES DATA   ;
;             MELALUI ALAMAT   ;
;             EFEKTIVE         ;
;\============================/;

        .MODEL SMALL
        .CODE
        ORG 100h

TData : JMP Proses
         Kal DB 'ABCDEF'

Proses:
         LEA  BX,Kal     ; Ambil Offset Kal
         MOV  CX,2
 Ulang:
         MOV  DL,[BX] ; kode ASCII yang ingin dicetak
         MOV  AH,02h   ; Nilai servis ntuk mencetak karakter
```

```
          INT  21h      ; Laksanakan !!
          ADD  BX,2     ; BX:=BX+2
          LOOP Ulang    ; Lompat ke Ulang

          INT 20h
END       TData
```

Program 11.2. Proses Register Indirect Addressing

Bila program 11.2. dijalankan maka dilayar anda akan tercetak :

AC

Pertama-tama kita mendefinisikan data untuk variabel 'Kal', dimana data ini nantinya akan disimpan pada memory, seperti berikut :

```
              +-----+-----+-----+-----+-----+-----+
              |  A  |  B  |  C  |  D  |  E  |  F  |
              +-----+-----+-----+-----+-----+-----+
Alamat Offset:   103   104   105   106   107   108
```

Pada perintah **LEA BX,Kal**, maka register BX akan menunjuk pada alamat efektive dari variabel Kal, sebagai berikut :

```
                BX=103
                _
              +-----+-----+-----+-----+-----+-----+
              |  A  |  B  |  C  |  D  |  E  |  F  |
              +-----+-----+-----+-----+-----+-----+
Alamat Offset:   103   104   105   106   107   108
```

Pada perintah **MOV CX,2**, kita memberikan nilai 2 kepada register CX untuk digunakan sebagai counter pada saat LOOP. Kini perhatikanlah bahwa kita mengambil nilai yang ditunjukkan oleh register BX yaitu 'A' dengan perintah **MOV DL,[BX]**. Tanda kurung siku menyatakan bahwa kita bukannya mengambil nilai BX tetapi nilai yang ditunjukkan oleh register BX. Setelah itu kita mencetak karakter tersebut dengan interupsi 21h servis 02 dimana kode ASCII dari karakter yang ingin dicetak telah kita masukkan pada register DL.

Pada perintah **ADD BX,2** kita menambahka nilai 2 pada BX sehingga kini BX akan berjalan sebanyak 2 byte dan menunjuk pada data 'C' sebagai berikut :

```
                      BX=105
                      _
              +-----+-----+-----+-----+-----+-----+
```

```
              | A | B | C | D | E | F |
              +---+---+---+---+---+---+
Alamat Offset:  103 104 105 106 107 108
```

Kini BX telah menunjuk pada alamat tempat data 'C' berada, sehingga pada pencetakan karakter selanjutnya, yang tercetak adalah karakter 'C'.

Ingatlah:

satu karakter menggunakan satu byte memory.

11.6. BASE RELATIVE ADDRESSING

Jenis addressing ini biasanya digunakan untuk mengakses suatu tabel dengan mengambil alamat efektivenya. Alamat efektive dari tabel tersebut nantinya digunakan sebagai patokan untuk mengakses data yang lain pada tabel tersebut. Register yang digunakan sebagai penunjuk alamat efektive ini adalah register BX,BP,SI dan DI.

```
;/==============================\;
;   PROGRAM : BRA0.ASM           ;
;   AUTHOR  : S'to               ;
;   FUNGSI  : MENGAKSES DATA     ;
;             DENGAN BASE        ;
;             RELATIVE ADDRESSING;
;                                ;
;\==============================/;

     .MODEL SMALL
     .CODE
      ORG 100h

TData : JMP Proses
        Tabel DW 11h,50h,0Ah,14h,5Ah
Proses:
        LEA  BX,Tabel
        MOV  AX,Tabel

        ADD  AX,[BX]+2
        ADD  AX,[BX]+4
        ADD  AX,[BX+6]
        ADD  AX,[BX+8]

        INT 20h
END     TData
```

Program 11.3. Proses Base Relative Addressing

Pertama-tama kita mendefinisikan suatu tabel yang berisi data 11h,50h,0Ah,14h dan 5Ah. Data ini akan disimpan dalam memory sebagai berikut :

```
              +------+------+------+------+------+
              | 0011 | 0050 | 000A | 0014 | 005A |
              +------+------+------+------+------+
```

```
Alamat Offset:   103     105    107    109    10A
```

Setelah itu kita mengambil alamat efektifnya dengan menggunakan register BX dengan perintah **LEA BX,Tabel** sehingga BX akan menunjuk pada alamat data yang pertama.

```
                BX=103

                 _
               +------+------+------+------+------+
               | 0011 | 0050 | 000A | 0014 | 005A |
               +------+------+------+------+------+
Alamat Offset:   103     105    107    109    10A
```

Dengan perintah **MOV AX,Tabel** maka AX akan berisi nilai pada word pertama variabel 'Tabel', yaitu 11. Dengan BX yang telah menunjuk pada data pertama(11) maka kita bisa menggunakannya sebagai patokan untuk mengakses data yang lain.

```
                BX     BX+2   BX+4   BX+6  BX+8

                 _      _      _      _      _
               +------+------+------+------+------+
               | 0011 | 0050 | 000A | 0014 | 005A |
               +------+------+------+------+------+
Alamat Offset:   103     105    107    109    10A
```

Perlu anda perhatikan bahwa kita mengakses data yang lain terhadap BX tanpa merubah posisi dari penunjuk BX, jadi BX tetap menunjuk pada offset Tabel. Kita menambah BX dengan 2 karena data terdefinisi sebagai word(2 byte). Pada contoh program 11.3. dapat anda lihat bahwa menambah BX didalam dan diluar kurung siku adalah sama.

```
;/==============================\;
;   PROGRAM : BRA1.ASM           ;
;   AUTHOR  : S'to               ;
;   FUNGSI  : MENCETAK KALIMAT   ;
;             DENGAN BASE        ;
;             RELATIVE ADDRESSING;
;                                ;
;\==============================/;

        .MODEL SMALL
        .CODE
        ORG 100h

TData : JMP Proses
        Kalimat DB 'NYAMUK GORENG'   ; 13 karakter
Proses:
        XOR  BX,BX           ; BX=0 Untuk penunjuk Offset
        MOV  CX,13           ; Counter LOOP
Ulang :
        MOV  DL,Kalimat[BX]  ; Ambil karakter yang ke BX
        MOV  AH,02           ; Servis untuk cetak karakter
        INT  21h             ; Cetak Karakter
        INC  BX              ; BX:=BX+1
        LOOP Ulang           ; Lompat ke Ulang sampai CX=0
```

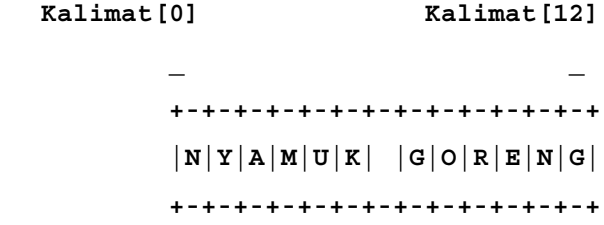
```
        INT 20h              ; Selesai, kembali ke DOS !!
END     TData
```

Program 11.4. Mencetak kalimat dengan Base Relative Addressing

Bila program 11.4. dijalankan maka dilayar akan tampak tulisan :

NYAMUK GORENG

Pada program 11.4. ini register BX dijadikan sebagai pencatat offset dari **"kalimat"**. Dengan nilai BX sama dengan nol(0), akan menunjuk pada karakter pertama dari Kalimat(ingat! XOR dengan bilangan yang sama pasti menghasilkan 0). Setelah itu kita memberikan nilai 13 kepada CX sebagai penghitung banyaknya LOOP yang akan terjadi.



Pada perintah **MOV DL,Kalimat[BX]**, register BX digunakan untuk menunjukkan offset dari kalimat. Dengan demikian saat pertama kali yang dimasukkan pada register DL untuk dicetak adalah karakter 'N' kemudian BX ditambah satu sehingga BX menunjuk pada karakter 'Y'. Demikian seterusnya sampai seluruh kalimat tersebut tercetak.

11.7. DIRECT INDEXED ADDRESSING

Direct Indexed Addressing mengambil alamat efektive dari suatu data dan mengakses data dengan menggunakan register DI atau SI. Sebagai contohnya akan kita lihat tanggal dikeluarkannya ROM BIOS komputer.

Tanggal dikeluarkannya ROM BIOS pada setiap komputer terdapat pada alamat mulai F000h:FFF5h sampai F000h:FFFCh. Pada daerah ini akan terdapat 8 byte (8 huruf) yang berisi tanggal dikeluarkannya ROM BIOS. Tanggal yang tercantum menggunakan format penulisan tanggal Amerika, misalnya 04/03/73 artinya 14 Maret 1973.

```
;/=========================\;
;   PROGRAM : VRBIOS.ASM    ;
;   AUTHOR  : S'to          ;
;   FUNGSI  : MELIHAT VERSI ;
;             BIOS KOMPUTER ;
;                           ;
```

```
;\=========================/;

         .MODEL SMALL
         .CODE
         ORG 100h
Proses :
         MOV AX,0F000h       ; Masukkan nilai F000 pada AX
         MOV ES,AX           ; Copykan nilai AX ke ES
         MOV BX,0FFF5h       ; Penunjuk Offset
         XOR SI,SI           ; Jadikan SI=0
         MOV CX,8            ; Counter untuk LOOP
  Ulang:
         MOV  DL,ES:[BX][SI] ; Ambil isi alamat ES:BX+SI
         MOV  AH,02h         ; Nilai servis mencetak karakter
         INT  21h            ; Cetak !!
         INC  SI             ; SI:=SI+1
         LOOP Ulang          ; Lompat ke Ulang sampai CX=0

         INT  20h            ; Selesai ! kembali ke DOS
                               END      Proses
```

Program 11.5. Melihat Versi ROM BIOS

Bila program 11.5. dijalankan, maka akan ditampilkan :

18/08/94 <pada komputer anda mungkin lain>

Kita tidak bisa langsung mengisikan sebuah nilai kepada segment register, oleh karena itu digunakan register AX sebagai perantara sebagai berikut:

```
MOV AX,0F000h
MOV ES,AX
```

Setelah itu register BX yang kita gunakan sebagai penunjuk offset, diisi dengan nilai FFF5, sedangkan SI yang nantinya digunakan sebagai displacement(perpindahan) kita jadikan nol. Register CX yang digunakan sebagai counter diisi dengan 8, sesuai dengan jumlah LOOP yang dinginkan:

```
MOV BX,0FFF5h
XOR SI,SI
MOV CX,8
```

Kini kita bisa mengambil data pada alamat F000:FFF5, dengan segmnent register ES dan offset pada register BX+SI. Segment register ES ini harus dituliskan, karena bila tidak dituliskan maka segment yang digunakan adalah segment default atau segment register DS. Register SI digunakan sebagai perpindahan terhadap register BX, **[BX][SI]** artinya register BX+SI.

```
MOV  DL,ES:[BX][SI]
MOV  AH,02h
INT  21h
INC  SI
LOOP Ulang
```

Proses diulangi sampai 8 karakter tanggal dikeluarkannya ROM BIOS tercetak semua.

11.8. BASED INDEXED ADDRESSING

Jenis addressing ini biasanya digunakan untuk mengakses suatu record

atau suatu array 2 dimensi.

```
;/==============================\;
;   PROGRAM : BIA.ASM             ;
;   AUTHOR  : S'to                ;
;   FUNGSI  : MENGAKSES ARRAY     ;
;             DENGAN BASE         ;
;             INDEXED ADDRESSING  ;
;\==============================/;

         .MODEL SMALL
         .CODE
         ORG 100h

TData : JMP  Proses
        Mahasiswa  STRUC
                   Nim      DW  0         ; 2 byte
                   Tinggi   DB  0         ; 1 byte
                   Nilai    DB  0,0,0,0   ; 4 byte
        Mahasiswa  ENDS
        Absen      Mahasiswa 10 DUP (<>)

Proses:
        LEA  BX,Absen      ; BX Menunjuk Offset Absen
        ADD  BX,21         ; BX Menunjuk pada Record ke 4
        XOR  SI,SI         ; SI=0

        MOV  [BX][SI].Nim    ,0099h ; NIM, record ke 4
        MOV  [BX][SI].Tinggi ,10h   ; Tinggi, record ke 4
        MOV  [BX][SI+1].Nilai,78h   ; Nilai pertama
        MOV  [BX][SI+2].Nilai,99h   ; Nilai kedua
        MOV  [BX][SI+3].Nilai,50h   ; Nilai keempat
        MOV  [BX][SI+4].Nilai,83h   ; Nilai kelima

        INT  20h                    ; Selesai !!
END     TData
```

Program 11.6. Teknik Mengakses Record

Pada program 11.6. akan kita lihat bagaimana based indexed addressding memudahkan kita dalam mengakses suatu array record.

```
Mahasiswa  STRUC
            Nim      DW  ?
            Tinggi   DB  ?
            Nilai    DB  ?,?,?,?
Mahasiswa  ENDS
Absen      Mahasiswa 10 DUP (<>)
```

Perintah "STRUC" digunakan untuk mendefinisikan suatu record dan diakhiri dengan "ENDS". Field-field yang kita definisikan untuk record mahasiswa ini adalah 2 byte untuk NIM, 1 byte untuk Tinggi, 4 byte untuk Nilai. Jadi besar satu record adalah 7 byte. Pada baris selanjutnya kita mendefinisikan 10 buah record mahasiswa dengan perintah DUP. Tanda cryptic

"**(<>)**" digunakan untuk menginialisasi nilai pada array menjadi nol.

```
ADD  BX,21
XOR  SI,SI
```

Pada contoh program ini kita akan memasukan data pada record ke 4, dan karena 1 record menggunakan 7 byte, maka BX kita tambah dengan 21 supaya BX menunjuk pada record ke 4. Register SI yang nantinya kita gunakan sebagai perpindahan dijadikan 0.

```
MOV  [BX][SI].Nim    ,0099h
MOV  [BX][SI].Tinggi ,10h
```

Dengan BX yang telah menunjuk pada record ke 4, maka kita bisa langsung memasukkan nilai untuk NIM dan Tinggi pada record ke 4.

```
MOV  [BX][SI].Nilai  ,78h
MOV  [BX][SI+1].Nilai,99h
MOV  [BX][SI+2].Nilai,50h
MOV  [BX][SI+3].Nilai,83h
```

Kini perhatikanlah bahwa dalam memasukkan angka untuk variabel "**nilai**" yang mempunyai 4 byte bisa kita gunakan register SI sebagai perpindahan. "MOV [BX][SI]" akan menunjuk pada byte pertama untuk variabel nilai sedangkan "[BX][SI+1]" akan menunjuk pada byte kedua untuk variabel nilai, demikianlah seterusnya. Mudah Bukan ?.

BAB XII

MENCETAK KALIMAT

12.1. MENCETAK KALIMAT DENGAN FUNGSI DOS

Untuk mencetak kalimat, bisa digunakan interupsi 21 fungsi 9 dengan aturan:

INPUT

AH = 9

DS:DX = Alamat String tersebut

CATATAN = Karakter '**$**' dijadikan tanda akhir tulisan

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~;
;    Program: kal0.asm                 ;
;    Oleh   : S'to                     ;
;    Fungsi : Mencetak String          ;
;             dengan Int 21 servis 9   ;
;======================================;

         .MODEL SMALL
         .CODE
         ORG 100h

Tdata :  JMP Proses
         Kal0  DB 'PROSES PENCETAKAN STRING ',13,10,'$'
         Kal1  DB 'DIBELAKANG TANDA $ TIDAK BISA DICETAK '
Proses:
         MOV  AH,09h         ; Servis ke 9
         MOV  DX,OFFSET Kal0 ; Ambil Alamat Offset Kal0
         INT  21h            ; Cetak perkarakter sampai tanda $

         LEA  DX,Kal0        ; Ambil Alamat Offset Kal0
         INT  21h            ; Cetak perkarakter sampai tanda $

         LEA  DX,Kal0+7      ; Ambil Alamat Offset KAl0+7
         INT  21h            ; Cetak perkarakter sampai tanda $

         LEA  DX,KAL1        ; Ambil Offset kal1
         INT  21h            ; Cetak perkarakter sampai ketemu $

         INT  20h            ; Selesai, kembali ke DOS
END      Tdata
```

Program 12.1. Mencetak kalimat dengan fungsi DOS

Pada saat program 12.1. anda jalankan, maka dilayar akan ditampilkan:

```
PROSES PENCETAKAN STRING
DIBELAKANG TANDA
```

Pada saat pendefinisian untuk variabel "KAL0" kita menambahkan tanda 13 dan 10. Kedua tanda ini merupakan karakter kontrol untuk pindah baris(tanda 10) dan menuju kolom 0(tanda 13). Pada akhir dari setiap kalimat yang ingin dicetak harus kita tambahkan dengan karakter "$". Karakter ini akan dipakai

sebagai tanda akhir dari kalimat.

Karena karakter "$" dijadikan sebagai tanda akhir dari kalimat yang ingin dicetak, maka pada proses pencetakan karakter yang kedua hanya kalimat "**DIBELAKANG TANDA**" yang tercetak. Sisa kalimatnya, yaitu "**TIDAK BISA DICETAK**" tidak tercetak keluar, karena terletak dibelakang tanda "$".

Dengan demikian, bila kita ingin mencetak kalimat yang mengandung tanda "$", harus digunakan fungsi yang lain, misalnya mencetak kalimat dengan perkarakter melalui interupsi 21 fungsi 2.

12.2. KARAKTER KONTROL

Pada program 12.1. kita telah menggunakan 2 buah karakter kontrol, yaitu 10(LF) dan 13(CR). Karakter kontrol yang tersedia untuk operasi pada video yang sering digunakan terdapat 5, yaitu 07, 08, 09, 10 dan 13(Gambar 12.1).

CODE	NAMA	FUNGSI
07	Bel	Memberikan suara BEEP
08	Backspace(BS)	Memindahkan kursor 1 kolom ke belakang
09	Horisontal Tab	Memindahkan kursor 8 kolom ke kanan
10	Line Feed(LF)	Memindahkan kursor 1 baris ke bawah
13	Carriage Return(CR)	Memindahkan kursor menuju awal baris

Gambar 12.1. Karakter Kontrol Yang Sering Digunakan

Selain dari karakter kontrol pada gambar 12.1, masih terdapat karakter-karakter kontrol lain, yang sebagian besar digunakan untuk keperluan komunikasi komputer dengan periferalnya. Karakter kontrol yang tersedia pada ASCII secara lengkap bisa anda lihat pada gambar 12.2.

CODE	NAMA	CODE	NAMA
00	Nul	16	Data Link Escape
01	Start Of Heading	17	Device Control
02	Start Of Text	18	Negative Acknowledge
03	End Of Text	19	Synchronous Idle
04	End Of Transmission	20	End Of Transmission Block
05	Enquiry	21	Cancel

```
 06     Acknowledge           | 22     End Of Medium
 07     Bel                   | 23     Substitute
 08     Backspace             | 24     Escape
 09     Horisontal Tabulation | 25     File Separator
 10     Line Feed             | 26     Group Separator
 11     Vertical Tabulation   | 27     Record Separator
 12     Form Feed             | 28     Unit Separator
 13     Carriage Return       | 29     Space
 14     Shift Out             | 30     Delete
 15     Shift In              |
-----------------------------+---------------------------------
```

Gambar 12.2. Karakter Kontrol Pada ASCII

12.3. MENCETAK KALIMAT DENGAN ATRIBUTNYA

Pada bagian sebelumnya kita mencetak kalimat dengan fungsi DOS yang mencetak kalimat tanpa atribut. Untuk mencetak kalimat dengan atributnya bisa digunakan fungsi dari BIOS, melalui interupsi 10h. Adapun yang harus anda persiapkan adalah: register AX diisi dengan 1300h, BL diisi dengan atribut yang ingin ditampilkan, BH diisi dengan halaman tampilan, DL diisi dengan posisi X tempat kalimat tersebut akan tercetak sedangkan DH diisi dengan posisi Y-nya. Karena fungsi ini tidak mengenal batas tulisan "$" seperti interupsi 21h servis 9, maka kita harus mengisikan CX dengan banyaknya karakter dalam kalimat. Register ES:BP digunakan untuk mencatat alamat dari kalimat yang ingin dicetak.

```
;/=============================================\;
;          Program : ATTR-KLM.ASM               ;
;          Author  : S'to                       ;
;          Fungsi  : Mencetak kalimat disertai  ;
;                    atributnya                 ;
;---------------------------------------------;
;                    INT  10h                   ;
;---------------------------------------------;
;    Input :                                    ;
;            AX = 1300h                         ;
;            BL = Atribut                       ;
;            BH = Halaman tampilan              ;
;            DL = Posisi X                      ;
;            DH = Posisi Y                      ;
;            CX = Panjang kalimat<dalam karakter>;
;         ES:BP = Alamat awal string            ;
;                                               ;
;\=============================================/;

         .MODEL SMALL
         .CODE
         ORG 100h

TData : JMP   Proses
```

```
        Kal0  DB ' Menulis kalimat dengan Atributnya '
Proses:
        MOV   AX,1300h       ; Servis 13h subfungsi 00
        MOV   BL,10010101b   ; Atribut tulisan
        MOV   BH,00          ; Halaman tampilan 0
        MOV   DL,20          ; Posisi X
        MOV   DH,12          ; Posisi Y
        MOV   CX,35          ; Banyaknya karakter dalam string
        LEA   BP,Kal0        ; ES:BP alamat string
        INT   10h            ; Cetak kalimat !

        INT   20h            ; Selesai, kembali ke DOS
END     TData
```

Program 12.2. Mencetak Kalimat Dengan Atributnya

Bila program 12.2. dijalankan, maka layar pada posisi kolom ke 20 dan baris ke 12 akan terdapat tulisan:

Menulis kalimat dengan Atributnya

Tulisan ditampilkan dengan warna tulisan putih dan warna dasar jingga. Mengenai halaman layar akan dibahas pada bagian yang lain, sedangkan mengenai atribut akan segera kita bahas.

12.4. PENGATURAN ATRIBUT

Atribut atau warna menggunakan **1** byte memory, yang akan menandakan warna tulisan dan warna dasar dari karakter yang akan tercetak. Byte atribut ini digunakan dengan masing-masing bitnya, dimana setiap bit mencatat warnanya masing-masing. Adapun spesifikasinya adalah:

```
           Warna Dasar     Warna Tulisan
           +----+----+     +----+----+
Bit-ke  7   6   5   4   3   2   1   0

        _   _   _   _   _   _   _   _
        BL  R   G   B   I   R   G   B
```

Catatan: BL = Blink atau berkedip
R = Merah
G = Hijau
B = Biru
I = Intensitas warna

Untuk menghidupkan warna yang diinginkan anda tinggal menjadikan bit tersebut menjadi satu. Sebagai contohnya bila anda menginginkan warna tulisan Biru dengan warna dasar Hijau, maka anda tinggal menghidupkan bit ke 0 dan 5 atau dengan angka 00100001b(21h). Untuk menjadikannya berintensitas tinggi dan berkedip anda juga tinggal menjadikan bit ke 3 dan 7 menjadi satu(10101001b).

Bila anda menghidupkan bit ke 0,1 dan 2 menjadi satu dan mematikan bit-bit lainnya maka anda akan mendapatkan campuran dari ketiga warna tersebut(Putih) untuk warna tulisan dan warna hitam untuk warna dasar. Inilah

warna normal yang biasa digunakan, yaitu warna dengan atribut 7.

BAB XIII
BANDINGKAN DAN LOMPAT

13.1. LOMPAT TANPA SYARAT

Perintah JMP(Jump), sudah pernah kita gunakan, dimana perintah ini digunakan untuk melompati daerah data program. Perintah JMP digunakan dengan syntax:

JMP Tujuan

Perintah JMP ini dikategorikan sebagai Unconditional Jump, karena perintah ini tidak menyeleksi keadaan apapun untuk melakukan suatu lompatan. Setiap ditemui perintah ini maka lompatan pasti dilakukan.

Selain dari perintah jump tanpa syarat, masih banyak perintah Jump yang menyeleksi suatu keadaan tertentu sebelum dilakukan lompatan. Perintah jump dengan penyeleksian kondisi terlebih dahulu biasanya diikuti dengan perintah untuk melihat kondisi, seperti membandingkan dengan perintah "CMP"(Compare).

13.2. MEMBANDINGKAN DENGAN CMP

Perintah CMP(Compare) digunakan untuk membandingkan 2 buah operand, dengan syntax:

CMP Operand1,Operand2

CMP akan membandingkan operand1 dengan operand2 dengan cara mengurangkan operand1 dengan operand2. CMP tidak mempengaruhi nilai Operand1 dan Operand2, perintah CMP hanya akan mempengaruhi flags register sebagai hasil perbandingan. Adapun flag-flag yang terpengaruh oleh perintah CMP ini adalah:

- OF akan 1, jika operand1 lebih kecil dari operand2 pada operasi bilangan bertanda.
- SF akan 1, bila operand1 lebih kecil dari operand2, pada operasi bilangan bertanda.
- ZF akan 1, jika operand1 nilainya sama dengan operand2.
- CF akan 1, jika operand1 lebih kecil dari operand2 pada operasi bilangan tidak bertanda.

Perlu anda ingat bahwa CMP tidak dapat membandingkan antar 2 lokasi memory.

13.3. LOMPAT YANG MENGIKUTI CMP

Perintah CMP yang hanya mempengaruhi flag register, biasanya diikuti dengan perintah lompat yang melihat keadaan pada flags register ini. Jenis perintah lompat yang biasanya mengikuti perintah CMP, terdapat 12 buah seperti pada gambar 13.1.

Perintah Lompat	Kondisi
JA <Jump If Above>	**Lompat, jika Operand1 > Operand2 untuk bilangan tidak bertanda**
JG <Jump If Greater>	**Lompat, jika Operand1 > Operand2**

```
                                  | untuk bilangan bertanda
      JE   <Jump If Equal>        | Lompat, jika Operand1 = Operand2
      JNE  <Jump If Not Equal>    | Lompat, jika Operand1 tidak sama
                                  | dengan Operand2
      JB   <Jump If Below>        | Lompat, jika Operand1 < Operand2
                                  | untuk bilangan tidak bertanda
      JL   <Jump If Less>         | Lompat, jika Operand1 < Operand2
                                  | untuk bilangan bertanda
      JBE  <Jump If Below or Equal>| Lompat, jika operand1 <= Operand2
                                  | untuk bilangan tidak bertanda
      JLE  <Jump If Less or Equal> | Lompat, jika Operand1 <= Operand2
                                  | untuk bilangan bertanda
      JAE  <Jump If Above or Equal>| Lompat, jika Operand1 >= Operand2
                                  | untuk bilangan tidak bertanda
      JGE  <Jump If Greater or    | Lompat, jika Operand1 >= Operand2
            Equal>                | untuk bilangan bertanda
      ----------------------------+---------------------------------
```

Gambar 13.1. Perintah Jump yang mengikuti CMP

Pada tabel 13.1. dapat anda lihat bahwa terdapat dua operasi yang berbeda, yaitu operasi bilangan bertanda dan tidak bertanda. Bilangan bertanda adalah bilangan yang akan membedakan bilangan negatif dan positif(Mis. 37 dan -37). Sedangkan bilangan tidak bertanda adalah bilangan yang tidak akan membedakan positif dan negatif, jadi angka -1 untuk operasi bilangan bertanda akan dianggap FFh pada bilangan tidak bertanda. Lebih jauh mengenai bilangan bertanda dan tidak ini bisa anda lihat pada bab1.

```
;/=======================================\;
;     Program : CMPJ.ASM                  ;
;     Author  : S'to                      ;
;     Fungsi  : Mendemokan perintah lompat  ;
;               yang mengikuti perintah CMP ;
;                                         ;
;\=======================================/;

        .MODEL SMALL
        .CODE
        ORG 100h

TData:  JMP Proses
         BilA  DB  67
         BilB  DB  66
         Kal0  DB  'Bilangan A lebih kecil dari bilangan B $'
         Kal1  DB  'Bilangan A sama dengan bilangan B $'
         Kal2  DB  'Bilangan A lebih besar dari bilangan B $'
Proses:
        MOV   AL,BilA     ; Masukkan bilangan A pada AL
        CMP   AL,BilB     ; Bandingkan AL(BilA) dengan Bilangan B
        JB    AKecil      ; Jika BilA < BilB, lompat ke AKecil
        JE    Sama        ; Jika BilA = BilB, lompat ke Sama
        JA    ABesar      ; Jika BilA > BilB, lompat ke ABesar
Akecil:
```

```
        LEA   DX,Kal0     ; Ambil offset Kal0
        JMP   Cetak       ; Lompat ke cetak
  Sama:
        LEA   DX,Kal1     ; Ambil offset Kal1
        JMP   Cetak       ; Lompat ke cetak
ABesar:
        LEA   DX,Kal2     ; Ambil offset Kal2
 Cetak:
        MOV   AH,09       ; Servis untuk mencetak kalimat
        INT 21h           ; Cetak kalimat !!

  EXIT: INT 20h           ; Kembali ke DOS.
END     TData
```

Program 13.1. Menggunakan Perintah Lompat Bersyarat

Bila program 13.1. dijalankan, maka akan tampak pada layar:

Bilangan A lebih besar dari bilangan B

Anda bisa mengganti nilai pada variabel BilA dan BilB untuk melihat hasil yang akan ditampilkan pada layar.

13.4. LOMPAT BERSYARAT

Pada gambar 13.1. anda telah melihat sebagian dari perintah lompat bersyarat. Kini akan kita lihat lompat bersyarat lainnya yang tersedia, seperti pada gambar 13.2. Tidak seperti lompat tanpa syarat, Lompat bersyarat hanya dapat melompat menuju label yang berjarak -128 sampai +127 byte dari tempat lompatan.

Perintah Lompat	Kondisi
JA <Jump If Above>	Lompat, jika Operand1 > Operand2 untuk bilangan tidak bertanda
JG <Jump If Greater>	Lompat, jika Operand1 > Operand2 untuk bilangan bertanda
JE <Jump If Equal>	Lompat, jika Operand1 = Operand2
JNE <Jump If Not Equal>	Lompat, jika Operand1 tidak sama dengan Operand2
JB <Jump If Below>	Lompat, jika Operand1 < Operand2 untuk bilangan tidak bertanda
JL <Jump If Less>	Lompat, jika Operand1 < Operand2 untuk bilangan bertanda
JBE <Jump If Below or Equal>	Lompat, jika operand1 <= Operand2 untuk bilangan tidak bertanda
JLE <Jump If Less or Equal>	Lompat, jika Operand1 <= Operand2 untuk bilangan bertanda
JAE <Jump If Above or Equal>	Lompat, jika Operand1 >= Operand2 untuk bilangan tidak bertanda
JGE <Jump If Greater or Equal>	Lompat, jika Operand1 >= Operand2 untuk bilangan bertanda
JC <Jump Carry>	Lompat, jika Carry flag=1

```
JCXZ <Jump If CX is Zero>     | Lompat, jika CX=0
JNA  <Jump If Not Above>      | Lompat, jika Operand1 < Operand2
                              | dengan CF=1 atau ZF=1
JNAE <Jump If Not Above nor   | Lompat, jika Operand1 < Operand2
      Equal>                  | dengan CX=1
JNB  <Jump If Not Below>      | Lompat, jika Operand1 > Operand2
                              | dengan CF=0
JNBE <Jump If Not Below nor   | Lompat, jika Operand1 > Operand2
      Equal>                  | dengan CF=0 dan ZF=0
JNC  <Jump If No Carry>       | Lompat, jika CF=0
JNG  <Jump If Not Greater>    | Lompat, jika Operand1 <= Operand2
                              | dengan ZF=1 atau SF tidak sama OF
JNGE <Jump If Not Greater     | Lompat, jika Operand1 <= Operand2
      Nor Equal>              | dengan SF tidak sama OF
JNL  <Jump If Not Less>       | Lompat, jika Operand1 >= Operand2
                              | dengan SF=OF
JNLE <Jump If Not Less        | Lompat, jika Operand1 > Operand2
      Nor Equal>              | dengan ZF=0 dan SF=OF
JNO  <Jump If No Overflow>    | Lompat, jika tidak terjadi
                              | tidak terjadi Overflow
JNP  <Jump If Not Parity>     | Lompat, jika Ganjil
JNS  <Jump If No Sign>        | Lompat, jika SF=0
JNZ  <Jump If Not Zero>       | Lompat, jika tidak 0
JO   <Jump On Overflow>       | Lompat, jika OF=1
JP   <Jump On Parity>         | Lompat, jika Genap
JPE  <Jump If Parity Even>    | Lompat, jika PF=1
JPO  <Jump If Parity Odd>     | Lompat, jika PF=0
JS   <Jump On Sign>           | Lompat, jika SF=1
JZ   <Jump Is zero>           | Lompat, jika 0
------------------------------+---------------------------------
```

Gambar 13.2. Daftar Perintah Jump

Bila dilihat pada daftar 13.2., perintah untuk lompat sebenarnya sangat mudah untuk digunakan karena setiap huruf melambangkan suatu kata. Dengan demikian kita tidak perlu untuk mengingat-ingat semua perintah diatas, kita hanya harus ingat bahwa huruf J=Jump, E=Equal, N=Not, S=Sign, Z=Zero, P=Parity, O=Overflow, C=Carry, G=Greater Than, A=Above, L=Less dan B=Below.

Ingatlah:

Huruf G dan L yang artinya Greater Than dan Less digunakan khusus untuk operasi bilangan bertanda. Sedangkan Huruf A dan B yang artinya Above dan Below digunakan khusus untuk operasi bilangan tidak bertanda.

```
;/=============================================\;
;    Program : JMPL.ASM                         ;
;    Author  : S'to                             ;
```

```
;    Fungsi  : Mencetak kalimat secara perkarakter ;
;              sampai ditemui karakter '*'         ;
;                                                  ;
;\================================================/;


       .MODEL SMALL
       .CODE
       ORG 100h

TData : JMP Proses
        Kal DB '    Lucky Luck menembak  ',13,10
            DB 'Lebih cepat dari bayangannya !! ',7,7,'*'
Proses:
        XOR  BX,BX          ; BX=0
        MOV  AH,02h         ; Servis Untuk Cetak Karakter
 Ulang:
        CMP  Kal[BX],'*'    ; Bandingkan dengan '*'
        JE   Exit           ; Jika Sama Lompat ke Exit
        MOV  DL,Kal[BX]     ; Masukkan karakter ke BX menuju DL
        INT  21h            ; Cetak karakter
        INC  BX             ; Tambah 1 pada BX
        JMP  Ulang          ; Lompat Ke Ulang

 Exit : INT  20h      ; Selesai ! kembali ke DOS
END     TData
```

Program 13.2. Perbandingan

Bila program 13.2. dijalankan, maka pada layar akan ditampilkan:

Lucky Luck menembak

Lebih cepat dari bayangannya !!

Angka 7 pada akhir kalimat digunakan untuk menghasilkan suara beep. Bila anda masih ingat pada addressing yang telah kita pelajari, maka program 13.2. tentunya tidak ada masalah.

BAB XIV
STACK

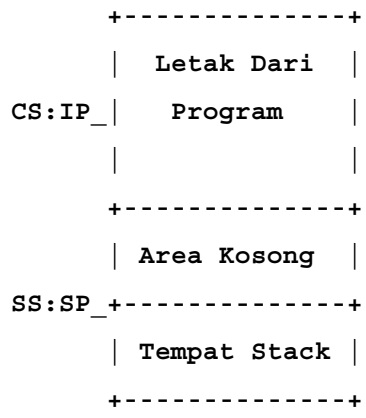
14.1. APA ITU STACK ?

Bila kita terjemahkan secara bebas, stack artinya adalah 'tumpukan'. Stack adalah bagian memory yang digunakan untuk menyimpan nilai dari suatu register untuk sementara. Operasi- operasi pada assembler yang langsung menggunakan stack misalnya pada perintah PUSH, POP, PUSF dan POPF.

Pada program COM yang hanya terdiri atas satu segment, dimanakah letak dari memory yang digunakan untuk stack ?. Seperti pasangan CS:IP yang menunjukkan lokasi dari perintah selanjutnya yang akan dieksekusi, pada stack digunakan pasangan SS:SP untuk menunjukkan lokasi dari stack. Untuk itu marilah kita lihat dengan debug:

```
C:\>debug
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=3143 ES=3143 SS=3143 CS=3143 IP=0100 NV UP EI PL NZ NA PO NC
3143:0100 0F           DB     0F
-q
```

Dari percobaan ini dapat kita lihat bahwa SS menunjukkan angka yang sama dengan CS(3143) atau dengan kata lain CS dan SS berada pada satu segment. Register IP yang menunjukkan lokasi stack bernilai FFFE atau dengan kata lain stack terletak pada akhir segment. Karena inilah pada program COM sebaiknya anda jangan sembarangan mengubah data pada akhir segment, karena hal ini akan mengacaukan program. Bila kita gambarkan letak dari stack akan tampak seperti gambar 14.1



Gambar 14.1. Lokasi Stack

14.2. CARA KERJA STACK

Seperti yang telah dikatakan, bahwa stack digunakan sebagai tempat penampung sementara nilai dari suatu register. Supaya lebih jelas lihatlah cara kerja dari program 14.1.

```
;/=======================================\;
;      Program : NSTACK.ASM               ;
;      Author  : S'to                     ;
;      Fungsi  : Mencetak kalimat 2 kali  ;
;                dengan operasi yang mirip ;
;                dengan stack              ;
;\=======================================/;

        .MODEL SMALL
        .CODE
        ORG 100h

TData : JMP Proses
        Kal    DB 'LANG LING LUNG  $'
        Ganti  DB  13,10,'$'
        Stacks DW ?
Proses:
        LEA  DX,Kal
        MOV  Stacks,DX

        MOV  AH,09
        INT  21h
        LEA  DX,Ganti
        INT  21h

        MOV  DX,Stacks
        INT  21h

 Exit : INT  20h
END     TData
```

Program 14.1. Mencetak kalimat 2 kali

Bila program 14.1. dan 14.2. dijalankan, maka pada layar akan ditampilkan:

LANG LING LUNG
LANG LING LUNG

Perhatikanlah, perintah:

LEA DX,Kal
MOV Stacks,DX

Pada baris pertama kita mendapatkan alamat efektif dari "Kal" dan disimpan pada DX. Kemudian kita simpan nilai DX yang menunjuk pada offset "Kal" ini pada variabel Stacks. Sehingga pada saat kita hendak mencetak 'Kal' untuk kedua kalinya, kita tinggal mengambil nilai dari variabel Stacks dengan perintah "**MOV DX,Stacks**".

Kini akan kita lihat bagaimana menggunakan stack yang sebenarnya untuk

tugas ini.

```
;/=======================================\;
;      Program : STACK.ASM                ;
;      Author  : S'to                     ;
;      Fungsi  : Mencetak kalimat 2 kali  ;
;                dengan operasi stack yang ;
;                sebenarnya                ;
;\=======================================/;

        .MODEL SMALL
        .CODE
        ORG 100h

TData : JMP Proses
        Kal    DB 'LANG LING LUNG  $'
        Ganti  DB  13,10,'$'
        Stacks DW ?
Proses:
        LEA  DX,Kal
        PUSH DX

        MOV  AH,09
        INT  21h
        LEA  DX,Ganti
        INT  21h

        POP  DX
        INT  21h

 Exit : INT  20h
END     TData
```

Program 14.2. Operasi Stack

Dengan perintah "**PUSH**", kita menyimpan nilai register DX pada stack, kemudian pada perintah "**POP**" kita mangambil keluar nilai yang disimpan tersebut dari stack. Dari program ini dapat dilihat bagaimana stack menggantikan varibel pada program 14.1. yang digunakan untuk menyimpan nilai pada register DX.

Kini lihatlah bagaimana program yang menggunakan pengulangan didalam pengulangan dengan memanfaatkan stack ini. Dalam bahasa Pascal programnya akan tampak seperti berikut:

```
For i:= 10 DownTo 1 Do
   For j:= 5 DownTo 1 Do
      For s:= 3 DownTo 1 Do
      Begin
      End
```

Dalam bahasa assembler akan tampak seperti:

```
        MOV     CX,10
      i:
        PUSH    CX
        MOV     CX,5
```

```
j:
  PUSH    CX
  MOV     CX,3
s:
  LOOP    s
  POP     CX
  LOOP    j
  POP     CX
  LOOP    i
```

14.3. PUSH DAN POP

Stack dapat kita bayangkan sebagai sebuah tabung yang panjang. Sedangkan nilai pada register dapat dibayangkan berbentuk koin yang dapat dimasukkan dalam tabung tersebut.

Untuk memasukkan nilai suatu register pada stack, digunakan perintah push dengan syntax:

PUSH Reg16Bit

Sebagai contohnya pada perintah:

```
MOV  AX,12
MOV  BX,33
MOV  CX,99
PUSH AX    ; Simpan nilai AX pada stack
PUSH BX    ; Simpan nilai BX pada stack
PUSH CX    ; Simpan nilai CX pada stack
```

Maka pada stack akan tampak seperti:

<<<< Gbr142.PIX >>>>>

Gambar 14.2. Penyimpanan Nilai Pada Stack

Dari gambar 14.2. dapat anda lihat bahwa nilai yang terakhir dimasukkan(99) akan terletak pada puncak tabung stack.

Untuk mengambil keluar koin nilai pada tabung stack, digunakan perintah pop dengan syntax:

POP Reg16Bit

Perintah POP akan mengambil koin nilai pada stack yang paling atas dan dimasukkan pada Reg16Bit. Dari sini dapat anda lihat bahwa data yang terakhir dimasukkan akan merupakan yang pertama dikeluarkan. Inilah sebabnya operasi stack dinamankan **LIFO**(Last In First Out).

Sebagai contohnya, untuk mengambil nilai dari register AX, BX dan CX yang disimpan pada stack harus dilakukan pada register CX dahulu barulah BX dan AX, seperti:

```
POP CX   ; Ambil nilai pada puncak stack, masukkan ke CX
POP BX   ; Ambil nilai pada puncak stack, masukkan ke BX
POP AX   ; Ambil nilai pada puncak stack, masukkan ke AX
```

Perhatikan:

Bila anda terbalik dalam mengambil nilai pada stack dengan POP AX kemudian POP BX dan POP CX, maka nilai yang akan anda dapatkan pada register AX, BX dan CX akan terbalik. Sehingga register AX akan bernilai 99 dan CX akan bernilai 12.

TRIK:

Seperti yang telah kita ketahui, data tidak bisa dicopykan antar segment atau memory. Untuk mengcopykan data antar segment atau memory anda harus menggunakan register general purpose sebagai perantaranya, seperti:

```
MOV AX,ES     ; Untuk menyamakan register
MOV DS,AX     ;      ES dan DS
```

Dengan adanya stack, anda bisa menggunakannya sebagai perantara, sehingga akan tampak seperti:

```
PUSH ES    ; Untuk menyamakan register
POP  DS    ;      ES dan DS
```

14.4. PUSF DAN POPF

PUSF dan POPF, sama halnya dengan perintah PUSH dan POP. Perintah PUSF digunakan untuk menyimpan nilai dari flags register pada stack sedangkan POPF digunakan untuk mengambil nilai pada stack dan disimpan pada flags register. Kedua perintah ini digunakan tanpa operand:

```
PUSHF  ; Simpan nilai Flags pada stack
POPF   ; Ambil nilai pada stack
```

Perintah PUSHF dan POPF digunakan untuk menyelamatkan kondisi dari flag terhadap perubahan. PUSHF dan POPF biasanya digunakan pada operasi yang sangat mementingkan nilai pada flag ini, seperti pada operasi aritmatika.

BAB XV
MASUKAN DARI KEYBOARD

Keyboard merupakan sarana bagi kita untuk berkomunikasi dengan program. Pada bagian ini akan kita lihat bagaimana caranya untuk menanggapi masukan dari keyboard. Tetapi sebelumnnya anda tentunya harus mengerti sedikit mengenai beberapa hal penting yang berkaitan dengan keyboard itu.

15.1. KODE SCAN DAN ASCII

Prosesor pada keyboard mendeteksi setiap penekanan maupun pelepasan tombol pada keyboard. Prosesor ini menterjemahkan setiap sinyal yang terjadi berdasarkan posisi tertentu menjadi apa yang dinamakan kode Scan. Dengan demikian tombol "A" dan "B" akan mempunyai kode Scan yang berbeda karena posisinya memang berbeda. Lain halnya untuk tombol "A"<A besar> dan "a"<a kecil> yang terdapat pada posisi yang sama, akan mempunyai kode Scan yang sama. Kode Scan ini biasanya tidak berguna bagi kita. Kita biasanya hanya menggunakan kode ASCII dan Extended yang merupakan hasil terjemahan dari kode scan oleh keyboard handler.

Kode ASCII adalah kode yang melambangkan suatu karakter baik berupa huruf,angka, maupun simbol-simbol grafik. Misalkan angka "1" akan dilambangkan dengan kode ASCII 49. Untuk kode ASCII ini bisa anda lihat pada lampiran.

15.2 APA ITU KODE EXTENDED ?

Kode ASCII telah menyediakan sebanyak 256 karakter dengan beberapa karakter kontrol, misalnya #10 untuk pindah baris dan #13 untuk Enter yang akan menggerakkan kursor kesamping kiri. Tetapi fungsi yang telah disediakan ini tidak mampu untuk menampilkan ataupun mendeteksi tombol fungsi misalnya F1, F2, F3 dan Home. Tombol kombisasi juga tidak dapat dideteksi oleh karakter ASCII , misalnya penekan tombol shif disertai tombol F1, penekanan Ctrl disertai tombol Home, dan lain-lain. Penekanan terhadap tombol-tombol fungsi dan tombol kombinasi akan menghasilkan kode ASCII 0<nil>.

Karena alasan diatas maka diciptakanlah suatu kode yang dinamakan sebagai kode EXTENDED. Kode Extended ini dapat mendeteksi penekanan terhadap tombol-tombol fungsi maupun tombol kombinasi. Untuk kode extended bisa anda lihat pada lampiran.

15.3. MASUKAN SATU KARAKTER

Interupsi dari BIOS, yaitu interupsi 16h servis 0 dapat digunakan untuk mendapatkan masukan satu karakter dari keyboard.

Hasil dari pembacaan karakter fungsi ini akan diletakkan pada register AX. Bila terjadi penekanan pada tombol biasa maka byte rendah dari AX<AL>, akan menunjukkan kode ASCII dari tombol tersebut dan byte tnggi dari AX<AH> akan berisi kode Scan dari tombol tersebut.

Bila yang ditekan adalah tombol khusus(extended) yang akan menghasilkan kode ASCII 0 maka byte rendah dari register AX<AL> akan menghasilkan kode ASCII 0 dan byte tinggi dari AX<AH> akan akan berisi kode extended dari tombol

tersebut.

```
;/=======================================\;
;        Program : READKEY.ASM            ;
;        Author  : S'to                   ;
;        Fungsi  : Input satu karakter    ;
;                  dari keyboard.         ;
;=======================================;
;             INTERUPSI 16h               ;
;=======================================;
;     Input:        OutPut:               ;
;    AH = 0     Jika tombol biasa, maka:  ;
;               AL = ASCII                ;
;               AH = SCAN                 ;
;                                         ;
;               Jika Tombol khusus, maka  ;
;               AL = 00                   ;
;               AH = Extended             ;
;                                         ;
;\=======================================/;

        .MODEL SMALL
        .CODE
         ORG 100h

TData : JMP   Proses
        T_ASCII    DB 13,10,'Ini adalah tombol ASCII : $'
        T_Extended DB 13,10,'Ini adalah tombol Extended  $'
 Proses :
          MOV  AH,0          ; Servis Input satu karakter
          INT  16h           ; Laksanakan
          PUSH AX            ; Simpan hasil pembacaan pada stack

          CMP  AL,00         ; Apakah ini karakter extended ?
          JE   Extended      ; Ya !, Lompat ke Extended
   ASCII:
          LEA  DX,T_ASCII    ; Ambil alamat efektif T_ASCII
          MOV  AH,09         ; Servis cetak kalimat
          INT  21h           ; Cetak kalimat !

          POP  AX            ; Ambil kembali nilai AX pada stack
          MOV  DL,AL         ; Ambil kode ASCII yang ditekan
          MOV  AH,2          ; Servis cetak karakter
          INT  21h           ; Cetak karakter !

          CMP  AL,'Q'        ; Apakah yang ditekan huruf 'Q' ?
          JE   exit          ; Ya !, lompat ke Exit
          CMP  AL,'q'        ; Apakah yang ditekan huruf 'q' ?
          JE   exit          ; Ya !, lompat ke Exit
          JMP  Proses        ; Lompat ke Proses
Extended:
          LEA  DX,T_Extended  ; Ambil alamat efektif T_Extended
          MOV  AH,09          ; Servis cetak kalimat
          INT  21h            ; Cetak kalimat !
          JMP  Proses         ; Lompat ke Proses

    exit: INT  20h            ; Kembali ke DOS !
END       TData
```

Program 15.1. Menunggu masukan satu karakter dari Keyboard

Bila anda menekan tombol extended, seperti penekanan tombol anak panah, F1, F2 dan sebagainya maka pada layar akan ditampilkan :

Ini adalah tombol Extended

Bila anda ingin mengetahui lebih lanjut mengenai tombol apa yang ditekan maka kode extendednya bisa dilihat pada register AH. Sedangkan bila yang ditekan adalah tombol biasa, seperti huruf 'S' maka pada layar akan ditampilkan:

Ini adalah tombol ASCII : S

Program akan selesai jika anda menekan tombol "q" atau "Q".

15.4. MENDETEKSI PENEKANAN SEMBARANG TOMBOL

Dengan fungsi 11h dari interupsi 16h, kita bisa mendeteksi terhadap penekanan tombol,sama halnya seperti yang dilakukan oleh fungsi keypressed pada bahasa pascal. Fungsi ini akan mendeteksi keyboard buffer, bila pada keyboard buffer terdapat suatu tombol maka ia akan membuat zerro flags menjadi nol<0> dan register AL berisi kode ASCII dari karakter tersebut sedangkan register AH akan berisi kode Scan dari tombol tersebut. Sebaliknya jika pada keyboard buffer tidak ada karakter maka zerro flags akan bernilai satu <1>.

Keyboard buffer adalah suatu penampung yang digunakan untuk menampung setiap penekanan tombol pada keybaord. Daya tampung normal dari keyboard buffer adalah 15 karakter. Jika keyboard buffer telah penuh, speaker akan mengeluarkan tanda berupa suara beep.

```
;/=============================================\;
;        Program : KEYPRESS.ASM                ;
;        Author  : S'to                        ;
;        Fungsi  : Mengecek apakah ada         ;
;                  tombol yang ditekan         ;
;=============================================;
;               INTERUPSI 16h                 ;
;=============================================;
;     Input:        OutPut:                     ;
;    AH = 1     Jika Ada tombol yang ditekan   ;
;               ZF = 0 dan                     ;
;                    AL = kode ASCII           ;
;                    AH = Scan Code            ;
;                                              ;
;               Jika Tidak ada penekanan Tombol ;
;               ZF = 1                         ;
;                                              ;
;\=============================================/;

        .MODEL SMALL
        .CODE
         ORG 100h

TData   : JMP   Proses
          Kal0   DB 'Tekan sembarang tombol untuk berhenti ! '
                 DB  13,10,'$'
 Proses :
          MOV  AH,1      ; Servis untuk mengecek buffer keyboard
          INT  16h       ; Laksanakan !
          JNZ  EXIT      ; Jika ada tombol yang ditekan, lompat
```

```
                    ; Ke EXIT
          MOV  AH,09     ; Servis untuk cetak kalimat
          LEA  DX,Kal0   ; Ambil alamat efektif Kal0
          INT  21h       ; Cetak kalimat !
          JMP  Proses    ; Lompat ke Proses

   exit : INT  20h           ; Kembali ke DOS !
END       TData
```

Program 15.2. Membuat fungsi Keypressed

Bila program 15.2. dijalankan, maka pada layar akan ditampilkan tulisan:

Tekan sembarang tombol untuk berhenti !

Tulisan ini akan ditampilkan terus sampai anda menekan sembarang tombol.

15.5. MASUKAN KALIMAT DARI KEYBOARD

Pada program-program sebelumnya kita hanya bisa mendapatkan masukan satu karakter pada keybaord, bagaimana jika diinginkan masukan berupa suatu kalimat? Untuk itu DOS telah menyedikannya.

Interupsi 21h servis ke 0Ah, digunakan untuk mendapatkan masukan dari keyboard lebih dari satu karakter. Adapun aturan pemakainya adalah:

INPUT	OUTPUT
AH = 0Ah	Buffer yang berisi string
DS:DX= Buffer	hasil masukan dari keyboard

Untuk menggunkan fungsi ini anda harus menyediakan sebuah buffer untuk menampung hasil masukan dari keyboard. Anda bisa membuat sebuah buffer seperti:

Buffer DB X,Y,Z DUP(?)

Pada byte pertama yang kita gambarkan sebagai **"X"**, digunakan sebagai tanda dari banyaknya karakter yang dapat dimasukkan dari keyboard ditambah 1. Seperti bila anda memberikan nilai 23, maka karakter maksimum yang dapat dimasukkan adalah 22 karakter, karena satu karakter lagi digunakan khusus oleh tombol Enter(0Dh).

Pada byte kedua yang kita gambarkan sebagai **"Y"** ,digunakan oleh fungsi ini sebagai indikator banyaknya karakter yang telah diketikkan oleh user(Tombol Enter<0Dh> tidak akan dihitung). Anda bisa memberikan tanda **"?"** untuk byte kedua ini, karena nilainya akan diisi secara otomatis nantinya.

Pada byte ketiga yang kita gambarkan sebagai **"Z"** inilah yang nantinya merupakan awal dari masukan string akan ditampung. Anda harus menyediakan banyaknya byte yang dibutuhkan, sesuai dengan byte pertama(**"X"**).

```
;/=========================================================\;
;                  Program : IN-KAL.ASM                    ;
;                  Author  : S'to                          ;
;                  Fungsi  : Input Kalimat dari            ;
;                            keyboard.                     ;
;=========================================================;
;                     INTERUPSI 21h                        ;
;=========================================================;
;     Input:                                               ;
;    AH    = 0Ah                                           ;
;    DS:DX = Penampung dengan spesifikasi:                 ;
;          Byte 1 = Maksimum karakter yang dapat dimasukkan ;
;          Byte 2 = Akan dijadikan Indikator banyaknya     ;
;                   karakter yang dimasukkan               ;
;          Byte 3 keatas = Tempat hasil masukan ditampung  ;
;                                                          ;
;                                                          ;
;\=========================================================/;

        .MODEL SMALL
        .CODE
         ORG 100h

TData : JMP   Proses
          T_Enter  EQU  0Dh
          Kal0     DB   'Ketikkan satu Kalimat  : $'
          Kal1     DB   13,10,'Kalimat pada buffer    : $'
          Buffer   DB   23,?,23 DUP(?)
Proses :  MOV  AH,09
          LEA  DX,Kal0
          INT  21h            ; Cetak kalimat Kal0

          MOV  AH,0Ah         ; Servis Input kalimat
          LEA  DX,Buffer      ; DX menunjuk pada offset Buffer
          INT  21h            ; Input kalimat !

          MOV  AH,09
          LEA  DX,Kal1
          INT  21h            ; Cetak kalimat Kal1

          LEA  BX,Buffer+2    ; BX menunjuk byte ke 3 Buffer
  Ulang:
          CMP  BYTE PTR [BX],T_Enter ; Apakah karakter Enter?
          JE   EXIT                  ; Ya! Lompat ke Exit

          MOV  DL,[BX]               ; Masukkan karakter pada DL
          MOV  AH,02                 ; Servis cetak karakter
          INT  21h                   ; Cetak karakter

          INC  BX                    ; BX := BX+1
          JMP  Ulang                 ; Lompat ke Ulang

   EXIT:  INT  20h                   ; Kembali ke DOS !
END       TData
```

Program 15.3. Masukan string dari Keyboard

Contoh dari hasil eksekusi program 15.3. setelah mendapat masukan dari keyboard:

```
Ketikkan satu Kalimat  : Equasoft
```

Kalimat pada buffer : Equasoft

Adapun proses yang dilakukan pada program 15.3. adalah:

```
MOV  AH,09
LEA  DX,Kal0
INT  21h
```

Pertama-tama cetak kalimat Kal0 dengan servis 9 interupsi 21h, setelah itu:

```
MOV  AH,0Ah
LEA  DX,Buffer
INT  21h
```

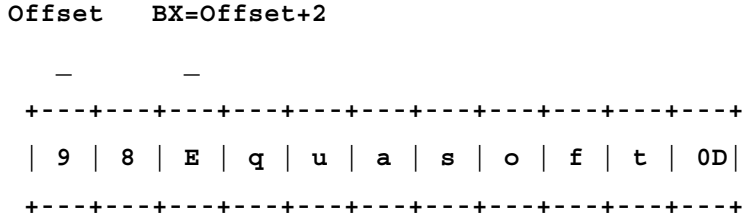
Pada bagian inilah kita meminta masukan dari keyboard, dengan DX menunjuk pada buffer yang digunakan sebagai penampung.

```
MOV  AH,09
LEA  DX,Kal1
INT  21h
```

Setelah itu cetaklah kalimat pada Kal1

```
LEA  BX,Buffer+2
```

Dengan perintah ini maka BX akan menunjuk pada byte ke 3, atau awal masukan string dari keyboard. Supaya lebih jelas, nilai pada buffer setelah mendapat masukan adalah:



Setelah BX mnunjuk pada karakter pertama hasil masukan, maka:

```
CMP  BYTE PTR [BX],T_Enter
JE   EXIT
```

Periksalah, apakah karakter yang ditunjukkan BX adalah 0D(Enter)? Bila ya, berarti akhir dari masukan. Perlu anda perhatikan disini, bahwa kita menggunakan BYTE PTR. Bila tidak digunakan, assembler akan bingung apakah kita ingin membandingkan isi alamat BX sebanyak 1 byte atau lebih dengan T_Enter.

```
MOV  DL,[BX]
MOV  AH,02
INT  21h
```

Bila bukan karakter enter, maka ambil karakter tersebut dan masukkan

pada register DL untuk dicetak.

```
INC  BX
JMP  Ulang
```

Tambahlah BX denga satu sehingga BX akan menunjuk pada karakter selanjutnya. Proses dilanjutkan sampai ketemu tanda 0D atau karakter Enter.

BAB XVI
PROCEDURE

Procedure merupakan suatu alat bantu yang sangat berguna. Dengan procedure suatu program yang besar bisa diselesaikan dengan lebih mudah. Proses pencarian kesalahanpun akan lebih mudah bila digunakan procedure.

16.1. MEMBUAT PROCEDURE

Untuk membuat procedure bisa anda gunakan bentuk seperti pada gambar 16.1.

```
-----------------------------------------------------------
        NamaP   PROC  NEAR/FAR
                +---------+
                | Program |
                +---------+
                RET
        NamaP   ENDP
-----------------------------------------------------------
```

Gambar 16.1. Model Procedure

"NamaP" adalah nama dari procedure yang kita definisikan sendiri. Untuk memudahkan nama untuk procedure bisa anda definisikan sesuai dengan fungsi dari procedure tersebut, seperti CLS untuk procedure yang tugasnya menghapus layar.

Dibelakang kata "PROC" anda harus memilih bentuk dari procedure tersebut, yaitu "NEAR" atau "FAR". Bentuk "NEAR" digunakan jika procedure tersebut nantinya dipanggil oleh program yang letaknya masih satu segment dari procedure tersebut. Pada program COM yang terdiri atas satu segment, kita akan selalu menggunakan bentuk "NEAR". Sebaliknya bentuk "FAR" ini digunakan bila procedure dapat dipanggil dari segment lain. Bentuk ini akan kita gunakan pada program EXE.

Perintah "RET(Return)" digunakan untuk mengembalikan Kontrol program pada sipemanggil procedure. Pada bentuk NEAR perintah RET ini akan memPOP atau mengambil register IP dari stack sebagai alamat loncatan menuju program pemanggil procedure. Sedangkan pada bentuk "FAR" perintah RET akan mengambil

register IP dan CS dari stack sebagai alamat loncatan menuju program pemanggil procedure. Alamat kembali untuk procedure disimpan pada stack pada saat procedure tersebut dipanggil dengan perintah "CALL", dengan syntax:

CALL NamaP

Perintah Call ini akan menyimpan register IP saja bila procedure yang dipanggil berbentuk "NEAR". Bila procedure yang dipanggil berbentuk "FAR", maka perintah "CALL" akan menyimpan register CS dan IP.

16.2. MENGGUNAKAN PROCEDURE

Sebagai contoh dari pemakaian procedure akan kita lihat pada program 16.1 yang mencetak karakter dengan procedure.

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~;
;   PROGRAM : PROC_KAR.ASM       ;
;   FUNGSI  : MENCETAK KARATER   ;
;             DENGAN PROCEDURE   ;
;                                ;
;==========================S'to=;

         .MODEL SMALL
         .CODE
         ORG 100h

Proses :  CALL Cetak_Kar  ; Panggil Cetak_Kar
          INT  20h

Cetak_Kar PROC  NEAR
          MOV  AH,02h
          MOV  DL,'S'
          INT  21h        ; Cetak karakter
          RET             ; Kembali kepada si pemanggil
Cetak_Kar ENDP            ; END Procedures

END       Proses
```

Program 16.1 Menggunakan Procedure

Bila program 16.1. dijalankan, maka pada layar akan ditampilkan huruf "S". Untuk membuat sebuah procedure ingatlah untuk menyimpan semua register yang digunakan oleh procedure tersebut dan mengembalikan semua isi register pada akhir procedure. Hal ini dilakukan untuk menjaga supaya program utama yang menggunakan procedure tersebut tidak menjadi kacau nantinya. Sebagai contohnya bisa anda lihat pada program 16.2.

```
;~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~;
;   PROGRAM : PROC_KA1.ASM       ;
;   FUNGSI  : MENCETAK KARATER   ;
```

```
;             DENGAN PROCEDURE  ;
;                               ;
;=========================S'to=;

         .MODEL SMALL
         .CODE
         ORG 100h

TData  :  JMP   Proses
          Kar   DB ?
          Klm   DB 'BATMAN SI MANUSIA KELELAWAR ' ; 28 Karakter

Proses :  MOV   CX,28       ; Banyaknya pengulangan
          XOR   BX,BX       ; Addressing Mode
Ulang  :
          MOV   DL,Klm[BX]
          MOV   Kar,DL
          CALL  Cetak_Kar   ; Panggil Cetak_Kar
          INC   BX
          LOOP  Ulang

          INT   20h

Cetak_Kar PROC  NEAR
          PUSH  AX        ; Simpan semua register
          PUSH  DX        ; Yang digunakan

          MOV   AH,02h
          MOV   DL,Kar
          INT   21h       ; Cetak karakter

          POP   DX        ; Kembalikan semua register
          POP   AX        ; Yang disimpan
          RET             ; Kembali kepada si pemanggil
Cetak_Kar ENDP            ; END Procedures

END       TData
```

Program 16.2. Menggunakan Procedure

Bila program 16.2. dijalankan, maka pada layar akan ditampilkan:

BATMAN SI MANUSIA KELELAWAR

Pada procedure kita tidak bisa menggunakan parameter, inilah salah satu kelemahan dari procedure yang sangat berarti. Untuk menggunakan parameter anda harus menggunakan MACROS.

BAB XVII
MACRO

Macro hampir sama dengan procedure, yang dapat membantu anda dalam membuat program yang besar. Dengan Macro anda tidak perlu menggunakan perintah "CALL" dan anda juga bisa menggunakan parameter dengan mudah. Suatu ciri dari pemrograman bahasa tingkat tinggi!

17.1. MEMBUAT MACRO

Macro adalah lebih mudah dibuat daripada procedure. Untuk membuat Macro bisa anda gunakan bentuk seperti pada gambar 17.1.

```
-----------------------------------------------------------------
                    NamaM  MACRO  [P1,P2,,]
                           +------------+
                           | Program    |
                           +------------+
                           ENDM
-----------------------------------------------------------------
```

Gambar 17.1. Model Macro

"P1" dan "P2" adalah parameter yang bisa anda gunakan pada macro. Parameter ini berbentuk optional, artinya bisa digunakan ataupun tidak. Supaya lebih jelas bisa anda lihat pada program MAC1 yang menggunakan macro ini untuk mencetak karakter.

```
Cetak_Kar   MACRO  Kar
              MOV    CX,3
              MOV    AH,02
              MOV    DL,Kar
      Ulang :
              INT    21h      ; Cetak Karakter
              LOOP   Ulang
            ENDM               ; End Macro

;-----------------------------------;
;       Program : MAC1.ASM          ;
;       Fungsi  : Menggunakan Macro ;
;                 Untuk mencetak    ;
;                 huruf 'SSS'        ;
;-----------------------------------;

        .MODEL  SMALL
        .CODE
        ORG 100h
Proses:
        Cetak_Kar 'S'        ; Cetak Huruf S

        INT    20h
        END    Proses
```

Program 17.1. Menggunakan Macro

Dari program MAC1 bisa anda lihat betapa mudahnya untuk menggunakan macro. Pada procedure, setiap kali kita memanggilnya dengan perintah CALL maka program akan melompat pada procedure tersebut, sehingga setiap procedure hanya terdapat satu kali saja pada program. Lain halnya dengan Macro, setiap terjadi pemanggilan terhadap macro atau dapat dikatakan secara kasar, setiap kita memanggil macro dengan menuliskan nama macronya dalam program, maka seluruh isi macro akan dipindahkan pada program yang memanggilnya. Dengan demikian bila pada program anda memanggil suatu macro sebanyak 10 kali maka macro tersebut akan disisipkan 10 kali pada program. Hal inilah yang menyebabkan program yang menggunakan macro ukuran programnya menjadi lebih besar. Tetapi hal ini juga yang menyebabkan program yang menggunakan macro lebih cepat daripada procedure, karena pada procedure komputer harus melakukan lompatan tetapi pada macro tidak perlu.

17.2. LABEL PADA MACRO

Pada macro anda bisa menggunakan label seperti biasa. Tetapi anda harus ingat, karena setiap pemanggilan Macro akan menyebabkan seluruh isi macro tersebut disisipkan pada program, maka pada macro yang didalamnya menggunakan label hanya dapat dipanggil sebanyak satu kali. Bila anda menggunakanya lebih dari satu kali maka akan terjadi "****Error** Symbol already defined elsewhere: ULANG**" karena dianggap kita menggunakan label yang sama.

Untuk menghindari hal itu, gunakanlah directif LOCAL. Dengan directif LOCAL assembler akan membedakan label tersebut setiap kali terjadi pemanggilan terhadapnya.

```
Cetak_Kar   MACRO    Kar
              LOCAL  Ulang    ; Label 'Ulang' jadikan Local
              MOV    CX,3
              MOV    AH,02
              MOV    DL,Kar
       Ulang:
              INT    21h      ; Cetak Karakter
              LOOP   Ulang
            ENDM               ; End Macro

;-----------------------------------;
;       Program : MAC2.ASM          ;
;       Fungsi  : Menggunakan Macro ;
;                 Untuk mencetak    ;
;                 huruf 'PPPCCC'    ;
;-----------------------------------;

        .MODEL  SMALL
        .CODE
        ORG 100h
Proses:
        Cetak_Kar 'P'         ; Cetak Huruf P
        Cetak_Kar 'C'         ; Cetak Huruf C

        INT    20h
        END    Proses
```

Program 17.2. Menggunakan Perintah LOCAL

17.3. PUSTAKA MACRO

Bila kita sering menggunakan suatu fungsi seperti mencetak kalimat pada setiap program yang kita buat, tentu saja akan sangat membosankan karena setiap kali kita harus membuat fungsi yang sama. Dengan macro anda bisa menghindari hal tersebut dengan membuat suatu pustaka macro. Pustaka tersebut bisa anda simpan dengan suatu nama, misalnya 'pustaka.mcr'. File yang tersimpan adalah dalam bentuk ASCII, tanpa perlu di compile.

```
;/========================\;
;   Program : PUSTAKA.MCR  ;
;\========================/;

Cetak_Kar  MACRO  Kar        ; Macro untuk mencetak
           MOV  AH,02        ; Karakter
           MOV  DL,Kar
           INT  21h
           ENDM

Cetak_Klm  MACRO  Klm        ; Macro untuk mencetak
           LEA    DX,Klm     ; kalimat
           MOV    AH,09
           INT    21h
           ENDM
```

Program 17.3. Pustaka.MCR

Setelah program 17.3. anda ketikkan, simpanlah dengan nama 'PUSTAKA.MCR'. Anda bisa menggunakan macro pada file pustaka.mcr dengan hanya manambahkan kata:

INCLUDE PUSTAKA.MCR

Sebagai contohnya bisa anda lihat pada program 17.4. yang menggunakan file pustaka.mcr ini untuk mencetak kalimat dan karakter.

```
;/====================================\;
;        Program : TPTK.ASM            ;
;        Fungsi  : Contoh menggunakan  ;
;                  pustaka macro       ;
;\====================================/;

INCLUDE   PUSTAKA.MCR          ; Gunakan file PUSTAKA.MCR

          .MODEL  SMALL
          .CODE
          ORG  100h

TData :   JMP  Proses
          Kal0  DB 'PENGGUNAAN PUSTAKA MACRO $'

Proses:
          Cetak_Klm  Kal0       ; Cetak Kalimat Kal0
          Cetak_Kar  'Y'        ; Cetak Huruf 'Y'
```

```
        INT   20h
END     TData
```

Program 17.4. Menggunakan Pustaka.MCR

Setelah program 17.4. anda jalankan, maka pada layar akan ditampilkan:

PENGGUNAAN PUSTAKA MACRO Y

17.4. MACRO ATAU PROCEDURE ?

Banyak pro dan kontra mengenai macro dan procedure ini. Sebagian orang menganggap macro akan merugikan program, tetapi banyak juga yang menganggap macro adalah pemecahan yang tepat dalam pemrograman assembler yang terkenal sulit untuk digunakan. Kini apa yang akan anda pakai ? Macro ataukah procedure ? Untuk itu marilah kita lihat dahulu perbedaan antara procedure dan macro ini.

- Procedure tidak memperpanjang program, karena hanya muncul sekali saja pada program.
- Macro akan muncul pada program setiap terjadi pemanggilan terhadap macro, sehingga macro akan memperpanjang program.
- Untuk menggunakan procedure anda harus memanggilnya dengan perintah CALL dan dalam procedure diakhiri dengan RET.
- Macro bisa anda gunakan dengan memanggil langsung namanya dan pada macro tidak perlu diakhiri dengan RET.
- Procedure akan memperlambat program, karena setiap pemanggilan terhadap procedure, komputer harus melakukan lompatan.
- Macro tidak memperlambat program karena komputer tidak perlu melakukan lompatan.
- Pada procedure anda tidak bisa menggunakan parameter secara langsung. Bila anda ingin menggunakan parameter bisa dengan melalui stack atau register.
- Macro dengan mudah bisa menggunakan parameter, suatu ciri bahasa tingkat tinggi.
- Macro lebih mudah dibuat dan digunakan daripada procedure.

Setelah melihat perbedaan-perbedaan tersebut, kapankah kita menggunakan procedure dan kapankah menggunakan macro ?

- Jika fungsi tersebut jarang dipanggil, gunakanlah **MACRO** karena macro tidak memperlambat proses.

- Jika fungsi tersebut sering dipanggil, gunakanlah **PROCEDURE k**arena procedure tidak memperbesar program.
- Jika fungsi tersebut kecil, gunakanlah **MACRO**. Karena pengaruh terhadap besarnya program hanya sedikit dan program akan lebih cepat.
- Jika fungsi tersebut besar, gunakanlah **PROCEDURE**. Karena procedure tidak memperbesar program.

BAB XVIII
OPERASI PADA LAYAR

Layar dapat dikatakan merupakan media yang menarik untuk dibahas, karena pada layar ini tampilan program bisa dijadikan semenarik mungkin. Pada bagian ini yang paling penting dan harus anda kuasai adalah bagian yang menerangkan mengenai memory layar yang digunakan sebagai data ditampilkan gambar/teks dilayar.

18.1. MEMORY LAYAR

Pada layar disediakan suatu buffer atau memory yang mencatat tentang apa yang akan ditampilkan dilayar. Komputer akan membaca data pada memory layar untuk memperbaharui tampilan pada layar yang dilakukan kurang lebih 70 kali setiap detiknya. Cepatnya penulisan kembali gambar pada layar ini dinamakan sebagai "refresh rate".

Pada layar monitor monokrom(tidak berwarna), alamat memory yang digunakan sebagai buffer digunakan lokasi memory dimulai pada alamat B000h:0000. Pada monitor berwarna digunakan lokasi memory mulai alamat B800h:0000. Untuk pembahasaan kita selanjutnya akan selalu digunakan alamat buffer layar berwarna(B800h).

18.2. TAMPILAN TEKS PADA LAYAR

Pada modus teks, setiap saat komputer akan selalu melihat pada alamat B800h:0000 sebanyak satu byte untuk menampilkan karakter ASCII pada posisi kolom 0 dan baris 0. Kemudian alamat B800h:0001 digunakan sebagai atribut dari posisi kolom 0 dan baris 0. Alamat B800h:0002 digunakan sebagai data untuk menampilkan karakter ASCII pada posisi kolom 1 dan baris 0. Dan alamat B800h:0003 digunakan sebagai data untuk menampilkan atribut dari posisi kolom 1 baris 0. Demikian seterusnya memory layar digunakan(Lihat gambar 18.1).

<<< Gbr181.PIX >>>

Gambar 18.1. Penggunaan Memory Layar Untuk Menampilkan Teks Dan Atributnya

Dari sini sudah dapat kita ketahui bahwa sebuah karakter pada saat ditampilkan dimonitor menggunakan 2 byte, dimana byte pertama digunakan untuk kode ASCII-nya dan byte berikutnya digunakan untuk atribut dari karakter tersebut.

Karena pada mode default, layar teks dibagi menjadi 80 kolom dan 25

baris(80*25), maka memory yang dibutuhkan untuk satu layar adalah:

80 * 25 * 2 = 4000 Byte

Dengan alamat memory yang digunakan secara berurutan ini, dimana teks akan menempati offset genap dan atribut menempati offset ganjil, alamat dari posisi karakter maupun atribut bisa dihitung dengan menggunakan rumus:

Offset Karakter = (Baris * 160) + (Kolom * 2)

Offset Atribut = (Baris * 160) + (Kolom * 2)+1

Dengan demikian bila kita ingin menampilkan karakter 'S' pada posisi kolom 40 dan baris 12 maka alamat yang digunakan adalah: (12*160)+(40*2)=2000, atau tepatnya B800h:2000. Untuk menampilkan atribut pada posisi kolom 40 dan baris 12 maka alamat yang digunakan adalah:(12*160)+(40*2)+1=2001, atau tepatnya B800h:2001. Sebagai contohnya bisa anda lihat pada program 18.1. yang akan menampilkan karakter "S" pada posisi kolom 40 dan baris 12 dengan atributnya 95.

```
Tulis_Kar   MACRO X,Y,Kar,Attr
            MOV  AX,0B800h
            MOV  ES,AX        ; ES Menunjuk pada segment layar

            MOV  AH,Y
            MOV  AL,160
            MUL  AH           ; Hitung offset baris
            MOV  BX,AX        ; Simpan hasilnya pada BX

            MOV  AH,X
            MOV  AL,2
            MUL  AH           ; Hitung offset kolom
            ADD  BX,AX        ; Tambahkan hasilnya pada BX

            MOV  AL,Kar       ; AL=karakter yang akan ditampilkan
            MOV  AH,Attr      ; AH=Atribut yang akan ditampilkan
            MOV  ES:[BX],AL   ; Tampilkan Karakter dan atributnya
            MOV  ES:[BX+1],AH ; pada posisi kolom X dan baris Y
            ENDM

;/=============================================\;
;           Program : LAYAR1.ASM                ;
;           Author  : S'to                      ;
;                                               ;
; Fungsi  : Menampilkan karakter dan  atributnya ;
;           dengan menuliskannya langsung pada  ;
;           memory layar                        ;
;\=============================================/;

         .MODEL SMALL
         .CODE
         ORG 100h
Proses :
         Tulis_Kar 40 12 'S' 95  ; Tulis karakter 'S' dengan
                                 ; no atribut 95 pada posisi
         INT  20h                ; kolom 40 dan baris 12
END      Proses
```

Program 18.1. Menuliskan langsung pada memory layar

Dengan mengertinya anda pada program 18.1. ini maka banyak program menarik yang dapat anda hasilkan, seperti program rontok, menu sorot, shadow dan lain sebagainya.

18.2.1 MEMBUAT PROGRAM RONTOK

Pada bagian ini akan kita lihat, bagaimana caranya menggeser tulisan dengan mengakses memory layar secara langsung dengan program rontok.

Program rontok adalah program yang akan membersihkan layar dengan cara menjatuhkan atau merontokkan huruf pada layar satu persatu.

```
Delay  MACRO
       PUSH  CX                ; Macro ini digunakan untuk
       XOR   CX,CX             ; menunda program, dan
Loop1:
       LOOP  Loop1             ; hanya melakukan looping
       POP   CX
       ENDM

Geser  MACRO  PosY
       PUSH   AX
       PUSH   BX
       PUSH   CX          ; Simpan semua register yang digunakan

       XOR    CX,CX
       MOV    AL,26
       SUB    AL,PosY
       MOV    CL,AL       ; CX=banyaknya pergeseran kebawah
Loop2:
       MOV    AL,BYTE PTR ES:[BX]       ; AL=Karakter pada layar
       MOV    BYTE PTR ES:[BX+160],AL   ; Geser ke bawah
Hilang:
       MOV    BYTE PTR ES:[BX],' '    ; Hapus karakter
                                      ; sebelumnya
       Delay                          ; delay, supaya bisa
                                      ; terlihat
       ADD    BX,160                  ; Menuju baris selanjutnya
       LOOP   Loop2                   ; Ulangi ke Loop2

       POP    CX
       POP    BX
       POP    AX    ; Kembalikan semua register yang digunakan
ENDM
```

```
;/================================================\;
;          Program : RONTOK.ASM                    ;
;          Author  : S'to                          ;
;          Fungsi  : Membersihkan layar dengan cara   ;
;                    merontokkan hurufnya satu persatu;
;                                                    ;
;\================================================/
```

```
        .MODEL SMALL
        .CODE
        ORG 100h

TData :   JMP Proses
          PosY  DB  ?
Proses:
          MOV   AX,0B800h
          MOV   ES,AX          ; ES mencatat segment layar

          MOV   BX,3998        ; Posisi karakter 80,25
          MOV   CX,25          ; Banyaknya pengulangan baris
UlangY  :
          MOV   PosY,CL        ; PosY mencatat posisi baris
          PUSH  CX             ; CX mencatat posisi Y
          MOV   CX,80          ; Banyaknya pengulangan Kolom
UlangX  :
          CMP   BYTE PTR ES:[BX],33   ; Apakah ada karakter
                                      ; pada layar ?
          JB    Tdk            ; Lompat ke Tdk, jika tidak ada
          Geser PosY           ; Geser karakter tersebut ke bawah
   Tdk  :
          SUB   BX,2           ; BX menunjuk karakter selanjutnya
          LOOP  UlangX         ; Proses 80 kali untuk kolom
          POP   CX             ; Ambil posisi Y
          LOOP  UlangY         ; Ulangi dan ganti baris ke atas
   EXIT:
          INT   20h
END       TData
```

Program 18.2. Merontokkan huruf pada layar

Bila program 18.2 dijalankan, maka semua huruf pada layar akan dirontokkan satu persatu sampai habis.

<<< Gbr182.PIX >>>

Gambar 18.2. Hasil eksekusi program 18.2.

Adapun penjelasan programnya adalah:

```
Delay  MACRO
       PUSH  CX
       XOR   CX,CX
Loop1:
       LOOP  Loop1
       POP   CX
       ENDM
```

Macro ini digunakan untuk menunda program. Dengan menolkan CX, maka looping yang akan didapatkan menjadi FFFFh kali, karena pengurangan 0 dengan 1 akan akan menghasilkan nilai -1 atau FFFFh.

```
Geser  MACRO  PosY
       PUSH   AX
       PUSH   BX
       PUSH   CX
```

Pada macro inilah nantinya huruf-huruf pada layar akan digeser. Untuk itu simpanlah semua register yang digunakan oleh macro ini karena pada program utama, register-register juga digunakan.

```
       XOR    CX,CX
       MOV    AL,26
       SUB    AL,PosY
       MOV    CL,AL
```

Ini adalah bagian dari macro geser yang akan menghitung banyaknya pergeseran kebawah yang akan dilakukan, dengan melihat posisi dari huruf yang digeser pada variabel "PosY".

```
Loop2:
       MOV    AL,BYTE PTR ES:[BX]
       MOV    BYTE PTR ES:[BX+160],AL
Hilang:
       MOV    BYTE PTR ES:[BX],' '
       Delay
       ADD    BX,160
       LOOP   Loop2
```

Bagian inilah yang akan menggeser tulisan pada layar. Register BX ditambah dengan 160 untuk mengakses baris dibawahnya.

```
       POP    CX
       POP    BX
       POP    AX
ENDM
```

Pada akhir macro, kembalikanlah semua register yang telah disimpan pada awal macro. Ingat urutannya harus terbalik. Pada program utama:

```
        .MODEL SMALL
        .CODE
```

```
        ORG 100h

TData :   JMP Proses
          PosY  DB  ?
```

Pertama-tama siapkanlah sebuah variabel untuk menampung posisi dari baris yang sedang diakses.

```
Proses:
          MOV   AX,0B800h
          MOV   ES,AX

          MOV   BX,3998
          MOV   CX,25
```

Register ES, kita gunakan sebagai penunjuk segment layar, yaitu pada segment B800h. Register BX yang nantinya akan kita gunakan sebagai penunjuk offset dari ES diberi nilai 3998. Dengan demikian pasangan ES:BP akan menunjuk pada karakter dipojok kanan bawah atau posisi 79,24.

```
UlangY  :
          MOV   PosY,CL
          PUSH  CX
          MOV   CX,80
UlangX  :
          CMP   BYTE PTR ES:[BX],33
          JB    Tdk
          Geser PosY
   Tdk  :
          SUB   BX,2
          LOOP  UlangX
          POP   CX
          LOOP  UlangY
   EXIT:
          INT   20h
END       TData
```

Kemudian lakukanlah proses dengan melihat apakah ada karakter atau tidak. Hal ini dapat dilakukan dengan membandingkannya dengan kode ASCII 33, bila data pada buffer layar dibawah ASCII 33 artinya tidak ada karakter pada

layar.

Jika ada karakter pada layar maka proses geser dilakukan, sebaliknya jika tidak ada karakter proses akan menuju pada posisi selanjutnya dan melakukan hal yang sama.

18.3. MENGGULUNG LAYAR KEATAS ATAU KEBAWAH

BIOS menyediakan suatu fungsi yang dapat digunakan untuk mengulung layar dengan batasan yang kita tentukan. Adapun aturan pemakaian dari interupsi ini adalah:

INPUT:

AH = Diisi dengan 6 untuk menggulung layar keatas, untuk
menggulung layar kebawah diisi dengan 7.

AL = Banyaknya pergeseran yang akan dilakukan. Jika diisi dengan
nol, maka seluruh isi window akan dihapus.

CH = Posisi baris kiri atas window

CL = Posisi kolom kiri atas window

DH = Posisi baris kanan bawah window

DL = Posisi kolom kanan bawah window

BH = Atribut yang akan mengisi hasil penggulungan window

Setelah semuanya anda persiapkan laksanakanlah interupsi 10h. Anda bisa membersihkan layar dengan fungsi ini dengan meletakkan 0 pada register AL, dan membuat window pada posisi 0,0 dan 79,24.

```
DELAY  MACRO            ; Macro untuk menunda program
       LOCAL Ulang
       PUSH CX
       XOR  CX,CX
 Ulang:
       LOOP Ulang
       POP  CX
       ENDM

Scrool MACRO  X1,Y1,X2,Y2,Arah
       PUSH   CX
       MOV    AH,Arah      ; Servis Gulung keatas atau kebawah
       MOV    AL,1         ; Jumlah Baris
       MOV    CL,X1        ; Kolom kiri atas
       MOV    CH,Y1        ; Baris kiri Atas
       MOV    DL,X2        ; Kolom kanan bawah
       MOV    DH,Y2        ; Baris kanan bawah
       MOV    BH,01000111b ; Atribut hasil penggulungan
       INT    10h
       POP    CX
       ENDM

;/=============================================\;
;               Program : SCROOL.ASM            ;
;               Author  : S'to                  ;
;               Fungsi  : Menggulung layar      ;
```

```
;\=============================================/;

      .MODEL SMALL
      .CODE
      ORG  100h

TData : JMP      Proses
        G_Atas   EQU 6         ; Servis untuk menggulung ke atas
        G_Bawah  EQU 7         ; Servis untuk menggulung ke bawah
Proses:
       MOV  CX,7
 Ulang:
       Scrool 20  7  60  14  G_Bawah
       delay
       LOOP   Ulang
       INT  20h
END    TData
```

Program 18.3. Mengulung layar

Bila program 18.3. anda jalankan, maka pada layar akan tampak seperti gambar 18.2.

<<< Gbr183.PIX >>>

Gambar 18.3. Hasil eksekusi program 18.3.

18.4. MEMINDAHKAN POSISI KURSOR

Untuk memindahkan posisi kursor, sama halnya dengan perintah GOTOXY pada pascal, bisa anda gunakan interupsi dari BIOS. Interupsi yang digunakan adalah interupsi 10h dengan aturan pemakaian:

INPUT:

AH = 2

DH = Posisi Baris(00-24)

DL = Posisi Kolom(00-79)

BH = Halaman layar(0=default)

Adapun contoh dari pemakaian fungsi ini dalam bentuk macro adalah:

```
GOTOXY    MACRO X,Y
          MOV  AH,02
          XOR  BX,BX
          MOV  DH,Y
          MOV  DL,X
          INT  10h
```

```
        ENDM
```

18.5. MENCARI POSISI KURSOR

Sama halnya dengan fungsi WhereX dan WhereY dalam pascal, didalam assembler anda juga bisa mengetahui posisi dari kursor. Untuk itu telah tersedia interupsi 10h dari BIOS dengan aturan pemakaian:

INPUT:	OUTPUT:
AH = 03	DH = Posisi Baris
BH = Halaman Layar(0=default)	DL = Posisi Kolom

Adapun contoh pemakaian fungsi ini dalam bentuk macro bisa anda lihat sebagai berikut:

```
WherePos    MACRO  X,Y
            MOV    AH,03
            MOV    BH,0
            MOV    X,DL
            MOV    Y,DH
            ENDM
```

18.6. MEMBUAT MENU SOROT

Dewasa ini, menu-menu yang disajikan oleh program yang besar hampir semuanya dalam bentuk menu sorot. Kini dengan sedikit pengetahuan mengenai memory layar akan kita buat suatu menu sorot yang sederhana. Menu ini bisa dikembangkan atau digunakan untuk program yang anda buat.

```
Cls         MACRO              ; Macro untuk menghapus layar
            MOV    AX,0600h
            XOR    CX,CX
            MOV    DX,184Fh
            MOV    BH,10       ; Atribut Hijau diatas hitam
            INT    10h
            ENDM

GotoXY      MACRO  X,Y         ; Macro untuk memindahkan kursor
            MOV    AH,02
            XOR    BX,BX
            MOV    DH,Y
            MOV    DL,X
            INT    10h
            ENDM

SimpanL     MACRO              ; Macro untuk menyimpan seluruh
            LOCAL  Ulang       ; isi layar monitor
            MOV    AX,0B800h
            MOV    ES,AX
            MOV    CX,4000
            XOR    BX,BX
   Ulang:
            MOV    AL,ES:[BX]
```

```
          MOV     Layar[BX],AL
          INC     BX
          LOOP    Ulang
          ENDM

BalikL    MACRO               ; Macro untuk mengembalikan semua
          LOCAL   Ulang       ; isi layar yang telah disimpan
          MOV     CX,4000
          XOR     BX,BX
   Ulang:
          MOV     AL,Layar[BX]
          MOV     ES:[BX],AL
          INC     BX
          LOOP    Ulang
          ENDM

Sorot     MACRO   X,Y         ; Macro untuk membuat sorotan
          LOCAL   Ulang       ; pada menu

          MOV     BL,Y
          MOV     AL,160
          MUL     BL
          MOV     BX,AX

          MOV     AL,X
          MOV     AH,2
          MUL     AH
          ADD     BX,AX
          INC     BX           ; Alamat warna pada posisi X,Y

          MOV     CX,25        ; Panjangnya sorotan
   Ulang:
          MOV     BYTE PTR ES:[BX],4Fh  ; Atribut sorotan
                                        ; putih diatas merah
          ADD     BX,2
          LOOP    Ulang
          ENDM

Readkey   MACRO               ; Macro untuk membaca masukan dari
          MOV     AH,00       ; keyboard.
          INT     16h         ; hasilnya AH=Extended, AL=ASCII
          ENDM

MenuL     MACRO   String      ; Macro untuk mencetak menu
          MOV     AH,09
          LEA     DX,String
          INT     21h
          ENDM

;/=============================================\;
;           Program : SOROT.ASM                ;
;           Author  : S'to                     ;
;           Fungsi  : Membuat menu sorot untuk ;
;                     digunakan program        ;
;\=============================================/;

          .MODEL SMALL
          .CODE
          ORG     100h

TData:    JMP     Proses
          Layar   DB 4000 DUP (?)
          Menu    DB  9,9,'+============================+',13,10
                  DB  9,9,'|      »»» MENU SOROT «««     |',13,10
                  DB  9,9,'+============================+',13,10
                  DB  9,9,'|                            |',13,10
                  DB  9,9,'|     1. Pilihan pertama     |',13,10
```

```
                DB  9,9,'|     2. Pilihan Kedua         |',13,10
                DB  9,9,'|     3. Pilihan Ketiga         |',13,10
                DB  9,9,'|     4. Pilihan Keempat        |',13,10
                DB  9,9,'|                               |',13,10
                DB  9,9,'+===============================+$'
          PosX  DB  22    ; Posisi kolom mula-mula
          PosY  DB  12    ; Posisi baris mula-mula
   Panah_Atas   EQU 72    ; Kode tombol panah atas
  Panah_Bawah   EQU 80    ; Kode tombolpanah bawah
       TEnter   EQU 0Dh   ; Kode tombol Enter

Proses :
          Cls                   ; Hapus layar
          GotoXY  0 8           ; kursor = 0,8
          MenuL   Menu          ; Gambar menu
          SimpanL               ; Simpan isi layar
 Ulang :
          BalikL                ; Tampilkan isi layar yang
                                ; disimpan
          Sorot   PosX,PosY     ; Sorot posisi X,Y
Masukan:
          Readkey                ; Baca masukan dari keyboard
          CMP     AH,Panah_Bawah ; Panah bawah yang ditekan ?
          JE      Bawah          ; Ya! lompat bawah

          CMP     AH,Panah_Atas  ; Panah atas yang ditekan ?
          JE      CekY           ; Ya, lompat CekY

          CMP     AL,TEnter      ; Tombol enter yang ditekan ?
          JNE     Masukan        ; Bukan, lompat ke ulangi
          JMP     Selesai        ; Ya, lompat ke selesai
  CekY :
          CMP     PosY,12        ; Apakah sorotan paling atas ?
          JE      MaxY           ; Ya! lompat ke MaxY
          DEC     PosY           ; Sorotkan ke atas
          JMP     Ulang          ; Lompat ke ulang
  MaxY :
          MOV    PosY,15         ; PosY=Sorotan paling bawah
          JMP    Ulang           ; lompat ke ulang
 Bawah :
          CMP    PosY,15         ; apakah sorotan paling bawah ?
          JE     NolY            ; Ya! lompat ke NolY
          INC    PosY            ; Sorotkan ke bawah
          JMP    Ulang           ; Lompat ke ulang
  NolY :
          MOV    PosY,12         ; Sorotan paling atas
          JMP    Ulang           ; Lompat ke ulang
Selesai:
          INT    20h
END       TData
```

Program 18.4. Membuat Menu Sorot

Bila program 18.4. dijalankan, maka anda akan mendapatkan suatu menu sorot yang menarik, seperti pada gambar 18.4.

<<< Gbr184.PIX >>>

Gambar 18.4. Hasil eksekusi program 18.4.

18.7. HALAMAN LAYAR

Telah kita bahas bahwa pada normalnya satu layar akan menggunakan 4000 byte memory. Tetapi memory yang disediakan untuk layar ini sebenarnya malah beberapa kali lipat lebih banyak dari 4000 byte, karenanya terciptalah apa yang dinamakan 'paging' atau halaman tampilan layar.

Banyaknya halaman tampilan ini sangat bervariasi karena tergantung jumlah memory yang tersedia dan jumlah memory yang digunakan oleh satu halaman tampilan. Untuk alamat awal dari masing-masing halaman tampilan bisa anda lihat pada gambar 18.5.

Halaman	80 X 25
0	B800:0000h
1	B800:1000h
2	B800:2000h
3	B800:3000h
4	B800:4000h*
5	B800:5000h*
6	B800:6000h*
7	B800:7000h*

Ket : * tidak berlaku pada CGA

Gambar 18.5. Alamat Awal Halaman Tampilan

Untuk mengakses memory halaman tampilan yang lain pada modus teks, rumus yang telah kita buat terdahulu bisa anda perbaharui menjadi:

Offset Karakter= (Baris * 160)+(Kolom * 2) + (Halaman*1000h)

Offset Atribut = (Baris * 160)+(Kolom * 2)+1+(Halaman*1000h)

18.8. MERUBAH HALAMAN TAMPILAN

Secara default halaman tampilan yang digunakan adalah halaman tampilan ke 0, yang beralamat awal pada B800:0000h. Untuk merubah halaman tampilan yang aktif ini bisa anda gunakan servis 5 dari interupsi 10h. Adapun aturan pemakaian servis ini adalah:

INPUT:

AH = 5

AL = Nomor halaman tampilan yang akan diaktifkan

```
Delay   MACRO Rep            ; Macro ini untuk menunda program
        LOCAL Ulang
        PUSH  CX
        MOV   DX,Rep
        SUB   CX,CX
 Ulang:
        LOOP  Ulang
        DEC   DX
        CMP   DX,0
        JNZ   Ulang
        POP   CX
        ENDM

Ak_Page MACRO No             ; Macro ini digunakan untuk
        MOV   AH,5           ; mengaktifkan halaman layar
        MOV   AL,No
        INT   10h
        ENDM

;/=============================================\;
;           Program : PAGE.ASM                 ;
;           Author  : S'to                     ;
;           Fungsi  : Untuk mengaktifkan halaman;
;                     layar tertentu           ;
;\=============================================/;

        .MODEL SMALL
        .CODE
        ORG 100h

TData : JMP   Proses
        Kal0  DB 'INI ADALAH HALAMAN TAMPILAN KE 2 ',13,10
              DB ' DENGAN ALAMAT AWAL B800:1000h !!! $'
Proses:
        Ak_Page 2          ; Aktifkan halaman layar yang ke 2
        MOV   AH,09        ;
        LEA   DX,Kal0      ; Tulis kalimat pada halaman ke 2
        INT   21h          ;

        MOV   CX,3         ; Banyaknya pengulangan
 Ulang:
        Ak_Page 2          ; Aktifkan halaman ke 2
        Delay   100
        Ak_Page 0          ; Aktifkan halaman ke 0
        Delay   100
        LOOP    Ulang

        INT   20h
                                END     Tdata
```

Program 18.5. Halaman Layar

Bila program 18.5. anda jalankan, maka dapat anda lihat perpindahan halaman aktif dari halaman tampilan 0 (default DOS) dan halaman tampilan 2.

Catatan:

Bila anda melakukan CLS dengan DOS, maka hanya halaman tampilan aktif yang akan terhapus, sedangkan data pada halaman tampilan yang lain akan tetap.

18.9. MERUBAH BENTUK KARAKTER

Pada modus teks, karakter-karakter tersusun atas titik-titik yang disebut sebagai pixel. Pixel-pixel yang membentuk karakter- karakter ini disimpan dalam tabel. pada EGA terdapat 4 buah tabel, sedangkan pada VGA terdapat 8 buah tabel karakter(Masing- masing 8 KB).

Karakter-karakter yang ditampilkan pada layar monitor diambil dari tabel-tabel yang membentuk karakter ini. Secara default tabel yang akan digunakan adalah tabel ke nol(0).

Bila monitor anda adalah monitor EGA keatas, maka bentuk karakter bisa diubah dengan mengubah isi dari tabel yang menyusun karakter-karakter ini. Untuk itu BIOS telah menyediakan interupsi 10h, service 11h, subservis 00 untuk keperluan ini. Adapun aturan dari pemakaiannya adalah:

INPUT:

AH = 11h
AL = 00h
CX = Jumlah bentuk karakter yang akan diganti
DX = Kode ASCII karakter awal yang akan diganti
BL = Nomor tabel karakter yang diubah
BH = Jumlah byte perkarakter
ES:BP = Alamat buffer pola karakter

```
;/=====================================================\;
;           Program  : MAP.ASM                        ;
;           Author   : S'to                           ;
;           Fungsi   : Untuk merubah bentuk karakter  ;
;                      yang biasa digunakan.          ;
;                      Huruf 'A', akan diubah bentuknya ;
;                      menjadi berbentuk pedang !     ;
;\=====================================================/;

         .MODEL SMALL
         .CODE
          ORG  100h

TData  : JMP  Proses
         Tabel  DB 00011000b       ;    __
                DB 00011000b       ;    __
                DB 10011001b       ; _  __  _
                DB 11111111b       ; ________
                DB 10011001b       ; _  __  _
                DB 00011000b       ;    __
                DB 00011000b       ;    __
                DB 00011000b       ;    __
                DB 00011000b       ;    __
                DB 00011000b       ;    __
                DB 00011000b       ;    __
                DB 00011000b       ;    __
                DB 00011000b       ;    __
                DB 00011000b       ;    __
                DB 00011000b       ;    __
                DB 00001000b       ;     _

Proses :
         MOV   AX,1100h ; Servis
         MOV   DX,'A'   ; Karakter ASCII awal yang akan diganti
         MOV   CX,1     ; Banyaknya karakter yang akan diganti
         MOV   BL,0     ; Nomor blok pemuatan karakter
```

```
          MOV   BH,16    ; Jumlah byte perkarakter
          LEA   BP,Tabel ; Lokasi tabel
          INT   10h

          INT   20h
END       TData
```

Program 18.6. Merubah bentuk karakter

Bila program 18.6. dijalankan, maka semua karakter "A" akan akan segera berubah bentuknya menjadi berbentuk pedang(gambar 18.6.).

<<<< Gbr186.PIX >>>>

Gambar 18.6. Hasil eksekusi program 18.6.

Huruf-huruf yang digunakan akan kembali normal, bila dilakukan pergantian mode. Cobalah anda buat sebuah program yang akan mengganti mode layar dan lihatlah hasil yang akan terjadi setelah membaca bagian 18.10 dibawah ini.

18.10. MODE LAYAR

Suatu subsistem video bisa memiliki lebih dari satu mode video, tetapi hanya satu mode yang dapat aktif pada satu saat. Banyaknya mode video yang terdapat pada suatu jenis subsistem tergantung pada adapter yang dipakai. Makin canggih adapter yang dipakai, makin banyak pula mode video yang didukungnya. Untuk lebih jelasnya mengenai mode video ini dapat dilihat pada gambar 18.7.

Mode	Teks/ Grafik	Jumlah Warna/ Mono	Resolusi	Sistem Video	Jumlah Halaman Tampilan
00h	T	Gray	40X 25	CMEV	8
01h	T	16	40X 25	CMEV	8
02h	T	Gray	80X 25	CMEV	8
03h	T	16	80X 25	CMEV	8
04h	G	4	320X200	CMEV	1

05h	G	Gray	320X200	CMEV	1
06h	G	2	640X200	CMEV	1
07h	T	Mono	80X 25	DEV	8
0Dh	G	16	320X200	EV	8
0Eh	G	16	640X200	EV	4
0Fh	G	Mono	640X350	EV	2
10h	G	16	640X350	EV	2
11h	G	Mono	640X480	MV	1
12h	G	16	640X480	V	1
13h	G	256	320X200	MV	1

Ket. C = CGA (Color Graphics Adapter)
M = MCGA (Memory Controller Card Array)
D = MDA (Monochrome Display Adapter)
E = EGA (Enchanced Graphics Adapter)
V = VGA (Video Graphics Array)

Gambar 18.7. Tabel Mode Video

Setelah melihat tabel mode video, mungkin ada yang heran mengapa setelah mode 07h langsung ke mode 0Dh. Sebenarnya mode- mode di antara kedua mode ini ada, tetapi hanya untuk perangkat video khusus seperti untuk keperluan PCjr yang sudah jarang. Mode video umum dari default DOS adalah mode 03h yaitu teks 80X25.

Dalam inisialisasi mode video ini kita akan menggunakan 2 service dari interupt 10h, yaitu untuk mendapatkan mode video aktif dan merubah mode video.

Untuk mendapatkan mode video aktif gunakan service 0Fh. Setelah pemanggilan service ini, maka beberapa nilai akan dikembalikan ke register yaitu :

- AL = mode video aktif (lihat tabel mode video)
- AH = jumlah karakter per kolom
- BH = halaman tampilan aktif

Halaman tampilan aktif di sini maksudnya adalah nomor dari halaman yang sedang ditampilkan. Jumlah halaman yang terdapat pada suatu sistem video tergantung pada jumlah memori video yang tersedia.

Sedangkan service 00h digunakan untuk merubah mode video. Seperti biasa register AH akan berisi nomor service yang dalam hal ini adalah 00h. Sedangkan mode yang diinginkan diletakkan dalam register AL.

```
SetMode  MACRO  Mode
         MOV    AH,00
         MOV    AL,Mode
         INT    10h
         ENDM
```

Setiap kali dilakukan perubahan pada mode video, maka otomatis memori video juga akan dikosongkan dan sebagai akibatnya layar juga akan dibersihkan.

TIP:

Karena setiap kali terjadi pergantian mode layar akan dibersihkan, anda bisa memanfaatkannya untuk membersihkan layar. Misalkan pada modus Teks default(03), dengan mengaktifkan mode 03 juga, maka isi dari layar akan langsung terhapus.

Bila kita tidak menginginkan terjadinya efek pembersihan layar ini, maka nomor mode video pada AL harus dijumlahkan dengan 128 atau dengan kata lain bit ke-7 pada AL dihidupkan. Dengan cara ini maka isi layar yang lama tidak akan hilang setelah perubahan mode.

BAB XIX

OPERASI PADA STRING

19.1. INTRUKSI PADA STRING

Apa itu string ? String adalah suatu jenis data yang terdiri atas kumpulan karakter, angka, maupun simbol. Pada operasi string register SI dan DI memegang suatu peranan yang khusus. Register SI(Source Index) digunakan untuk mencatat alamat dari sumber string yang akan dimanipulasi sedangkan register DI(Destination Index) digunakan untuk mencatat alamat atau tempat hasil dari manipulasi string. Operasi pada string secara lengkap bisa anda lihat pada tabel 19.1.

INTRUKSI	ARTI
CLD	Clear Direction Flag
STD	Set Direction Flag
CMPS	Compare String
CMPSB	Compare String 1 Byte
CMPSW	Compare String 1 Word
CMPSD	Compare String 1 Double Word <80386 & 80486>
LODS	Load String
LODSB	Load String 1 Byte To AL
LODSW	Load String 1 Word To AX
LODSD	Load String 1 Double Word To EAX <80386 & 80486>
MOVS	Move String
MOVSB	Move String 1 Byte
MOVSW	Move String 1 Word
MOVSD	Move String 1 Double Word <80386 & 80486 >
REP	Repeat
REPE	Repeat If Equal
REPZ	Repeat If Zero
REPNE	Repeat If Not Equal
REPNZ	Repeat If Not Zero

SCAS	Scan String
SCASB	Scan String 1 Byte
SCASW	Scan String 1 Word
SCASD	Scan String 1 Double Word <80386 & 80486>
STOS	Store String
STOSB	Store AL at ES:DI String
STOSW	Store AX at ES:DI String
STOSD	Store EAX at ES:DI String <80386 & 80486>

Gambar 19.1. Perintah Untuk Operasi String

19.2. PENGCOPYAN DAN ARAH PROSES OPERASI STRING

Sama halnya dengan perintah MOV, pada string digunakan perintah MOVS(Move String) untuk mengcopy data dari DS:SI menuju ES:DI. Pasangan DS:SI mencatat alamat dari sumber string sedangkan ES:DI mencatat alamat hasil dari operasi string.

Setiap kali terjadi operasi string(MOVS) maka register SI dan DI akan berkurang atau bertambah sesuai dengan direction flag. Anda bisa menaikkan nilai SI dan DI pada setiap proses dengan perintah CLD(Clear Direction Flag) dan STD(Set Direction Flag) untuk menurunkan nilai SI dan DI pada setiap proses. Pada saat program dijalankan, secara otomatis direction flag akan menunjuk pada proses menaik.

```
;/=======================================\;
;        PROGRAM : STRING1.ASM            ;
;        AUTHOR  : S'to                   ;
;\=======================================/;

       .MODEL SMALL
       .CODE
       ORG 100h

TData : JMP      Proses
        Kalimat  DB 'Donald Duck$' ; 12 karakter
        Buffer   DB 12 DUP(?)
Proses:
        LEA      SI,Kalimat        ; SI = sumber
        LEA      DI,Buffer         ; DI = tujuan
        CLD                        ; Arah proses menaik
        MOV      CX,18             ; Banyaknya pengulangan
Ulang :
        MOVS     ES:Buffer,Kalimat ; Pindahkan data pada
```

```
        LOOP    Ulang           ; DS:SI ke ES:DI

        MOV     AH,09           ;
        LEA     DX,Buffer       ;
        INT     21h             ; Cetak data pada buffer

        INT     20h
END     TData
```

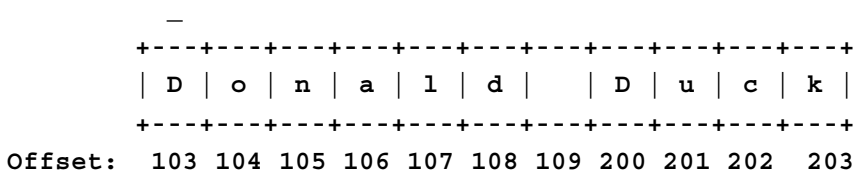
Program 19.1. Penggunaan perintah MOVS

Pada program 19.1. dapat anda lihat bagaimana proses pengcopyan data 'Kalimat' ke 'buffer'. Bila program 19.1. dijalankan maka dilayar akan ditampilkan:

Donald Duck

Hasil yang tercetak merupakan data pada buffer yang diambil pada variabel 'kalimat'. Perintah CLD digunakan untuk memastikan supaya arah proses menaik(SI dan DI ditambah setiap kali operasi).

(STD)Menurun <--- DS:SI ---> Menaik(CLD)



Karena sumber(kalimat) dan tujuan(buffer) pada program 19.1. digunakan tipe data byte(DB) maka oleh assembler perintah MOVS akan dijadikan **MOVSB**(Move string byte), sehingga register SI dan DI setiap kali proses akan ditambah dengan 1. Bila sumber dan tujuan didefinisikan dengan DW, maka assembler akan menjadikannya **MOVSW**(Move string word), dan setiap kali operasi SI dan DI akan ditambah dengan 2.

Selain dengan perintah MOVS, anda bisa juga langsung menggunakan perintah MOVSB atau MOVSW. Mungkin ada yang bertanya-tanya, mengapa kita harus menggunakan MOVSB atau MOVSW, jika dengan perintah MOVS assembler akan merubahnya secara otomatis. Bila anda menggunakan perintah MOVSB atau MOVSW secara langsung maka hal ini akan membantu assembler karena ia tidak perlu lagi menterjemahkannya, selain itu program akan lebih efisien.

Intruksi MOVSB dan MOVSW tidak memerlukan operand, oleh karena itu bila pada program 19.1. ingin anda rubah dengan MOVSB, maka pada perintah:

```
MOVS    ES:Buffer,Kalimat
```

Bisa anda ganti menjadi:

```
MOVSB
```

19.3. PENGULANGAN PADA STRING

Pada program 19.1. kita masih menggunakan pengulangan yang primitif.

Sebenarnya untuk operasi string ini assembler telah menyediakan beberapa pengulangan khusus, yaitu:

-**REP** <Repeat> : Melakukan pengulangan suatu operasi string sebanyak CX kali(register CX akan dikurangi 1 secara otomatis). Ini merupakan bentuk pengulangan tanpa syarat yang akan melakukan pengulangan terus sampai CX mencapai 0.
-**REPE** <Repeat If Equal> : Melakukan pengulangan operasi string sebanyak CX kali atau bila sampai terdapat ketidaksamaan pada kedua operand yang membuat zero flag menjadi tidak aktif(ZF=0).
-**REPZ** <Repeat If Zero> : Perintah ini sama dengan REPE.
-**REPNE** <Repeat If Not Equal> : Melakukan pengulangan operasi string sebanyak CX kali atau bila sampai terdapat kesamaan pada kedua operand yang membuat zero flag menjadi aktif(ZF=1).
-**REPNZ** <Repeat If Not Zero> : Perintah ini sama dengan REPNE.
Perhatikanlah:

Anda hanya bisa menggunakan bentuk pengulangan string bersyarat(REPE,REPZ,REPNE,REPNZ) ini disertai dengan perintah CMPS dan SCAS. Hal ini dikarenakan hanya CMPS dan SCAS yang mempengaruhi zero flag.

Bila pada program 19.1. digunakan perulangan string, maka hasilnya akan menjadi seperti program 19.2.

```
;/=======================================\;
;        PROGRAM : STRING2.ASM            ;
;        AUTHOR  : S'to                   ;
;\=======================================/;

       .MODEL SMALL
       .CODE
       ORG 100h

TData : JMP      Proses
        Kalimat  DB 'Donald Duck$' ; 12 karakter
        Buffer   DB 12 DUP(?)
Proses:
        LEA      SI,Kalimat          ; SI = sumber
        LEA      DI,Buffer           ; DI = tujuan
        CLD                          ; Arah proses menaik
        MOV      CX,18               ; Banyaknya pengulangan

  REP   MOVS     ES:Buffer,Kalimat   ; Pindahkan data
                                     ; 'kalimat' ke 'Buffer'
        MOV      AH,09               ;
        LEA      DX,Buffer           ;
        INT      21h                 ; Cetak Data pada Buffer

        INT      20h
END     TData
```

Program 19.2. Penggunaan perintah REP

19.3. PERBANDINGAN PADA STRING

Pada dasarnya perbandingan string sama dengan pengcopyan string. Pada perbandingan string juga terdapat bentuk CMPS yang dapat berupa

CMPSB(perbandingan byte), CMPSW(perbandingan word) dan CMPSD(perbandingan double word pada 80386 keatas).

Pada string, perbandingan akan dilakukan pada lokasi memory DS:SI dan ES:DI. Perbandingan bisa dilakukan perByte, PerWord atau perDouble Word(Untuk 80386 keatas).

```
Cetak_Klm  MACRO  Kal
           MOV    AH,09
           LEA    DX,Kal     ; Macro untuk mencetak kalimat
           INT    21h
           ENDM

;/=====================================\;
;       PROGRAM : CMPS.ASM               ;
;       AUTHOR  : S'to                   ;
;       FUNGSI  : Menggunakan perbandingan;
;                 pada string            ;
;\=====================================/;

      .MODEL  SMALL
      .CODE
      ORG 100h

TData:  JMP   Proses
        Kal1   DB 'akjsdfhakjvhdf'
        Kal2   DB 'akjsdfhakPvhdf'
        Pesan1 DB 'Kedua kalimat yang dibandingkan sama ! $'
        Pesan2 DB 'Kedua kalimat yang dibandingkan tidak sama !$'
Proses :
        LEA    SI,Kal1
        LEA    DI,Kal2
        CLD                ; Arah proses menaik
        MOV    CX,14       ; Banyaknya perbandingan dilakukan
Ulang  :
        REP    CMPSB       ; Bandingkan selama sama
        JNE    TdkSama     ; Jika tidak sama, lompat ke TdkSama
        Cetak_Klm Pesan1   ; Cetak pesan tidak sama
        JMP    EXIT        ; Selesai
TdkSama:
        Cetak_Klm Pesan2   ; Cetak pesan sama
EXIT   :
        INT    20h
END     TData
```

Program 19.3. Perbandingan String

Bila program 19.3. dijalankan, maka pada layar akan ditampilkan:

Kedua kalimat yang dibandingkan tidak sama !

Perlu anda perhatikan, bahwa perbandingan akan dilakukan sebanyak 14 kali(Nilai CX) atau terdapat ketidak-samaan pada kedua lokasi memory. Bila ditemukan adanya ketidak samaan, perbandingan akan selesai dilakukan dan register SI dan DI tetap ditambah dengan satu, sehingga akan menunjuk pada karakter selanjutnya(sesudah karakter yang tidak sama, pada contoh 19.3. berupa karakter "v").

19.4. OPERASI SCAN PADA STRING

Operasi scan pada string digunakan untuk membandingkan nilai pada register AL, AX atau EAX(80386) dengan data pada ES:DI. Adapun syntax pemakaian SCAN ini adalah:

SCANS Operand

Sama halnya dengan operasi pada string lainnya, bila digunakan perintah diatas, assembler masih akan menerjemahkannya dalam bentuk SCASB(perbandingan AL dengan ES:DI), SCASW(perbandingan AX dengan ES:DI) atau SCASD(perbandingan EAX dengan ES:DI) yang tidak memerlukan operand.

```
Cetak_Klm   MACRO   Kal
            MOV     AH,09       ;
            LEA     DX,Kal      ;
            INT     21h         ; Macro untuk mencetak kalimat
            ENDM

;/=========================================\;
;         Program : SCAN.ASM                ;
;         Author  : S'to                    ;
;         Fungsi  : Melihat proses pencarian ;
;                   string (Scan)           ;
;\=========================================/;
      .MODEL SMALL
      .CODE
      ORG 100h

TData : JMP     Proses
        Cari    DB 'akddtiuerndfalDfhdadfbn'   ; 24 buah karakter
        Ketemu  DB ' Karakter ''s''yang dicari ketemu ! $'
        Tidak   DB ' Karakter ''s'' yang dicari tidak ketemu ! $'
Proses:
        LEA     DI,Cari         ; Lokasi dari string yang diScan
        MOV     AL,'s'          ; Karakter yang dicari
        MOV     CX,24           ; Banyaknya proses Scan
        REPNE   SCASB           ; Scan sebanyak CX atau sampai ZF=1
        JNZ     Tdk_Ada         ; Jika tidak ketemu, maka lompat!
        Cetak_Klm   Ketemu      ; Cetak ketemu
        JMP     Exit            ; Habis
Tdk_Ada:
        Cetak_Klm   Tidak       ; Cetak tidak ketemu

EXIT  : INT      20h            ; Selesai
END     TData
```

Program 19.4. Operasi Scan pada String

Bila program 19.4. dijalankan, maka pada layar akan ditampilkan:

Karakter 's' yang dicari tidak ketemu !

19.5. MENGAMBIL STRING

LODS merupakan bentuk umum untuk mengambil string dari lokasi memory DS:[SI] menuju AL, AX atau EAX. Sama halnya dengan operasi string lainnya, LODS juga akan diterjemahkan oleh assembler ke dalam bentuk LODSB(DS:[SI] ke

AL), LOSW(DS:[SI] ke AX) atau LODSD(DS:[SI] ke EAX<80386>).

19.6. MENGISI STRING

STOS merupakan bentuk umum untuk mengisi string dari AL,AX atau EAX menuju ES:[DI]. Sama halnya dengan operasi string lainnya, STOS juga akan diterjemahkan oleh assembler ke dalam bentuk STOSB(AL ke ES:[DI]), STOSW(AX ke ES:[DI]) atau STOSD(EAX ke ES:[DI]).

Jika Direction flag bernilai 0(dengan CLD) maka setelah intruksi STOS dijalankan register DI akan ditambah secara otomatis, sebaliknya jika Direction flag bernilai 1(dengan STD) maka register DI akan dikurang secara otomatis. Anda bisa menggunakan intruksi pengulangan pada string disertai dengan perintah STOS ini.

BAB XX

MENCETAK ANGKA

20.1. MASALAH DALAM MENCETAK ANGKA

Pada assembler, untuk mencetak suatu angka tidaklah semudah mencetak angka pada bahasa tingkat tinggi. Hal ini dikarenakan baik oleh BIOS maupun DOS tidak disediakan fungsinya. Misalkan kita mempunyai suatu angka 7, untuk mencetaknya kita harus menerjemahkan ke dalam kode ASCII 55 dahulu barulah mencetaknya. Demikian halnya bila ingin mencetak angka 127, maka kita juga harus menterjemahkannya dalam kode ASCII 49, 50 dan 55 untuk kemudian dicetak.

Selanjutnya akan kita lihat, bagaimana caranya untuk mencetak angka dalam bentuk desimal maupun hexadesimal.

20.2. MENCETAK ANGKA DALAM BENTUK DESIMAL

Cara yang paling banyak dilakukan oleh programmer assembler, untuk mencetak angka dalam bentuk desimal adalah dengan membagi angka tersebut dengan 10. Kemudian sisa pembagiannya disimpan dalam stack. Pada saat pencetakan, angka-angka yang disimpan dalam stack akan diambil satu persatu untuk dicetak.

Misalkan anda mempunyai angka 345, maka hasil pembagian dengan 10 sebanyak 3 kali akan menghasilkan sisa berturut-turut 5, 4 dan 3. Sisa pembagian ini kemudian disimpan pada stack. Karena sifat stack yang LIFO <Last In First Out>, maka pada saat pengambilan angka pada stack untuk dicetak akan diambil berturut-turut angka 345 !.

```
;/=========================================\;
;       Program : CD-ANGKA.ASM                ;
;       Author  : S'to                        ;
;       Fungsi  : Mencetak angka yang bernilai ;
;                 antara 0 sampai 65535 dalam  ;
;                 format desimal               ;
;\=========================================/;

         .MODEL SMALL
         .CODE
         ORG 100h
TData :
         JMP   Proses
         Test_Angka  DW 65535    ; Angka yang akan dicetak
Proses:
         MOV   AX,Test_Angka     ; AX = angka yang akan dicetak
         MOV   BX,10             ; BX = penyebut
         XOR   CX,CX             ; CX = 0
Ulang :
         XOR   DX,DX     ; Cegah sisa bagi menjadi pembilang !
         DIV   BX        ; Bagi angka yang akan dicetak dengan 10
         PUSH  DX        ; Simpan sisa bagi dalam stack
```

```
        INC   CX        ; CX ditambah 1
        CMP   AX,0      ; Apakah hasil bagi sudah habis ?
        JNE   Ulang     ; Jika belum, ulangi lagi !
Cetak :
        POP   DX          ; Ambil 1 angka yang disimpan
        ADD   DL,'0'      ; Ubah angka tersebut dalam kode ASCII
        MOV   AH,02       ;
        INT   21h         ; Cetak angka tersebut
        LOOP  Cetak       ; ulangi

        INT   20h
END     TData
```

Program 20.1. Mencetak angka dalam bentuk desimal

Bila program 20.1. dijalankan, maka pada layar akan ditampilkan:

65535

20.3. MENCARI DAN MENAMPILKAN BILANGAN PRIMA

Apa itu bilangan prima? Bilangan prima adalah bilangan yang hanya habis dibagi oleh dirinya sendiri dan 1. Contoh dari bilangan prima ini adalah 2, 3, 5, dan sebagainya.

Secara matematika, untuk mengetest apakah suatu bilangan adalah prima atau bukan, adalah dengan cara pembagian. Misalkan kita ingin mengetahui apakah angka 7 adalah prima atau bukan, kita akan mencoba untuk membaginya dengan 6, 5, 4,..2. Ternyata semua sisa pembagiannya adalah tidak nol atau tidak habis dibagi. Sebagai kesimpulannya, angka 7 adalah prima.

Pada program 20.2. akan anda lihat bagaimana mencari dan menapilkan semua angka prima yang terletak antara angka 0 sampai 1000. Kita akan menggunakan program 20.1. untuk menampilkan angka prima yang telah berhasil dicari.

```
Cetak_Klm MACRO Klm         ; Macro untuk mencetak kalimat
          MOV   AH,09
          LEA   DX,Klm
          INT   21h
          ENDM

CDesimal MACRO  Angka
         LOCAL Ulang, Cetak
         MOV   AX,Angka     ; AX = angka yang akan dicetak
         MOV   BX,10        ; BX = penyebut
         XOR   CX,CX        ; CX = 0
Ulang :
         XOR   DX,DX    ; Cegah sisa bagi menjadi pembilang !
         DIV   BX       ; Bagi angka yang akan dicetak dengan 10
         PUSH  DX       ; Simpan sisa bagi dalam stack
         INC   CX       ; CX ditambah 1
         CMP   AX,0     ; Apakah hasil bagi sudah habis ?
         JNE   Ulang    ; Jika belum, ulangi lagi !
Cetak :
         POP   DX          ; Ambil 1 angka yang disimpan
```

```
          ADD   DL,'0'      ; Ubah 1 angka dalam kode ASCII
          MOV   AH,02       ;
          INT   21h         ; Cetak angka tersebut
          LOOP  Cetak       ; ulangi
          ENDM


;/=============================================\;
;          Program : PRIMA.ASM                 ;
;          Author  : S'to                      ;
;          Fungsi  : Mencari dan menampilkan angka ;
;                    prima dari 0 sampai 1000      ;
;\=============================================/;

          .MODEL SMALL
          .CODE
          ORG 100h

TData :JMP     Awal
       Batas   DW 1000
       Prima   DW 0
       I       DW 2
       J       DW 2
       Spasi   DB '  $'
       Header  DB 9,9,9,'Bilangan Prima 1 sampai 1000 : ',13,10
               DB 9,9,9,'------------------------',13,10,10,'$'

Awal   :
          Cetak_Klm Header
Proses :
          MOV     AX,Batas    ; Jika bilangan yang dicek
          CMP     AX,I        ; sudah sama dengan Batas
          JE      Exit        ; maka selesai
ForI   :
          MOV     J,2         ; J untuk dibagi oleh I
          MOV     Prima,0     ; Prima = Tidak
ForPrima:
          MOV     AX,Prima    ;
          CMP     AX,0        ; Apakah prima = Tidak ?
          JNE     TambahI     ; jika Prima = Ya, lompat ke TambahI
          MOV     AX,I        ;
          CMP     AX,J        ; I = J ?
          JNE     Tidak       ; Jika tidak sama, lompat ke Tidak

          CDesimal I          ; Cetak angka prima
          Cetak_Klm Spasi     ; Cetak spasi
          MOV     Prima,1     ; Prima = Ya
          JMP     TambahJ     ; Lompat ke TambahJ
Tidak  :
          MOV     DX,0       ;
          MOV     AX,I       ;
          MOV     BX,J       ;
          DIV     BX         ; Bagi I dengan J
          CMP     DX,0       ; Apakah sisa bagi=0?
          JNE     TambahJ    ; Jika tidak sama lompat ke TambahJ
          MOV     Prima,1    ; Prima = Ya
TambahJ :
          INC     J           ; Tambah J dengan 1
          JMP     ForPrima    ; Ulangi, bagi I dengan J
TambahI :
          INC     I           ; Tambah I dengan 1
          JMP     Proses      ; Ulangi Cek I = prima atau bukan
Exit   :
          INT     20h
END       TData
```

Program 20.2. Mencari dan menapilkan bilangan prima

Bila program 20.2. dijalankan, maka pada layar akan ditampilkan:

```
        Bilangan Prima 0 sampai 1000 :

        ----------------------------

2  3  5  7  11  13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73

79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163

167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251

257 263 269 271 277 281 283 293 307 311 313 317 331 337 347 349

353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443

449 457 461 463 467 479 487 491 499 503 509 521 523 541 547 557

563 569 571 577 587 593 599 601 607 613 617 619 631 641 643 647

653 659 661 673 677 683 691 701 709 719 727 733 739 743 751 757

761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863

877 881 883 887 907 911 919 929 937 941 947 953 967 971 977 983

991 997
```

Dengan program 20.2. bilangan prima antara 0 sampai 65535 dapat anda ditampilkan.

20.4. MENCETAK ANGKA DALAM BENTUK HEXADESIMAL

Untuk mencetak angka dalam bentuk hexadesimal, adalah lebih mudah daripada mencetak angka delam bentuk desimal. Hal ini dikarenakan sifat dari hexadesimal yang setiap angkanya terdiri atas 4 bit.

Untuk itu anda bisa membuat suatu tabel untuk hexadesimal yang terdiri atas angka 0 sampai F. Kemudian ambillah angka yang ingin dicetak secara 4 bit untuk digunakan sebagai penunjuk dalam mencetak angka tersebut.

```
Cetak    MACRO
         MOV  DL,Tabel_Hex[BX]    ;      MACRO untuk
         MOV  AH,02               ;        mencetak
         INT  21h                 ;  huruf ke BX pada tabel_Hex
         ENDM

;/=====================================================\;
;       Program : CH-ANGKA.ASM                         ;
;       Author  : S'to                                 ;
;       Fungsi  : Mencetak angka yang bernilai  antara ;
;                 0000 sampai 255 <FFh> dalam  format  ;
;                 hexadesimal                          ;
;\=====================================================/;

         .MODEL SMALL
         .CODE
         ORG 100h
TData :
         JMP   Proses
         Tabel_Hex   DB '0123456789ABCDEF'
         Test_Angka  DB 255    ; Angka yang akan dicetak 255=FFh
Proses:
```

```
        SUB   BH,BH              ; Jadikan BH=0
        MOV   BL,Test_Angka      ; BL = angka yang akan dicetak
        PUSH  BX                 ; Simpan angka tersebut

        MOV   CL,4               ; Ambil 4 bit tinggi dari +
        SHR   BL,CL              ; BL untuk dicetak
        Cetak                    ; Cetak 1 angka hexa tingginya

        POP   BX                 ; Ambil angka yang disimpan
        AND   BL,0Fh             ; Ambil 4 bit rendah dari +
        Cetak                    ; BL untuk dicetak

        INT   20h
END     TData
```

Program 20.3. Mencetak angka dalam bentuk hexadesimal

Bila program 20.3. dijalankan, maka pada layar akan tercetak:

FF

BAB XXI

PENGAKSESAN PORT DAN PENGAKTIFAN SPEAKER

21.1. PORT

Port bila diterjemahkan secara bebas, artinya adalah pelabuhan atau terminal, yang merupakan tempat keluar masuk. Pengertian port dalam komputer juga tidak jauh berbeda. Port merupakan tempat komputer berhubungan dengan alat-alat lain(dunia luar).

Untuk periferal yang dihubungkan dengan komputer seperti disk-drive, keyboard, monitor, mouse, printer dan speaker biasanya akan diberi nomor portnya masing-masing. Pengontrolan kerja dari periferal yang dihubungkan dengan komputer biasanya dilakukan melalui portnya.

Oleh IBM, alat-alat yang digunakan pada komputer PC telah diberi nomor portnya masing-masing(Gambar 21.1.)

Peralatan	PC/XT	AT
DMA Controler (8237A-5)	000-00F	000-01F
Inteerupt Controler (8295A)	020-021	020-03F
Timer	040-043	040-05F
PPI 8255A-5	060-063	---
Keyboard (8042)	---	060-06F
RealTime Clock (MC146818)	---	070-07F
DMA Page Register	080-083	080-09F
Interrupt Controler 2(8259A)	---	0A0-0BF
DMA Controller 2 (8237A-5)	---	0C0-0DF
MathCo	---	0F0-0F1
MathCo	---	0F8-0FF
Hard Drive Controler	320-32F	1F0-1F8
Game Port for Joysticks	200-20F	200-207
Expansion Unit	210-217	---
LPT2	---	278-27F
COM2	2F8-2FF	2F8-2FF
Prototype Card	300-31F	300-31F
NetWork Card	---	360-36E
LPT1	378-37F	378-37F
MDA & Parallel Interface	3B0-3BF	3B0-3BF

CGA	3D0-3DF	3D0-3DF
Disk Controller	3F0-3F7	3F0-3F7
COM1	3F8-3FF	3F8-3FF

Gambar 21.1. Tabel Nomor Port

21.2. PENGAKSESAN PORT

Untuk melihat nilai pada suatu port digunakan perintah:

IN Reg,NoPort

"Reg" harus merupakan register AL atau AX sedangkan "NoPort" merupakan nomor port yang akan diakses. Perintah IN akan mengcopykan nilai pada "NoPort" ke dalam "Reg". Perintah IN akan mengcopykan nilai pada port sebanyak 1 byte bila digunakan regiser AL atau 1 word bila digunakan register AX.

Nomor Port yang ingin diakses bisa dituliskan secara langsung jika nomor Port tersebut dibawah 255(FFh). Bila nomor port diatas FFh, maka nomor port tersebut harus dimasukkan ke dalam register DX.

Untuk memasukkan nilai pada suatu port, digunakan perintah:

OUT NoPort,Reg

Perintah OUT ini sama dengan perintah IN hanya perintah OUT digunakan untuk mengirimkan 1 byte(bila Reg=AL) atau 1 word(bila Reg=AX) pada port. Sama halnya dengan perintah IN, bila nomor port yang diakses melebihi 255, harus digunakan register DX. Contoh :

```
IN   AL,60h  ; ambil nilai 1 byte pada port 60h
OUT  60h,AL  ; masukkan nilai AL pada port 60h
```

Pada dasarnya pengaksesan Port sama dengan pengaksesan memory, tetapi harus diingat bahwa alamat Port dan memory adalah lain. Jadi misalkan alamat Port 60h dan alamat memory 60h adalah lain. Walupun keduanya mempunyai nilai yang sama tetapi sebenarnya alamat keduanya tidak ada hubungan sama sekali.

Pengaksesan Port sebenarnya tidaklah sesederhana yang dibayangkan oleh banyak orang. Karena Port berhubungan langsung dengan perangkat keras komputer maka kita harus mengetahui dengan jelas cara mengaksesnya. Ada Port yang cuma bisa ditulisi atau dibaca.

Untuk Port yang mempunyai hubungan dua arah atau untuk baca tulis biasanya mempunyai suatu lagi Port yang digunakan sebagai Port pengontrol. Port pengontrol ini berfungsi untuk mengatur modus dari operasi yang akan dilakukan terhadap alat tersebut. Lebih jauh dari pengaksesan Port ini tidak

bisa penulis ungkapkan disini karena hal tersebut sangat tergantung dari jenis perangkat kerasnya.

21.3. PENGAKTIFAN SPEAKER

Untuk membunyikan speaker, suka atau tidak anda harus mengakses port. Hal ini dikarenakan sampai saat ini tidak adanya fasilitas dari DOS maupun BIOS untuk membunyikan speaker. Pada bagian ini akan kita lihat teknik-teknik pemrograman dari speaker supaya dapat menghasilkan suatu frekwensi atau nada.

21.4. MENGAKTIFKAN SPEAKER SECARA LANGSUNG

Port 61h merupakan suatu I/O port yang digunakan untuk berbagai keperluan, diantaranya oleh speaker pada bit 0 dan 1. Kedua bit ini dihubungkan pada speaker melalui gerbang logika "AND" sehingga untuk mengaktifkan suara pada speaker ini maka bit ke 0 dan 1 dari port 61h ini harus bernilai 1. Ingatlah, pengaktifan pada speaker ini jangan sampai mengganggu bit lainnya.

<<<<< Gbr212.PIX >>>>>>

Gambar 21.2. Skema Rangkaian Speaker pada PC

```
Readkey  MACRO              ; Macro untuk
         MOV  AH,00         ; menunggu masukan dari keyboard
         INT 16h
         ENDM

;/=======================================================\;
;               Program  : SOUND1.ASM                    ;
;               Author   : S'to                          ;
;               Fungsi   : membunyikan speaker dan       ;
;                          mematikannya melalui port 61h ;
;\=======================================================/;

       .MODEL SMALL
       .CODE
       ORG 100h

Proses :
         IN   AL,61h          ; Ambil data port 61h <Speaker>
         OR   AL,00000011b    ; Jadikan Bit ke 0 & 1 menjadi 1
         OUT  61h,AL          ; Bunyikan speaker

         Readkey            ; Menunggu penekanan sembarang tombol

         AND  AL,11111100b    ; Jadikan bit ke 0 & 1 menjadi 0
         OUT  61h,AL          ; Matikan speaker

         INT  20h             ; selesai
END      Proses
```

Program 21.1. Mengaktifkan speaker

Untuk menghasilkan suatu frekwensi yang tepat dengan program 21.1. memang agak sulit, karena frekwensi yang terjadi sangat tergantung dari kecepatan komputer yang bersangkutan. Bila pada komputer anda terdapat tombol TURBO, cobalah ubah-ubah kecepatan dari komputer untuk melihat perubahan dari frekwensi yang dihasilkan.

21.5. PENGONTROLAN FREKWENSI MELALUI TIMER

Untuk menghasilkan suatu frekwensi yang tidak terpengaruh oleh kecepatan komputer, bisa dilakukan dengan memrogram timer <pewaktu> yang digunakan oleh speaker ini. Frekwensi yang dihasilkan dengan menggunakan tetapan waktu akan lebih mudah dihasilkan dan lebih tepat.

PIT <Programmable Interval Timer> merupakan suatu timer yang dapat diprogram. Keluaran dari PIT ini digunakan antara lain oleh detik jam waktu(IRQ0) dan RAM dinamik untuk me-refresh dirinya. Keluaran ketiga (OUT2) dari PIT untuk menghasilkan sinyal gelombang persegi yang digunakan oleh speaker.

Karena frekwensi yang dihasilkan oleh PIT ini dapat diatur melalui software maka dapat dimanfaatkan untuk membentuk nada pada speaker.

Untuk memrogram timer<PIT> ini, pertama-tama kita harus mengirimkan nilai B6h pada port 43h. Pengiriman nilai ini akan menyebabkan port 42h siap untuk menerima nilai 16 bit untuk dijadikan tetapan perhitungan<counter>. Untuk nilai counter yang diberikan pada port 43h ini digunakan rumus :

Counter = 123540h / Hz Hz=<frekwensi ingin dihasilkan>

Hasil dari perhitungan ini kemudian dimasukkan kedalam timer melalui port 42h dan akan disimpan dalam regiser internal 16 bit. Tetapi karena Timer ini hanya mempunyai 8 bit masukan maka kita tidak bisa memasukkan 16 bit sekaligus. Hal ini dapat anda bayangkan sebagai suatu kamar yang dapat menampung 2 orang tetapi pintunya hanya dapat dilalui oleh 1 orang. Untuk itu masukkanlah byte rendahnya terlebih dahulu kemudian masukkan byte tingginya.

Setelah timer diprogram, maka speaker tinggal diaktifkan untuk menghasilkan frekwensi yang sesuai dengan timer. Secara teori frekwensi yang dapat dihasilkan berupa 1 Hz - 1,193 Mhz (bandingkan dengan kemampuan dengar manusia 20 Hz - 20 Khz).

Dengan frekwensi yang tepat, sebenarnya banyak hal yang dapat kita lakukan. Kita dapat saja membuat program Pengusir Nyamuk ataupun program pengusir Burung dan Tikus. Binatang- binatang ini biasanya takut pada frekwensi yang tinggi seperti frekwensi 20 Khz - 40 Khz. Untuk mengusir nyamuk frekwensi 25 Khz sudah memadai, tetapi bila anda ingin mengusir tikus sebaikkan frekwensinya dibuat suatu layangan. Artinya frekwensi yang dihasilkan diubah-ubah antara 20 Khz - 40 Khz.

```
NoPCsound  MACRO
           IN    AL,61h      ; Ambil data Port 61h
```

```
            AND   AL,0FCh     ; Matikan bit ke 6 & 7
            OUT   61h,AL      ; Masukkan nilainya pada Port 61h
            ENDM

PCsound     MACRO  Hz
            MOV    AL,0B6h    ;
            OUT    43h,AL     ;  Persiapkan Timer

            MOV    DX,0012h   ;
            MOV    AX,3540h   ; Bagi 123540H dengan frekwensi
            MOV    BX,Hz      ; yang akan dihasilkan.
            DIV    BX         ; < 123540:Hz > , hasil pada AX

            OUT    42h,AL     ; Masukkan byte rendah dahulu.
            MOV    AL,AH      ; Port hanya dapat melalui AL/AX
            OUT    42h,AL     ; Masukkan byte tingginya.

            IN     AL,61h     ; Ambil data port 61h <Speaker>
            OR     AL,03      ; Jadikan Bit ke 6 & 7 menjadi 1
            OUT    61h,AL     ; Bunyikan speaker
            ENDM

;/=========================================================\;
;               Program  : NYAMUK.ASM                     ;
;               Author   : S'to                           ;
;               Fungsi   : membunyikan speaker dan mengatur ;
;                          frekwensinya melalui Timer.      ;
;                          Frekwensi yang dihasilkan dapat  ;
;                          digunakan untuk mengusir nyamuk  ;
;\=========================================================/;

       .MODEL SMALL
       .CODE
       ORG 100h

Proses :
         PCsound 25000      ; Frekwensi untuk mengusir nyamuk.

         MOV     AH,00
         INT     16h        ; Readkey

         NoPCsound          ; Matikan suara.
         INT  20h           ; selesai
END      Proses
```

Program 21.2. Pengontrolan speaker dan timer

Frekwensi yang dihasilkan pada program 21.2. tidak akan terdengar, oleh karena itu bila anda ingin mendengar suatu frekwensi cobalah ubah nilai 25000 dengan nilai 20 - 20000.

BAB XXII

PROGRAM BERPARAMETER

22.1. APA ITU PARAMETER ?

Program format, copy dan delete dari Dos tentunya sudah tidak asing lagi bagi kita. Misalkan pada program copy, untuk mengcopy suatu file yang bernama SS.ASM pada drive B: menuju drive C: dengan nama TT.ASM dapat kita tuliskan dengan:

```
        Parameter 1 Parameter 2
         +---+----++---+----+
```

COPY B:SS.ASM C:TT.ASM

Yang dimaksud dengan parameter program adalah semua tulisan yang terdapat dibelakang kata copy(B:SS.ASM dan C:TT.ASM). B:SS.ASM dikatakan sebagai parameter pertama sedangkan C:TT.ASM dikatakan sebagai parameter kedua.

22.2. FILE CONTROL BLOCK

Masih ingatkah anda, pada setiap program COM kita selalu menyediakan 100h byte kosong dengan perintah ORG 100h. 100h byte kosong kosong ini dinamakan sebagai PSP atau Program Segment Prefix dan digunakan oleh DOS untuk mengontrol jalannya program kita. Salah satu pengontrolan yang dilakukan, ialah terhadap paramater program.

PSP sebenarnya masih dibagi-bagi lagi menjadi bagian-bagian yang tugasnya berbeda-beda. Salah satu bagian yang mengatur terhadap parameter program adalah yang dinamakan sebagai FCB(File Control Block). FCB ini terdiri atas 2 bagian, yaitu FCB1 dan FCB2.

FCB1 bertugas menampung parameter pertama dari program, dan berada pada offset 5Ch sampai 6Bh(16 Byte). Sedangkan FCB2 bertugas menampung parameter kedua dari program, dan berada pada offset 6Ch sampai 7Bh(16 Byte).

Ingatlah, bila anda menjalankan sebuah program pada prompt Dos maka yang terakhir dimasukkan pastilah karakter Enter(0Dh).

```
Cetak_Klm  MACRO Klm        ; Macro untuk mencetak kalimat
           LEA   DX,Klm
           MOV   AH,09
           INT   21h
           ENDM

Cetak_Kar  MACRO  Kar       ; Macro untuk mencetak karakter
           MOV    DL,Kar
           MOV    AH,02
           INT    21h
```

```
          ENDM

;/=====================================================\;
;          Program : FCB12.ASM                        ;
;          Author  : S'to                             ;
;          Fungsi  : Mencetak isi FCB1 dan FCB2 yang  ;
;                    menampung parameter program. Untuk ;
;                    mencobanya, tambahkanlah parameter ;
;                    pada program saat menjalankannya,  ;
;                    seperti:                           ;
;                        C:\> FCB12  P111  P222         ;
;\=====================================================/;


        .MODEL SMALL
        .CODE
        ORG 100h

TData  : JMP    Proses
         Para1  DB ' Parameter pertama : $'
         Para2  DB 13,10,' Parameter kedua   : $'

Proses : Cetak_Klm Para1    ; Cetak kalimat Para1
         MOV    BX,5Ch      ; Alamat FCB1
         MOV    CX,16
Ulang1 :
         Cetak_Kar [BX]     ; Cetak parameter pertama (FCB1)
         INC    BX
         LOOP   Ulang1

         Cetak_Klm Para2    ; Cetak kalimat Para2
         MOV    CX,16
Ulang2 :
         Cetak_Kar [BX]     ; Cetak parameter kedua (FCB2)
         INC    BX
         LOOP   Ulang2

         INT    20h
END      TData
```

Program 22.1. Mengambil parameter 1 dan 2 program dari FCB

Untuk mencoba dengan program 22.1., masukkanlah parameter program. Hasil eksekusinya akan tampak seperti:

```
C:\> FCB12  F111 F222

Parameter pertama : F111

Parameter kedua   : F222
```

22.3. DATA TRANSFER AREA

Dengan FCB kita hanya mampu mengambil 2 parameter dari program. Bagaimana jika parameter yang dimasukkan lebih dari 2?. Untuk itu anda harus mengakses bagian lain dari PSP yang dinamakan sebagai DTA atau Data Tranfer Area. DTA mencatat semua parameter yang dimasukkan pada program.

DTA terletak mulai pada offset ke 80h sampai FFh dari PSP. Bila anda

memasukkan parameter dalam program, seperti:

C:\> FCB 111

Maka pada DTA, offset ke 80h akan berisi banyaknya karakter yang diketikkan termasuk karakter enter(Dalam contoh=04). Pada offset ke 81h akan berisi spasi(20h), dan pada offset ke 82h berisi karakter pertama parameter program.

```
Cetak_Klm  MACRO Klm        ; Macro untuk mencetak kalimat
           LEA   DX,Klm
           MOV   AH,09
           INT   21h
           ENDM

Cetak_Kar  MACRO  Kar       ; Macro untuk mencetak karakter
           MOV    DL,Kar
           MOV    AH,02
           INT    21h
           ENDM

;/=====================================================\;
;          Program : DTA.ASM                           ;
;          Author  : S'to                              ;
;          Fungsi  : Mencetak isi dari DTA program yang  ;
;                    menampung parameter program. Untuk  ;
;                    mencobanya, tambahkanlah parameter  ;
;                    pada program saat menjalankannya,   ;
;                    seperti:                            ;
;                        C:\> DTA    P111  P222   P333   ;
;\=====================================================/;

        .MODEL SMALL
        .CODE
        ORG 100h

TData  : JMP     Proses
         Para    DB  13,10,' Parameter program : $'
         T_Enter EQU 0Dh
         Spasi   EQU 20h

Proses : Cetak_Klm Para               ; Cetak kalimat Para1
         MOV    BX,81h                ; Alamat DTA
Ulang  :
         INC    BX                    ; BX = Alamat DTA
         CMP    BYTE PTR [BX],T_Enter ; Apakah tombol Enter ?
         JE     Exit                  ; Ya! Lompat ke Exit
         Cetak_Kar [BX]               ; Bukan! Cetak karakter tsb
         CMP    BYTE PTR [BX],Spasi   ; Apakah spasi ?
         JNE    Ulang                 ; Bukan, lompat ke ulang
         Cetak_Klm Para                ; Ya! Cetak kalimat
         JMP    Ulang                 ; Lompat ke ulang
Exit   :
         INT    20h
END      TData
```

Program 22.2. Mengambil parameter program dari DTA

Bila program 22.2. dijalankan seperti:

```
C:\>DTA P111 P222 P333

Parameter program : P111
Parameter program : P222
Parameter program : P333
```

BAB XXIII
OPERASI PADA FILE

23.1. PENANGANAN FILE

Pada Dos versi 1.0 penanganan file dilakukan melalui FCB. System ini muncul, sebagai hasil dari kompabilitas dengan CP/M. Ternyata penanganan file melalui FCB ini banyak kendalanya, seperti kemampuan manampung nama file yang hanya 11 karakter. Karena hanya mampu menampung 11 karakter maka nama untuk directory tidak akan tertampung sehingga Dos versi awal tidak bisa menangani adanya directory. Penanganan file melalui FCB ini sudah ketinggalan zaman dan telah banyak ditinggalkan oleh programmer-programmer. Karena itu maka pada buku ini, kita hanya akan membahasnya sekilas.

Pada Dos versi 2.0 diperkenalkan suatu bentuk penanganan file yang baru. Penanganan file ini dinamakan File Handle yang mirip dengan fungsi pada UNIX. Dengan file handle penanganan terhadap directory dengan mudah dilakukan. Operasi file yang dilakukan dengan file handle harus dibuku(Open) terlebih dahulu, selain itu file handle bekerja dengan apa yang dinamakan dengan ASCIIZ. **ASCIIZ** atau ASCII+Zero byte adalah suatu teknik penulisan string yang diakhiri dengan byte nol(0) atau karakter Null. Contohnya dalam penulisan nama file:

```
Nama DB 'DATA.DAT ',0  <----- ASCIIZ
```

23.2. MEMBUAT FILE BARU

Untuk menciptakan suatu file baru, dapat digunakan fungsi 3Ch dari intrupsi 21h. Adapun aturan dari pemakaian interupsi ini adalah dengan memasukkan nilai servis 3Ch pada AH, pasangan register DS:DX menunjuk pada nama file ASCIIZ yang akan diciptakan, CX diisi dengan atribut file atau maksud dari pembukaan file tersebut dengan spesifikasi nomor Bit:

- 0 untuk File Read Only, yaitu file yang dibuka hanya untuk dibaca.
- 1 untuk File Hidden, yaitu file yang disembunyikan. Jenis file ini tidak akan ditampilkan pada proses DIR dari DOS.
- 2 untuk File System, yaitu file yang akan otomatis dijalankan pada saat BOOT. Jenis file ini biasanya berkaitan erat dengan mesin komputer dan biasanya ditandai dengan ektensi SYS.
- 3 untuk Volume label.
- 4 untuk Nama subdirectory.
- 5 untuk File Archive, yaitu suatu bentuk file normal.

Jika fungsi ini berhasil menciptakan suatu file baru, maka Carry flag akan dijadikan 0(Clear) dan AX akan berisi nomor handle(Nomor pembukaan file). Setiap file yang dibuka akan mempunyai nomor handle yang berbeda-beda, umumnya bernilai 5 keatas. Sebaliknya jika proses penciptaan file gagal, maka Carry flag akan dijadikan 1(Set) dan AX akan berisi kode kesalahan. Adapun kode kesalahan yang sering terjadi adalah:

- 03h artinya Path tidak ditemukan
- 04h artinya Tidak ada handle
- 05h artinya akses terhadap file ditolak

Dalam menggunakan fungsi ini anda harus berhati-hati. Bila pada proses penciptaan file baru dan ternyata file tersebut telah ada, maka fungsi ini akan menjadikan file tersebut menjadi nol. Untuk menghindari hal ini anda bisa menggunakan fungsi 4Eh untuk mengecek apakah file yang akan diciptakan telah ada atau belum.

```
Create  MACRO NamaFile, Attribut, Handle
        PUSH  CX
        PUSH  DX
        MOV   AH,3Ch
        MOV   CX,Attribut
        LEA   DX,NamaFile
        INT   21h
        MOV   Handle,AX
        POP   DX
        POP   CX
        ENDM
```

23.3. MEMBUKA FILE YANG TELAH ADA

Untuk membuka suatu file yang telah ada, digunakan fungsi 3Dh dari interupsi 21h. Adapun aturan dari pemakaiannya adalah:

INPUT:

AH = 3Dh

AL = Tujuan pembukaan file:
- 00 hanya untuk dibaca (Read Only)
- 01 hanya untuk ditulisi (Write Only)
- 02 untuk ditulis dan dibaca (Read/Write)

DS:DX = Nama file dengan ASCIIZ

OUTPUT:

Jika berhasil, maka CF = 0 dan AX = Nomor handle

Jika tidak berhasil, maka CF = 1 dan AX = kode kesalahan

Hati-hatilah:

Pembukaan file yang dilakukan oeh fungsi ini akan memindahkan pointer file pada awal file. Bila anda membuka suatu file dengan fungsi ini dan langsung menulis pada file tersebut, maka isi dari file tersebut akan tertimpa. Untuk menghindari ini pointer file harus dipindahkan pada akhir file sesudah pembukaan. Adapun contoh dari macro untuk membuka file dengan fungsi 3Dh ini adalah:

```
Open  MACRO NamaFile,Attribut,Handle
      PUSH  DX
      MOV   AH,3Dh
      MOV   AL,Attribut
      LEA   DX,NamaFile
      INT   21h
      MOV   Handle,AX
      POP   DX
      ENDM
```

23.4. MENUTUP FILE

Untuk menutup suatu file yang telah dibuka, dapat digunakan fungsi ke 3Eh dari interupsi 21h. Fungsi ini akan menutup file yang dibuka dengan 3Dh. Untuk menggunakannya isilah AH dengan 3Eh dan BX dengan nomor handle file tersebut. Adapun contoh dari macro untuk menutup suatu file yang terbuka adalah:

```
Close    MACRO  Handle
         PUSH BX
         MOV  AH,3Eh
         MOV  BX,Handle
         INT  21h
         POP  BX
         ENDM
```

Sebenarnya tidak semua file yang telah dibuka harus ditutup dengan interupsi khusus. Bila anda mengakhiri program tidak dengan interupsi 20h tetapi dengan fungsi 4Ch dari interupsi 21h, maka penutupan file yang terbuka

sebenarnya tidak perlu. Hal ini dikarenakan interupsi ini akan menutup semua file yang terbuka secara otomatis. Mengenai fungsi 4Ch dari interupsi 21h ini akan kita bahas lebih lanjut nantinya.

Ingatlah:_

Kemampuan Dos dalam menangani file yang terbuka adalah terbatas. Kemampuan Dos dalam menangani file yang terbuka dapat diatur melalui Config.Sys dengan perintah: Files=n.

23.5. MEMBACA FILE

Pembacaan file handle dapat dilakukan melalui fungsi 3Fh dari interupsi 21h. Adapun aturan dari pemakaian fungsi ini adalah:

```
INPUT:
AH    = 3Fh
CX    = Banyaknya data(dalam byte) yang ingin dibaca
BX    = Nomor File handle
DS:DX = Alamat buffer tempat hasil pembacaan akan disimpan
OUTPUT:
Jika berhasil, maka CF = 0 dan AX = Jumlah byte yang telah
dibaca. Bila AX=0 atau AX < CX artinya akhir file telah
dicapai.
Jika tidak berhasil, maka CF = 1 dan AX = kode kesalahan
```

Adapun contoh dari penggunaannya dalam macro adalah:

```
Read  MACRO  Handle,Number,Buffer
      PUSH   BX
      PUSH   CX
      PUSH   DX
      MOV    AH,3Fh
      MOV    BX,Handle
      MOV    CX,Number
      LEA    DX,Buffer
      INT    21h
      POP    DX
      POP    CX
      POP    BX
      ENDM
```

Perhatikanlah:

Fungsi ini akan membaca banyaknya data dari suatu file dari posisi yang tidak tetap, tergantung dari pointer file pada saat itu. Fungsi ini hanya mampu membaca sebesar 1 segment(64 KB) dalam sekali pembacaan. Untuk membaca file yang besar anda bisa membacanya dengan proses looping. pada proses looping, posisi pointer tidak perlu diatur lagi karena setiap kali selesai pembacaan, pointer akan diatur secara otomatis.

23.6. MENULIS PADA FILE

Untuk menulisi file dapat digunakan fungsi 40h dari interupsi 21h. Adapun aturan dari pemakaian fungsi ini adalah:

INPUT:
AH = 40h
BX = Nomor File Handle
CX = Banyaknya data(dalam byte) yang akan dituliskan
DS:DX = Alamat dari data yang akan ditulis ke file

OUTPUT:
Jika berhasil, maka CF = 0 dan AX = Jumlah byte yang berhasil dituliskan.
Jika tidak berhasil, maka CF = 1 dan AX = kode kesalahan

Adapun contoh macro dari penggunaan fungsi ini adalah:

```
Write  MACRO  Handle,Number,Buffer
       PUSH   BX
       PUSH   CX
       PUSH   DX
       MOV    AH,40h
       MOV    BX,Handle
       MOV    CX,Number
       LEA    DX,Buffer
       INT    21h
       POP    DX
       POP    CX
       POP    BX
       ENDM
```

Fungsi ini hampir sama dengan fungsi 3Fh. Dengan melihat pada nilai AX setelah operasi penulisan berhasil, maka dapat diketahui:

- Bila AX = CX, artinya operasi penulisan berhasil dengan sukses.
- Bila AX < CX, artinya penulisan hanya berhasil menulis sebagian dari data yang seharusnya dituliskan.

```
Cetak     MACRO Kal           ; Macro untuk mencetak
          MOV   AH,09         ; kalimat
          LEA   DX,Kal
          INT   21h
          ENDM

;/=============================================\;
;         Program : FCOPY.ASM                  ;
;         Author  : S'to                       ;
;         Fungsi  : Mengcopy file              ;
;\=============================================/;

           .MODEL SMALL
           .CODE
           ORG 100h

TData   :  JMP MULAI
Sumber     DB       'Nama file sumber       : $'
Tujuan     DB 13,10,'Nama file tujuan       : $'
File1      DB 70,?,70 Dup (0)
File2      DB 70,?,70 Dup (0)
Handle1    DW ?
Handle2    DW ?
Good       DB 13,10,'Pengcopyan File telah dilaksanakan ....$'
Err1       DB 13,10,'Salah perintah .....$'
Err2       DB 13,10,'File tidak ditemukan .....$'
Err3       DB 13,10,'Path tidak ditemukan .....$'
Err4       DB 13,10,'File yang dibuka, kebanyakan ...$'
Err6       DB 13,10,'Penggunaan File handle yang salah ..$'
Err15      DB 13,10,'Spesifikasi Drive yang salah ....$'
Err21      DB 13,10,'Drive tidak siap .....$'
Err29      DB 13,10,'Kesalahan penulisan ....$'
Err30      DB 13,10,'Kesalahan pembacaan ....$'
ErrL       DB 13,10,'Kesalahan yang lain .....$'

Mulai    :
           MOV     AX,3              ; Ganti mode
           INT     10H               ; untuk membersihkan layar
 TanyaF1 :
           Cetak   Sumber            ; Cetak kalimat "sumber"

           MOV     AH,0AH            ;
           LEA     DX,File1          ; Tanya nama file yang ingin
           INT     21h               ; dicopy

           LEA     BX,File1          ; Jika user tidak mengetikkan
           INC     BX                ; nama file yang ingin dicopy
           CMP     BYTE PTR [BX],0   ; atau langsung ditekan enter
           JE      TanyaF1           ; maka tanya lagi
 Ulang1 :
           INC     BX                  ;
           CMP     BYTE PTR [BX],0Dh   ;
           JNE     Ulang1              ; Jadikan file "sumber"
           MOV     BYTE PTR [BX],0     ; menjadi ASCIZZ
TanyaF2 :
           Cetak   Tujuan              ; Cetak kalimat "tujuan"
```

```
          MOV     AH,0AH            ; Tanya nama file
          LEA     DX,File2          ; "tujuan" atau nama file
          Int     21h               ; hasil pengcopyan

          LEA     BX,File2          ; Jika user tidak mengetikkan
          INC     BX                ; nama file untuk "tujuan"
          CMP     BYTE PTR [BX],0   ; atau langsung ditekan enter
          JE      TanyaF2           ; maka tanya lagi
Ulang2 :
          INC     BX                     ;
          CMP     BYTE PTR [BX],0Dh      ;
          JNE     Ulang2                 ; Jadikan file Tujuan
          MOV     BYTE PTR [BX],0        ; menjadi ASCIZZ

          MOV     AH,3DH          ; Servis buka file
          MOV     AL,0            ; Mode file Read Only
          LEA     DX,File1 + 2    ; Nama file "sumber"
          INT     21h             ;
          JC      Er              ; Jika error, lompat

          MOV     Handle1,AX      ; AX=nomor handle file "sumber"

          MOV     AH,3CH          ; Buat file baru
          LEA     DX,File2 + 2    ; Dengan nama "tujuan"
          MOV     CX,00100000b    ; File Archive / normal
          Int     21h             ;
          JC      Er              ; Jika error, lompat

          MOV     Handle2,AX      ; AX=nomor handle file "tujuan"
Copy  :
          MOV     AH,3FH          ; servis baca file
          MOV     BX,Handle1      ; Baca file "sumber"
          MOV     CX,1024         ; Banyaknya pembacaan
          LEA     DX,Buffer       ; Lokasi penampungan
          INT     21h             ;
          JC      Er              ; Jika error, lompat

          CMP     AX, 0           ; Pembacaan file sudah EOF?
          JE      Selesai         ; Ya, exit

          MOV     CX,AX             ; Banyaknya data
          MOV     AH,40h            ; Servis tulis file
          MOV     BX,Handle2        ; Tulis file "Tujuan"
          LEA     DX,Buffer         ; Lokasi data
          Int     21h
          JC      Er                ; Jika error, lompat

          CMP     CX,AX             ; File habis dicopy?
          JE      Copy              ; belum, ulangi
Selesai :
          Cetak   Good              ; Pengcopyan selesai

          MOV     AH,3Fh            ; Tutup file
          MOV     BX,Handle1        ; "sumber"
          INT     21h               ;

          MOV     BX,Handle2        ; Tutup file
          INT     21h               ; "Tujuan"
   EXIT :
          INT     20h               ; Selesai
     Er :
          CMP     AX,1
          JNE     NotErr1
          Cetak   Err1
          JMP     EXIT
NotErr1 :
          CMP     AX,2
```

```
            JNE     NotErr2
            Cetak   Err2
            JMP     EXIT
 NotErr2 :
            CMP     AX,3
            JNE     NotErr3
            Cetak   Err3
            JMP     EXIT
 NotErr3 :
            CMP     AX,4
            JNE     NotErr4
            Cetak   Err4
            JMP     EXIT
 NotErr4 :
            CMP     AX,6
            JNE     NotErr6
            Cetak   Err6
            JMP     EXIT
 NotErr6 :
            CMP     AX,21
            JNE     NotErr21
            Cetak   Err21
            JMP     EXIT
 NotErr21 :
            CMP     AX,29
            JNE     NotErr29
            Cetak   Err29
            JMP     EXIT
 NotErr29 :
            CMP     AX,30
            JNE     NotErr30
            Cetak   Err30
            JMP     EXIT
 NotErr30 :
            Cetak   ErrL
            JMP     EXIT

            Buffer  LABEL BYTE
END         TData
```

Program 23.1. Mengcopy File

Bila program 23.1. dijalankan, maka program akan meminta anda memasukan nama file yang akan dicopy dan nama file hasil copy-an, seperti:

Nama file sumber : TASM.EXE

Nama file tujuan : B:SS.PROG

Maka file TASM.EXE akan dicopykan pada drive b: dengan nama SS.PROG. Bila pengcopyan file berhasil dengan sukses, maka pada layar akan ditampilkan:

Pengcopyan File telah dilaksanakan ....

Keterangan program:

MOV AX,3

INT 10H

Ini adalah perintah untuk mengaktifkan mode layar 03, atau mode default

dari DOS. Dengan pengaktifan mode layar ini seluruh isi layar akan terhapus.

```
TanyaF1 :
          Cetak   Sumber

          MOV     AH,0AH
          LEA     DX,File1
          INT     21h
```

Mintalah dari user untuk memasukkan nama file yang akan dicopy. Hasil input dari user ini, disimpan pada varibel penampung "File1" yang didefinisikan untuk mampu menampung sampai 70 karakter.

```
          LEA     BX,File1
          INC     BX
          CMP     BYTE PTR [BX],0
          JE      TanyaF1
 Ulang1 :
          INC     BX
          CMP     BYTE PTR [BX],0Dh
          JNE     Ulang1
          MOV     BYTE PTR [BX],0
```

Setelah didapat nama file yang ingin dicopy, jadikanlah nama file tersebut menjadi ASCIZZ. Karena setiap input dari keyboard selalu diakhiri dengan enter(0Dh), maka kita tinggal mencari karakter enter tersebut dan menggantinya dengan byte 0, untuk membuatnya menjadi ASCIZZ.

```
TanyaF2 :
          Cetak   Tujuan

          MOV     AH,0AH
          LEA     DX,File2
          Int     21h

          LEA     BX,File2
          INC     BX
          CMP     BYTE PTR [BX],0
          JE      TanyaF2
 Ulang2 :
```

```
INC     BX
CMP     BYTE PTR [BX],0Dh
JNE     Ulang2
MOV     BYTE PTR [BX],0
```

Proses untuk menanyakan nama file hasil copy-an, sama dengan proses meminta nama file yang ingin dicopy. Nama file hasil copyan juga dijadikan ASCIIZ.

```
MOV     AH,3DH
MOV     AL,0
LEA     DX,File1 + 2
INT     21h
JC      Er
MOV     Handle1,AX
```

Bukalah file yang akan dicopy dengan atribut pembukaan 0, atau Read Only. Nomor handle file tersebut akan diberikan oleh DOS berupa suatu angka. Simpanlah nomor handle yang diberikan DOS untuk file yang dibuka ini. Untuk selanjutnya anda tinggal menggunakan nomor handle dari file ini untuk mengaksesnya.

```
MOV     AH,3CH
LEA     DX,File2 + 2
MOV     CX,00100000b
INT     21h
JC      Er
MOV     Handle2,AX
```

Buatlah sebuah file baru dengan atribut pembukaan 32 atau file archive(normal). Simpanlah nomor handle yang diberikan DOS untuk file yang baru diciptakan ini. Jika file yang baru diciptakan ini telah ada pada disk sebelumnya, maka file tersebut akan dihapus.

Copy :

```
MOV     AH,3FH
MOV     BX,Handle1
MOV     CX,1024
LEA     DX,Buffer
INT     21h
JC      Er

CMP     AX, 0
```

```
        JE      Selesai

        MOV     CX,AX
        MOV     AH,40h
        MOV     BX,Handle2
        LEA     DX,Buffer
        Int     21h
        JC      Er

        CMP     CX,AX
        JAE     Copy
```

Setelah itu, bacalah file yang akan dicopy sebanyak 1024 byte, dan disimpan pada variabel penampung buffer. Setelah pembacaan, lakukanlah penulisan kepada file baru sebanyak byte yang berhasil dibaca. Proses baca dan tulis ini dilakukan terus sampai file tersebut berhasil dicopykan semuanya.

Perhatikanlah:

Untuk membaca file yang dicopy dan menulisi file baru, kita tinggal menggunakan nomor handle yang kita dapatkan pada saat pembukaan dan pembuatan file tersebut.

```
 Selesai :
          Cetak   Good

          MOV     AH,3Fh
          MOV     BX,Handle1
          INT     21h

          MOV     BX,Handle2
          INT     21h
    EXIT :
          INT     20h
```

Setelah proses pengcopyan file selesai, tutuplah file tersebut dengan menggunakan nomor handle-nya.

```
      Er :
          CMP     AX,1
          JNE     NotErr1
          Cetak   Err1
          JMP     EXIT
```

```
NotErr1 :
          CMP     AX,2
          JNE     NotErr2
          Cetak   Err2
          JMP     EXIT
NotErr2 :
          CMP     AX,3
          JNE     NotErr3
          Cetak   Err3
          JMP     EXIT
NotErr3 :
          CMP     AX,4
          JNE     NotErr4
          Cetak   Err4
          JMP     EXIT
NotErr4 :
          CMP     AX,6
          JNE     NotErr6
          Cetak   Err6
          JMP     EXIT
NotErr6 :
          CMP     AX,21
          JNE     NotErr21
          Cetak   Err21
          JMP     EXIT
NotErr21 :
          CMP     AX,29
          JNE     NotErr29
          Cetak   Err29
          JMP     EXIT
NotErr29 :
          CMP     AX,30
          JNE     NotErr30
          Cetak   Err30
          JMP     EXIT
NotErr30 :
          Cetak   ErrL
```

```
            JMP     EXIT
```

Ini adalah bagian yang akan menerjemahkan kode kesalahan dari Dos. Proses yang dilakukan memang agak sedikit bertele-tele, karena dibandingkan satu persatu. Kita akan mempelajari cara/teknik untuk menerjemahkan kode kesalahan ini secara profesional pada bagian 23.11.

```
            Buffer  LABEL BYTE
```

Ini adalah suatu pendefinisian data istimewa dalam assembler. Dengan mendefinisikannya pada akhir file, akan kita dapatkan suatu buffer/penampung yang besar sekali (sebatas memory yang tersedia). Untuk lebih jelasnya anda bisa lompat pada bagian 24.7. sebentar, untuk membaca keterangan mengenai tipe data ini.

23.7. MENGHAPUS FILE

Untuk menghapus suatu file, dapat digunakan fungsi ke 41h dari interupsi 21h. Adapun aturan dari pemakaian fungsi ini adalah:

INPUT:

AH = 41h

DS:DX = Nama file(ASCIIZ) yang akan dihapus

OUTPUT:

Jika berhasil, maka CF = 0

Jika tidak berhasil, maka CF = 1 dan AX = kode kesalahan

Perlu anda ketahui, fungsi ini tidak mendukung karakter khusus seperti '?' dan '*' dari Dos. Dengan demikian fungsi ini hanya mampu menghapus 1 buah file setiap saat.

```
Delete MACRO Nama
        MOV   AH,41h      ; Servis untuk menghapus file
        LEA   DX,Nama     ; DS:DX menunjuk pada nama file
        INT   21h
        ENDM

 ;/=====================================================\;
 ;              Program : HAPUS.ASM                    ;
 ;              Author  : S'to                         ;
 ;              Fungsi  : Menghapus file, seperti perintah;
 ;                        delete dari Dos.             ;
 ;\=====================================================/;

       .MODEL SMALL
       .CODE
       ORG 100h

TData :JMP   Proses
       Error DB ' Sorry, File anda tidak bisa dihapus !',13,10
             DB ' Anda harus menggunakan parameter     ',13,10
             DB ' seperti:                             ',13,10
```

```
            DB '          C:\> Hapus FILE_X        ',13,10,10
            DB ' untuk menghapus file FILE_X $'

Proses : MOV    DI,80h        ; Alamat awal parameter
         MOV    AL,0Dh        ; Karakter Enter
       REPNE    SCASB         ; Cari karakter Enter
         DEC    DI            ; DI menunjuk karakter Enter

         MOV    AL,0          ; Jadikan ASCIIZ
         STOSB                ; Letakkan byte 0 pada DS:[DI]

         MOV    DI,82h        ; Awal String
         Delete [DI]          ; Hapus file parameter
         JNC    Exit          ; Jika tidak ada kesalahan, Habis

         MOV    AH,09         ; Jika ada kesalahan
         LEA    DX,Error      ; Tampilkan peringatan !
         INT    21h
Exit   :
         INT   20h
END      TData
```

Program 23.2. Menghapus File

Program 23.2. bisa anda gunakan untuk menghapus suatu file dengan parameter. Misalkan file COBA.TXT akan dihapus, maka bisa dihapus dengan cara:

HAPUS COBA.TXT

23.8. MEMINDAHKAN PENUNJUK(POINTER) FILE

Pada file terdapat pointer(penunjuk) yang berguna untuk menunjukkan suatu lokasi tertentu pada file. Dengan fungsi 42h dari Dos, penunjuk file ini dapat dipindah-pindahkan. Adapun aturan dari penggunaan fungsi ini adalah:

INPUT:

AH = 42h

BX = Handle

CX = Offset Hi(tinggi) yang menunjukkan besarnya perpindahan

DX = Offset Lo(rendah) yang menunjukkan besarnya perpindahan

AL = Mode perpindahan, dengan nilai:

- 00 untuk berpindah dari awal file
- 01 untuk berpindah terhadap posisi sekarang
- 02 untuk berpindah dari akhir file

OUTPUT:

Jika berhasil, maka CF = 0 dan DX:AX = menunjuk pada posisi baru

Jika tidak berhasil, maka CF = 1 dan AX = kode kesalahan

Adapun contoh macro dari penggunaan fungsi ini adalah:

```
Seek   MACRO  Handle,Mode,OffsetLo,OffsetHi
       PUSH   BX
       PUSH   CX
       MOV    AH,42h
       MOV    AL,Mode
       MOV    BX,Handle
       MOV    CX,OffsetHi
       MOV    DX,OffsetLo
       INT    21h
       POP    CX
       POP    BX
       ENDM
```

23.9. MENGATUR ATRIBUT FILE

Seperti yang telah kita ketahui, pada file terdapat suatu byte yang digunakan sebagai atribut dari file tersebut. Atribut ini menentukan jenis dari file tersebut(lihat sub bab 23.2).

DOS menyediakan fungsi ke 43h untuk mengatur atribut file. Adapun aturan dari pemakaian fungsi ini adalah:

```
  INPUT:
  AH    = 43h
  AL    = Mode, dengan spesifikasi
          00 untuk melihat atribut dari suatu file
          01 untuk mengubah atribut dari suatu file
  DS:AX = Nama file dalam bentuk ASCIIZ
  CX    = Atribut, dengan spesifikasi Nomor bit:
          - 0 untuk File Read Only
          - 1 untuk File Hidden
          - 2 untuk File System
          - 3 untuk Volume label.
          - 4 untuk Nama subdirectory.
          - 5 untuk File Archive
  OUTPUT:
  Jika berhasil, maka CF = 0 dan CX menunjukkan atribut dari
  file, jika mode pada AL=0
  Jika tidak berhasil, maka CF = 1 dan AX = kode kesalahan
```

```
Cetak_Kal MACRO Kal          ; Macro untuk mencetak kalimat
```

```
          MOV   AH,09
          LEA   DX,Kal
          INT   21h
          ENDM
;/================================================\;
;           Program : ATTR.ASM                     ;
;           Author  : S'to                         ;
;           Fungsi  : Melihat atribut dari file.   ;
;                     Anda bisa menggunakan parameter;
;                     dengan program ini           ;
;\================================================/;

        .MODEL SMALL
        .CODE
        ORG 100h

TData : JMP     Proses
        Error   DB 'Sorry, file tidak ditemukan atau ',13,10
                DB ' anda tidak mengunakan parameter $'
        Jenis   DB 'Jenis File ini : $'
        Attr1   DB 13,10,'-> Read Only $'
        Attr2   DB 13,10,'-> Hidden $'
        Attr3   DB 13,10,'-> System $'
        Attr4   DB 13,10,'-> Volume $'
        Attr5   DB 13,10,'-> Nama Directory $'
        Attr6   DB 13,10,'-> Archive $'
        Tabel   DW  Attr1,Attr2,Attr3,Attr4,Attr5,Attr6

Proses:
        MOV       BX,82h            ; Alamat DTA, Awal parameter
Ulang :
        CMP       BYTE PTR [BX],0Dh ; Apakah=Enter?
        JE        ASCIIZ            ; Ya, Jadikan ASCIIZ
        INC       BX                ; tunjuk karakter selanjutnya
        JMP       Ulang             ; ulangi, cari enter
ASCIIZ:
        MOV       BYTE PTR[BX],0    ; Jadikan ASCIIZ

        MOV       AH,43h          ; Servis untuk atribut file
        MOV       DX,82h          ; DX = Awal nama file ASCIIZ
        MOV       AL,00           ; Untuk membaca atribut
        INT       21h             ; Baca atribut file
        JNC       Good            ; Jika tidak error lompat
        Cetak_Kal Error           ; Error, cetak pesan dan
        JMP       Exit            ; Selesai
Good  :
        Cetak_Kal Jenis           ; Cetak kalimat
        MOV       AX,1            ; AX untuk mengetes Bit
        XOR       BX,BX           ; BX=penunjuk isi Tabel
CEK   :
        PUSH      AX              ;
        PUSH      BX              ; simpan isi register
        PUSH      CX              ;

        AND       CX,AX           ; Mengetest bit
        JCXZ      Tidak           ; Jika CX=0 artinya bit=0
        ADD       BX,BX           ; BX kali 2, untuk akses Tabel

        MOV       AH,09           ; Servis cetak kalimat
        MOV       DX,Tabel[BX]    ; Alamat offset dari kalimat
        INT       21h             ; Cetak kalimat
Tidak :
        POP       CX              ;
        POP       BX              ; Ambil nilai register
        POP       AX              ;

        ADD       AX,AX         ; AX untuk test bit berikutnya
```

```
        INC       BX            ; BX=Banyaknya pengulangan
        CMP       BX,6          ; Apakah telah mengetest 6 bit?
        JE        Exit          ; Ya! selesai
        JMP       CEK           ; Ulangi, test bit berikutnya
Exit  :
        INT       20h
END     TData
```

Program 23.3. Melihat Atribut File

Dengan program 23.3. anda bisa melihat atribut dari suatu file, termasuk file hidden, seperti IO.SYS. Untuk melihat atribut dari file tersebut ketikkan:

```
C:\>ATTR IO.SYS
Jenis File ini :
-> Read Only
-> Hidden
-> System
-> Archive
```

Mengenai penggunaan Tabel, akan dijelaskan pada sub bab 23.11 yang menjelaskan mengenai teknik untuk menerjemahkan kode kesalahan Dos dengan cepat dan mudah.

23.10. MENGUBAH NAMA FILE

Untuk mengganti nama suatu file, dapat digunakan fungsi ke 56h dari Dos. Adapun aturan dari pemakaian fungsi ini adalah:

```
INPUT:
AH    = 56h
DS:DX = Nama file lama (ASCIIZ)
ES:DI = Nama file baru (ASCIIZ)

OUTPUT:
Jika berhasil, maka CF = 0
Jika tidak berhasil, maka CF = 1 dan AX = kode kesalahan
```

23.11. KODE KESALAHAN DOS

Pada bab ini, kita telah banyak menyinggung mengenai kode kesalahan. Kode kesalahan umumnya dilaporkan dalam register AX setelah suatu operasi mengalami kesalahan(Error). Adapun arti dari kode kesalahan ini, bisa anda lihat pada gambar 23.1.

Kode salah	Arti Kode Salah
01	Salah perintah
02	File tidak ditemukan
03	Path tidak ditemukan
04	File yang dibuka terlalu banyak
05	Operasi ditolak
06	penggunaan file handle yang salah
07	MCB(Memory Control Blocks) telah rusak
08	Kekurangan memory
09	Kesalahan alamat memory blok
10	Kesalahan environment string
11	Kesalahan format
12	Kesalahan kode akses
13	Kesalahan data
15	Kesalahan spesifikasi drive
16	Tidak dapat menghapus directory aktif
17	Device yang tidak sama
18	Tidak ada file yang ditemukan lagi
19	Tidak dapat menulis pada disket yang diprotek
20	Unit tidak diketahui
21	Drive belum siap
22	Perintah tidak diketahui
23	Data disk terdapat kesalahan
25	Pencarian alamat pada disket ada kesalahan
26	Tipe media tidak diketahui
27	Pencarian nomor sektor tidak ditemukan
28	Printer tidak diberi kertas
29	Kesalahan pada saat penulisan
30	Kesalahan pada saat pembacaan
61	Queue Printer telah penuh
65	Perintah ditolak
80	File telah ada
82	Tidak bisa membuat entri directory
83	Kesalahan pada interupsi 24
86	Kesalahan password

87 Kesalahan parameter

Gambar 23.1. Arti dari kode kesalahan DOS yang umum

Pada program yang lengkap biasanya bila terdapat error, akan dilaporkan kepada pemakainya. Cara yang kuno untuk digunakan untuk mencetak arti kesalahan adalah dengan membandingkan kode kesalahan yang dihasilkan dan mencetak pesannya, seperti:

```
TData: JMP     Proses
       Error1  DB ' Salah perintah ! $'
       Error2  DB ' File tidak ditemukan ! $'
       Error3  DB ' Path tidak ditemukan ! $'
               :
Proses:        :
       CMP     AX,1     ; Apakah error kode 1 ?
       JNE     Err2     ; Bukan! lompat ke Err2
       Cetak   Error1   ; Ya! Cetak pesan dari kode error 1
       JMP     Exit     ; Keluar
  Err2:
       CMP     AX,2     ; Apakah error kode 2 ?
       JNE     Err3     ; Bukan! lompat ke Err3
       Cetak   Error2   ; Ya! Cetak pesan dari kode error 2
       JMP     Exit     ; Keluar
  Err3:
       CMP     AX,3     ; Apakah error kode 3 ?
       JNE     Err4     ; Bukan! lompat ke Err4
       Cetak   Error3   ; Ya! Cetak pesan dari kode error 3
       JMP     Exit     ; Keluar
  Err4:
          :
          :
```

Apakah cara diatas sudah tepat ? Tidak!. Bila pengecekan dari kode kesalahan hanyalah 1 atau 2 buah, cara diatas dapat anda gunakan. Bagaimana jika kode kesalahan yang akan kita cek, ternyata jumlahnya mencapai ratusan?

Program anda akan tampak bertele-tele dan panjang selain itu ukuran file akan menjadi sangat besar.

Salah satu cara yang dapat anda gunakan untuk memecahkan masalah diatas adalah dengan membuat suatu tabel array yang berisi alamat offset dari masing-masing pesan kesalahan. Kemudian dari tabel alamat kode kesalahan ini digunakan untuk mencetak pesan salah yang dihasilkan.

```
Cetak  MACRO Kal
       MOV   AH,09
       MOV   DX,Kal
       INT   21h
       ENDM

;/=============================================\;
;            Program : ERROR.ASM                ;
;            Author  : S'to                     ;
;            Fungsi  : Mencetak pesan kode error   ;
;                      dengan cara yang praktis.   ;
;\=============================================/;

     .MODEL SMALL
     .CODE
     ORG  100h

TData : JMP      Proses
        Error01  DB 'Salah perintah $'
        Error02  DB 'File tidak ditemukan $'
        Error03  DB 'Path tidak ditemukan $'
        Error04  DB 'File yang dibuka terlalu banyak $'
        Error05  DB 'Operasi ditolak $'
        Error06  DB 'Penggunaan file handle yang salah $'
        Error07  DB 'MCB(Memory Control Blocks) telah rusak $'
        Error08  DB 'Kekurangan memory $'
        Error09  DB 'Alamat memory blok salah $'
        Error10  DB 'Environment String salah $'
        Error11  DB 'Kesalahan format $'
        Error12  DB 'Kode akses salah $'

        Tabel    DW  Error01,Error02,Error03,Error04,Error05
                 DW  Error06,Error07,Error08,Error09,Error10
                 DW  Error11,Error12
      Test_Error DW  03
Proses :
        MOV   AX,Test_Error
        DEC   AX
        ADD   AX,AX
        MOV   BX,AX
        CETAK Tabel[BX]       ; Cetak pesan kode error

        INT 20h
END     TData
```

Program 23.4. Cara yang praktis untuk mencetak arti kode kesalahan DOS

Bila program 23.4. dijalankan, maka pada layar akan tercetak:

Path tidak ditemukan

Seperti yang diharapkan, arti kode error 03(Test_Error) akan tercetak pada layar. Anda bisa mengubah kode salah pada variabel Test_Error dengan angka 01 sampai 12 untuk melihat pesan yang akan ditampilkan.

Perhatikanlah:

Pada varibel Tabel kita mencatat alamat offet dari masing- masing pesan kesalahan dengan cara:

```
Tabel     DW  Error01,Error02,Error03,Error04,Error05
          DW  Error06,Error07,Error08,Error09,Error10
          DW  Error11,Error12
```

dimana masing-masing alamat offset menggunakan 1 word.

```
MOV   AX,Test_Error
DEC   AX
```

Karena kode error yang pertama adalah 01, maka perlu kita kurangi dengan 1 supaya menjadi 0. Dengan demikian kode error 1 akan menunjuk pada word pertama pada Tabel yang kita ketahui bahwa word pertama dari Tabel merupakan alamat offset pesan kode salah 01.

```
ADD   AX,AX
MOV   BX,AX
```

Karena setiap alamat offset dari pesan kode salah menggunakan 1 word atau 2 byte, maka untuk mengambil word selanjutnya dari Tabel yang mencatat alamat offset pesan kode error selanjutnya, kita harus mengalikan kode error dengan 2 atau menambah kode error dengan dirinya sendiri.

```
CETAK Tabel[BX]
```

Kemudian dengan Register Indirect Addressing kita mengambil alamat offset pesan kode salah dari Tabel. Seperti biasa, pada pencetakan kalimat kita mengambil alamat offset dari suatu string yang diakhiri dengan tanda '$' untuk dicetak dengan fungsi 09 dari Dos. Pada pencetakan string ini alamat offset sudah didapat dari Tabel[BX], sehingga perintah: "LEA DX,Kal" dari fungsi 09 dapat dirubah menjadi: "MOV DX,Kal"

BAB XXIV
PROGRAM RESIDEN

24.1. VEKTOR INTERUPSI

Pada bab 3 telah dibahas mengenai pengertian dasar interupsi, bila anda sudah lupa, bacalah kembali sebelum membaca bagian ini. Pada bagian ini akan kita lihat lebih lanjut khusus mengenai vektor interupsi.

Seperti yang telah dikatakan, setiap interupsi menggunakan 4 byte memory sebagai alamat awal interupsi, yaitu alamat yang akan dituju setiap terjadi interupsi. Keempat byte ini dicatat pada Interrupt Vektor Table yang terdapat pada memory rendah, 0000:0000 sampai 0000:03FFh. Dengan demikian, interupsi 00 akan menggunakan alamat 0000:0000-0000:0003, interupsi 01 akan menggunakan alamat 0000:0004-0000:0007, dan seterusnya. Untuk mencari alamat awal dari suatu nomor interupsi digunakan rumus:

Alamat Awal = 4 * Nomor-Interupsi

Sebagai contohnya, setiap kali kita menekan tombol PrtScr untuk mencetak isi layar pada printer akan selalu terjadi interupsi 05. Komputer kemudian akan menuju alamat awal interupsi 05, yaitu 0000:0020 (4*05=20). Dari alamat awal ini kemudian akan dilihat isi dari keempat byte, yaitu pada alamat 0000:0020 - 0000:0023. Keempat byte ini mencatat alamat CS(2 byte) dan IP(2 byte), yaitu alamat yang akan dituju oleh komputer selanjutnya. Misalkan isi dari keempat byte ini adalah 3200h:0D8Bh, artinya komputer akan melompat pada alamat tersebut dan menjalankan program yang terdapat pada alamat tersebut sampai bertemu dengan perintah IRET. Program inilah yang disebut sebagai Interrupt Handler 05, yaitu program yang akan dilaksanakan setiap kali terjadi interupsi 05. Secara default program yang akan dilaksanakan terdapat pada BIOS, dimana program tersebut akan mencetak tampilan pada layar ke printer.

24.2. MENDAPATKAN ALAMAT VEKTOR INTERUPSI

Untuk melihat isi dari alamat awal suatu vektor interupsi dapat digunakan dua cara. Cara pertama, adalah dengan membaca secara langsung keempat byte alamat awal yang mencatat alamat berturut-turut Offset Lo, Offset Hi, Segment Lo dan Segment Hi dari interrupt handler. Cara kedua adalah dengan menggunakan interupsi 21h fungsi 35h. Cara kedua lebih mudah untuk digunakan, oleh sebab itu akan kita gunakan pada program-program selanjutnya.

Untuk menggunakan fungsi ke 35h ini, isilah AH dengan 35h dan AL dengan nomor vektor interupsi sebelum dilaksanakan interupsi 21h. Hasil dari

interupsi ini akan disimpan pada pasangan register ES:BX. Dimana ES mencatat alamat segment dan BX mencatat alamat offset vektor interupsi dari nomor interupsi yang dimasukkan pada AL.

```
Ambil_Vec  MACRO   NoInt,Alamat
           MOV     AH,35h          ; Servis untuk mencari vektor
           MOV     AL,NoInt        ; No inteurpsi
           INT     21h             ; Laksanakan
           MOV     Alamat,BX       ; Offset
           MOV     Alamat[2],ES    ; Segment
           ENDM
```

Untuk menggunakan macro ini anda bisa menyediakan suatu varibael 2 word untuk menampung alamat hasil dari interupsi ini, seperti: Alamat DW ?,?.

24.3. MERUBAH VEKTOR INTERUPSI

Secara default, nomor interupsi 00h-7Fh akan menjalankan program yang terdapat ROM BIOS, dan nomor interupsi 20h-FFh akan menjalankan program yang disediakan oleh DOS. Interrupt Handler yang disediakan oleh BIOS ini tidak bisa dihapus secara SoftWare dan selalu tersedia pada setiap komputer. Sedangkan Interrupt Handler yang disediakan oleh DOS akan tersedia pada saat sistem operasi DOS telah masuk kedalam memory komputer.

Suatu interrupt handler bisa saja diganti, misalkan kita menginginkan penekanan tombol PrtScr tidak mencetak isi layar tetapi mem-BOOT komputer sama halnya dengan penekanan tombol Ctrl+Alt+Del. Karena Interrupt handler yang asli, baik dari BIOS maupun DOS tidak bisa dihapus maka cara yang digunakan untuk merubah interrupt handler adalah dengan mengganti isi dari Interrupt Vektor Table.

Untuk mengganti atau mengarahkan suatu nomor interupsi dapat secara langsung atau menggunakan fungsi 25h dari interupsi 21h. Untuk menggunakan fungsi ini, isilah AH dengan 25h, AL dengan nomor interupsi yang akan diganti vektornya, pasangan DS:DX berisi alamat yang akan dituju pada saat terjadi interupsi tersebut.

```
Arah_Vec   MACRO   NoInt,Alamat
           MOV     AX,Alamat[2]
           MOV     DS,AX           ; DS = segment
           MOV     DX,Alamat       ; DX = offset
           MOV     AH,25h          ; Servis untuk merubah vektor
           MOV     AL,NoInt        ; No interupsi
```

```
            INT     21h
            ENDM
```

Sama seperti macro untuk mendapatkan alamat vektor interupsi, untuk menggunakan macro ini anda harus menyediakan suatu varibael 2 word yang digunakan sebagai penampung alamat yang akan dituju dari suatu interupsi, seperti: Alamat DW ?,?.

Pada program berikut ini akan anda lihat bagaimana membelokkan interupsi 05h(PrtScr) ke interupsi 1Bh. Interupsi 1Bh adalah suatu interupsi yang akan selalu terjadi bila anda menekan tombol Ctrl+Break. Dengan demikian setelah program "breaks" dijalankan, penekanan tombol PrtScr akan sama halnya dengan penekanan tombol Ctrl+Break.

```
Arah_Vec    MACRO   NoInt,Alamat
            MOV     AX,Alamat[2]
            MOV     DS,AX           ; DS = segment
            MOV     DX,Alamat       ; DX = offset
            MOV     AH,25h          ; Servis untuk merubah vektor
            MOV     AL,NoInt        ; No interupsi
            INT     21h
            ENDM

Ambil_Vec   MACRO   NoInt,Alamat
            MOV     AH,35h          ; Servis untuk mencari vektor
            MOV     AL,NoInt        ; No inteurpsi
            INT     21h             ; Laksanakan
            MOV     Alamat,BX       ; Offset
            MOV     Alamat[2],ES    ; Segment
            ENDM

;/=====================================================\;
;            Program : BREAKS.ASM                      ;
;            Author  : S'to                            ;
;            Fungsi  : Program yang akan mengganti     ;
;                      intrupsi 05 <PrtScr> menjadi    ;
;                      interupsi 1Bh <Ctrl+Break>.     ;

;\=====================================================/;

          .MODEL SMALL
          .CODE
          ORG 100h

TData :   JMP    Res_kan
          Break              EQU  23h
          PrtScr             EQU  05
          Addr_Break         DW   ?,?   ; Untuk menyimpan Alamat
                                        ; vektor Ctrl Break

Res_Kan :
          Ambil_Vec  Break,Addr_Break   ; Anbil alamat Ctrl+C
          Arah_Vec  PrtScr,Addr_Break   ; Rubah vektor PrtScr

          INT    20h
END       TData
```

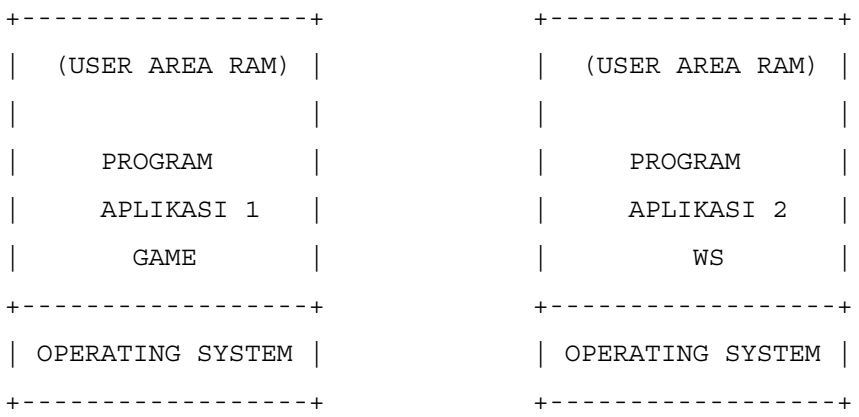
Program 24.1. Mengganti fungsi PrtScr menjadi Ctrl+Break

Bila progrm 24.1. dijalankan, maka tombol PrtScr sudah tidak akan berfungsi seperti biasanya, tetapi berfungsi seperti penekanan tombol Ctrl Break.

24.4. APA ITU PROGRAM RESIDEN ?

Pada waktu kita menyalakan komputer, ia mencari sistem operasi di drive A: ataupun C: ,kemudian memasukkannya kedalam memori bawah. Selanjutnya sistem akan terus berada disitu dan apabila kita menjalankan program aplikasi misalnya game maka program tersebut akan disimpan di atas sistem operasi, sehingga sistem operasi tetap ada walaupun kita sedang menjalankan game tersebut. Inilah yang disebut residen, yaitu program yang tetap tinggal di memori.

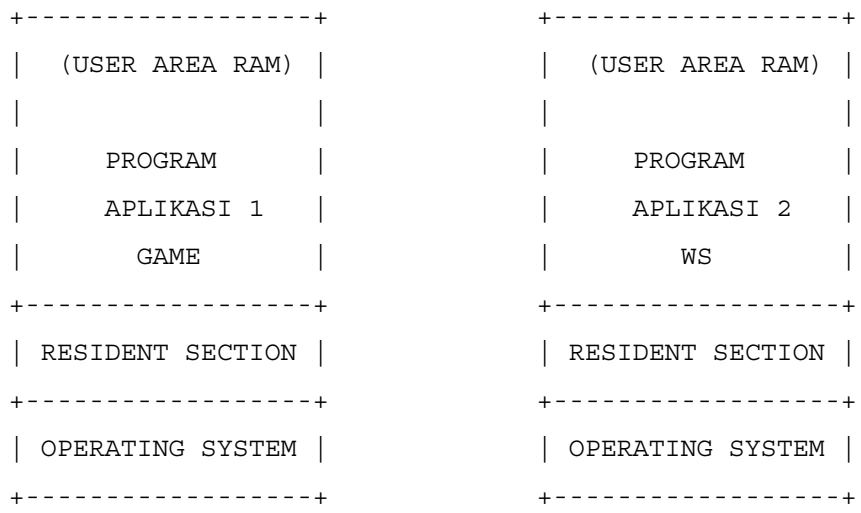
Dalam contoh kita ini bila game tadi telah selesai maka ia akan lenyap dari memori dan bila kita menjalankan program aplikasi lainnya, misalnya WS maka tempat memori yang digunakan oleh game kita akan digunakan oleh WS. Ini adalah contoh dari program yang tidak residen karena ia hanya sementara waktu berada di memori. Contoh program residen yang terkenal misalnya SideKick, Print(dos) dan Doskey.



Gambar 24.1. Peta RAM tanpa program Residen

Program residen adalah program yang akan menetap dimemory seperti halnya DOS dan program residen ini akan berada tepat diatas Operating System. Program residen akan dianggap sebagai bagian dari Operating System sehingga bila

dijalankan program aplikasi maka program aplikasi tersebut akan ditaruh diatas program residen sehingga program residen kita tetap utuh.



Gambar 24.2. Peta RAM dengan program residen

Program residen adalah suatu bentuk program yang menarik. Karena program residen menetap pada memory, maka semakin banyak program residen dijalankan, memory akan semakin berkurang untuk digunakan oleh program aplikasi. Program residen, haruslah dibuat sekecil mungkin untuk menghindari pemakaian memory yang terlalu banyak. Hanya dengan Assembler-lah, sebuah program dapat dibuat sekecil mungkin! Bayangkan, program untuk menghapus layar, dengan bahasa tingkat tinggi seperti pada pascal dan C digunakan sekitar 3232 byte, sedangkan pada assembler sekitar 7 byte.

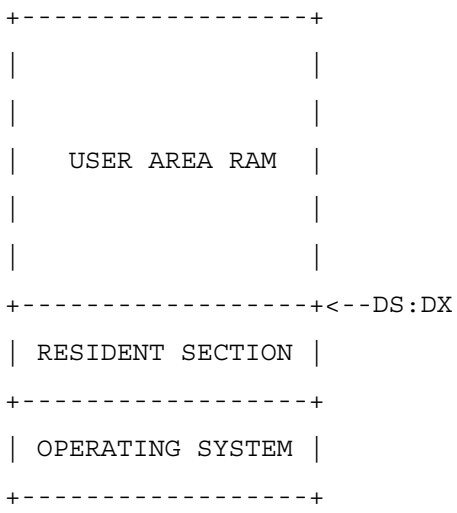
24.5. MODEL PROGRAM RESIDEN

Dalam pembuatan program residen, kita dapat membaginya dalam 2 bagian pokok, yaitu :

- **Initialize section**, yaitu bagian dari program yang bertugas meresidenkan residen section. Bagian ini sendiri tidak residen, dan pada bagian inilah suatu vektor interupsi diubah.
- **Residen section**, yaitu bagian program yang akan menetap pada memory. Program ini akan tetap tinggal pada memory sampai dihilangkan, atau sampai komputer direset.

Pada program sebelumnya, kita selalu mengakhiri program dengan interupsi 20h yang akan mengembalikan kontrol program sepenuhnya pada DOS. Pada program residen, program akan selalu kita akhiri dengan interupsi 27h ataupun

interupsi 21h fungsi 31h. Untuk menggunakan interupsi 27h, kita tinggal mengisi pasangan register DS:DX dengan batas memory yang akan diresidenkan.



Gambar 24.3. Penggunaan interupsi 27h untuk meresidenkan program

Untuk membuat program residen, anda bisa menggunakan bentuk program seperti pada gambar 24.4.

```
-----------------------------------------------------------------
         .MODEL SMALL
         .CODE
         ORG 100h

TData :  JMP    Res_kan
        +------------------+
        |  Tempat   untuk  |
        |  mendefinisikan  |
        |      DATA        |
        +------------------+
Bag_Res  PROC
         PUSH   AX           ;
         PUSH   BX           ;
         PUSH   CX           ;
         PUSH   DX           ;
         PUSH   ES           ; Simpan isi semua register
         PUSH   DI           ;
         PUSH   DS           ;
```

```
          PUSH   SI           ;
          +------------------+
          |  Tempat handler  |
          |  interupsi yang  |
          |      baru        |
          +------------------+
          POP    SI           ;
          POP    DS           ;
          POP    DI           ;
          POP    ES           ;
          POP    DX           ; Kembalikan isi semua register
          POP    CX           ;
          POP    BX           ;
          POP    AX           ;
          IRET                ; Akhir dari interupt handler
Bag_Res   ENDP

Res_Kan :
          +------------------+
          |  Tempat   untuk  |
          |   memanipulasi   |
          | vektor interupsi |
          +------------------+
          LEA    DX,Res_Kan
          INT    27h
END       TData
-------------------------------------------------------------------
```

Gambar 24.4. Model Program Residen

24.6. PROGRAM RESIDEN PERTAMA

Pada program berikut akan kita lihat bagaimana membelokkan merubah vektor interupsi PrtScr menuju program kita. Dengan cara yang sama anda bisa membelokkan vektor interupsi yang lain, dan membuat suatu handler yang baru untuknya.

```
;/=====================================================\;
;            Program : RES1.ASM                       ;
;            Author  : S'to                           ;
```

```
;            Fungsi  : Program residen yang membelokkan  ;
;                      intrupsi 05 atau interupsi PrtScr ;
;                      menuju program buatan sendiri     ;
;\=======================================================/;

          .MODEL SMALL
          .CODE
          ORG 100h

TData :   JMP     Res_kan
          Pesan   DB '  Interupsi 5<PrtScr> telah di belokkan !!  '
          NoInt   EQU 05h

Bag_Res   PROC
          PUSH    AX            ;
          PUSH    BX            ;
          PUSH    CX            ;
          PUSH    DX            ;
          PUSH    ES            ; Simpan isi semua register
          PUSH    DI            ;
          PUSH    DS            ;
          PUSH    SI            ;

          MOV     AX,1300h      ;
          MOV     BL,01001111b  ;
          MOV     BH,00         ;
          MOV     DL,20         ;
          MOV     DH,12         ; Program interupt handler PrtScr
          MOV     CX,44         ; yang baru.
          PUSH    CS            ;
          POP     ES            ;
          LEA     BP,Pesan      ;
          INT     10h           ;

          POP     SI            ;
          POP     DS            ;
          POP     DI            ;
          POP     ES            ;
          POP     DX            ; Kembalikan isi semua register
          POP     CX            ;
          POP     BX            ;
          POP     AX            ;
          IRET                  ; Akhir dari interupt handler
Bag_Res   ENDP

Res_Kan :
          MOV     AH,25h        ;
          MOV     AL,NoInt      ; Untuk merubah vektor interupsi
          LEA     DX,Bag_Res    ; 05 menuju 'Bag_Res'
          INT     21h           ;

          LEA     DX,Res_Kan    ;
          INT     27h           ; Untuk meresidenkan bagian
END       TData                 ; "Bag_Res"
```

Program 24.2. Membuat Program Residen

Bila program 24.2. dijalankan, maka tombol PrtScr sudah tidak akan berfungsi lagi. Setiap kali tombol PrtScr ditekan, pada posisi 20,12 akan ditampilkan pesan:

Interupsi 5<PrtScr> telah di belokkan !!

Perhatikanlah, bahwa pada program ini terdapat 2 bagian pokok, yaitu bagian yang residen dan bagian yang meresidenkan. Bagian yang meresidenkan hanya dijalankan sekali, sedangkan bagian yang residen akan dijalankan setiap kali terjadi penekanan tombol PrtScr. Bagian yang meresidenkan adalah:

```
Res_Kan :
        MOV     AH,25h        ;
        MOV     AL,NoInt      ; Untuk merubah vektor interupsi
        LEA     DX,Bag_Res    ; 05 menuju 'Bag_Res'
        INT     21h           ;

        LEA     DX,Res_Kan    ;
        INT     27h           ; Untuk meresidenkan bagian
END     TData                 ; "Bag_Res"
```

Bagian ini tugasnya meresidenkan bagian Bag_Res. Sebelum bagian Bag_Res diresidenkan, vektor interupsi PrtScr(05) diubah menuju progam Bag_Res. Bila anda hanya merubah interupsi PrtScr menuju program Bag_Res tanpa diresidenkan, maka akan menyebabkan komputer anda menjadi hang, mengapa? Walaupun vektor interupsi tetap menunjuk pada lokasi atau alamat yang sama, tetapi tempat yang digunakan program kita telah diserahkan kepada Dos untuk digunakan oleh aplikasi lain.

```
Bag_Res  PROC
        PUSH    AX            ;
        PUSH    BX            ;
        PUSH    CX            ;
        PUSH    DX            ;
        PUSH    ES            ; Simpan isi semua register
        PUSH    DI            ;
        PUSH    DS            ;
        PUSH    SI            ;
```

Ini adalah awal dari bagian yang residen. Simpanlah semua nilai register pada awal program residen untuk mencegah terganggunya program lain yang sedang berjalan pada saat tombol PrtScr ditekan.

```
        MOV     AX,1300h      ;
        MOV     BL,01001111b  ;
        MOV     BH,00         ;
```

```
        MOV     DL,20           ;
        MOV     DH,12           ; Program interupt handler PrtScr
        MOV     CX,44           ; yang baru.
        PUSH    CS              ;
        POP     ES              ;
        LEA     BP,Pesan        ;
        INT     10h             ;
```

Bagian ini dapat dikatakan sebagai handler baru bagi interupsi PrtScr. Tombol PrtScr yang biasanya mencetak tampilan layar pada printer akan berubah menjadi mencetak pesan pada layar. dengan demikian anda bisa membuat handler baru yang akan melakukan sesuatu setiap kali terjadi penekanan tombol PrtScr.

Perhatikanlah! :

untuk mencetak pesan pada layar digunakan interupsi 10h, dan bukannya interupsi Dos fungsi 09 yang biasanya kita gunakan. Mengapa demikian ? Sebagian besar Interupsi Dos tidak bisa digunakan pada program residen, karena sifat dari Dos yang tidak reentrant. Masalah ini akan kita bicarakan lebih lanjut nantinya.

```
        POP     SI              ;
        POP     DS              ;
        POP     DI              ;
        POP     ES              ;
        POP     DX              ; Kembalikan isi semua register
        POP     CX              ;
        POP     BX              ;
        POP     AX              ;
        IRET                    ; Akhir dari interupt handler
Bag_Res ENDP
```

Pada akhir program residen, kembalikanlah nilai semua register yang disimpan, disertai perintah IRET(Interrupt Return). Perintah IRET akan mengambil alamat CS dan IP serta nilai Flag pada stack untuk kembali menuju program yang diselanya. CS, IP dan nilai flag disimpan pada stack pada saat terjadi interupsi, inilah rahasianya mengapa program dapat berjalan normal kembali setelah mendapat interupsi.

24.7. MENGUNCI CAPS LOCK

Pada alamat 40h:17h terdapat data tentang status tombol keyboard dimana

bit ke 7 digunakan untuk menandakan keadaan dari tombol caps lock. Bit tersebut akan bernilai 1 bila caps lock sedang aktif dan 0 bila caps lock tidak aktif. Dengan mengubah bit ke 7 pada alamat 40h:17h tersebut kita bisa menyalakan tombol caps lock tanpa menekannya.

```
Aksi    MACRO
        MOV    AX,40h
        MOV    ES,AX          ; ES=40h
        MOV    AX,ES:[17h]    ; AX=40h:17h
        OR     AX,01000000b   ; Jadikan bit ke 7 menjadi 1
        MOV    ES:[17h],AX    ; Masukkan kembali ke 40h:17h
        ENDM


;/================================================\;
;            Program : CAPS-ON.ASM                 ;
;            Author  : S'to                        ;
;            Fungsi  : Program residen yang akan   ;
;                      mengunci Caps Lock sehingga ;
;                      nyala terus                 ;
;\================================================/;

         .MODEL SMALL
         .CODE
         ORG 100h

TData :  JMP    Res_kan
         NoInt  EQU 1Ch

Bag_Res  PROC
         PUSH   AX            ;
         PUSH   BX            ;
         PUSH   CX            ;
         PUSH   DX            ;
         PUSH   ES            ; Simpan isi semua register
         PUSH   DI            ;
         PUSH   DS            ;
         PUSH   SI            ;

         Aksi

         POP    SI            ;
         POP    DS            ;
         POP    DI            ;
         POP    ES            ;
         POP    DX            ; Kembalikan isi semua register
         POP    CX            ;
         POP    BX            ;
         POP    AX            ;
         IRET                 ; Akhir dari interupt handler
Bag_Res  ENDP

Res_Kan :
         MOV    AH,25h        ;
         MOV    AL,NoInt      ; Untuk merubah vektor interupsi
         LEA    DX,Bag_Res    ; 1Ch menuju 'Bag_Res'
         INT    21h           ;

         LEA    DX,Res_Kan    ;
         INT    27h           ; Untuk meresidenkan bagian
END      TData                ; 'Bag_Res'
```

Program 24.3. Mengunci Caps Lock

Pada program kita kali ini yang dibelokkan adalah interupsi 1Ch. Handler Interupsi ini secara defaultnya hanyalah berisi perintah IRET karena interupsi ini memang disediakan untuk digunakan oleh pemakai.

Interupsi 1Ch terjadi kurang lebih 18,2 kali setiap detiknya. Karenanya dengan menggunakan interupsi 1Ch ini penekanan tombol Caps Lock menjadi seakan-akan tidak berarti lagi karena selalu dinyalakan oleh program kita.

24.8. TIPE DATA ISTIMEWA

Pada assembler terdapat suatu tipe data yang istimewa, yaitu pendefinisian data melalui perintah LABEL, dengan syntax:

Nama LABEL TipeData

Pendefinisian data dengan DB, DW, DD, DF, DQ dan DT akan menyebabkan assembler menyediakan suatu tempat khusus. Misalkan anda mendefinisikan suatu data dengan "A DW ?", maka assembler akan menyediakan 2 byte di memory untuknya. Anda hanya dapat menggunakan 2 byte pada memory melalui variabel "A".

Dengan penggunaan label, assembler akan menyediakan memory dimulai dari lokasi pendefinisiannya sampai sebatas memory anda. Selain itu penggunaan Label tidak menggunakan memory khusus. Pada program 24.4, program COM yang dihasilkan menggunakan memory 26 byte. Bila penggunaan label dihilangkan dan pengisian angka untuk variabel A,B dan C dilakukan secara langsung, memory yang digunakan oleh pada program 24.4. juga 26 byte!.

```
;/=======================================\;
;       Program : LABEL.ASM               ;
;       Author  : S'to                    ;
;       Fungsi  : pendefinisian data dengan ;
;                 LABEL                   ;
;\=======================================/;

         .MODEL SMALL
         .CODE
         ORG 100h

TData : JMP  Proses
        XX   LABEL BYTE
        A    DB    1
        B    DB    2
        C    DB    3

Proses:
        MOV  XX[0],0Ah    ; = MOV A,0Ah
        MOV  XX[1],0Bh    ; = MOV B,0Bh
        MOV  XX[2],0Ch    ; = MOV C,0Bh
```

```
        INT  20h
END     TData
```

Program 24.4. Penggunaan LABEL

Karena kita mendefinisikan "XX label byte" diatas variabel A, maka byte pertama dari "XX" akan sama dengan variabel "A", byte keduanya sama dengan "B" dan byte ketiganya sama dengan "C". Dengan demikian perintah "MOV XX[0],0Ah" adalah identik dengan "MOV A,0Ah".

```
     +-XX+---+---+---+---
     +---+---+---+---+---
     | 1 | 2 | 3 |   |
     +---+---+---+---+---
      "A" "B" "C"
```

Dengan penggunaan label ini, kita bisa mengakses suatu tempat di memory dengan menggunakan 2 atau lebih nama yang berlainan. Apa kelebihan lainnya ?

- Dengan mendefinisikan suatu variabel label pada akhir program, maka akan didapatkan suatu variabel penampung yang besar sekali, tanpa harus memperbesar program.
- Dengan pendefinisian label juga dimungkinkan pengaksesan data dengan tipe data yang berlainan pada variabel. Supaya lebih jelas, bisa anda lihat pada program berikut ini:

```
;/=======================================\;
;       Program : LABEL1.ASM              ;
;       Author  : S'to                    ;
;       Fungsi  : pendefinisian data dengan ;
;                 LABEL                   ;
;\=======================================/;

         .MODEL SMALL
         .CODE
         ORG 100h

TData : JMP  Proses
        XX   LABEL WORD
        A    DB    1,2
Proses:
        MOV  XX,0Ah    ;=> A[0]=0Ah  Dan A[1]=00

        INT  20h
END     TData
```

Program 24.5. Merubah tipe data dengan Label

Pada program 24.5. dapat dilihat bahwa kita bisa saja mengakses nilai pada variabel A yang didefinisikan dengan tipe data byte diakses dengan tipe data word. Dengan demikian penggunaan label memungkinkan kita untuk mengakses suatu data dengan tipe data yang berbeda.

```
+--XX---+-------+---
+---+---+---+---+---
| 1 | 2 |   |   |
+---+---+---+---+---
A[0] A[1]
```

24.9. MEMANGGIL HANDLER INTERUPSI LAMA

Pada program yang mengganti handler interupsi, kadang-kadang kita masih ingin melaksanakan handler yang asli. Untuk itu lihatlah pada program 24.6. berikut:

```
Cetak_Pesan MACRO   Pesan,Banyak,X,Y
            MOV     AX,1300h     ; Fungsi untuk mencetak kalimat
            MOV     BL,01001111b ; Atribut
            MOV     BH,00        ; Nomor halaman
            MOV     DL,X         ; Posisi kolom
            MOV     DH,Y         ; Posisi baris
            MOV     CX,Banyak    ; Banyaknya karakter
            PUSH    CS           ; ES:BP mencatat +
            POP     ES           ; lokasi kalimat
            LEA     BP,Pesan     ;
            INT     10h          ; laksanakan
            ENDM

Readkey     MACRO                ; Macro untuk menunggu penekanan
            MOV      AH,00       ; sembarang tombol dari keyboard
            INT      16h         ;
            ENDM

Ambil_Vec   MACRO   NoInt,Alamat
            MOV     AH,35h          ; Servis untuk mencari vektor
            MOV     AL,NoInt        ; No interupsi
            INT     21h             ; Laksanakan
            MOV     Alamat,BX       ; Offset
            MOV     Alamat[2],ES    ; Segment
            ENDM

;/============================================================\;
;          Program : OLDINT.ASM                               ;
;          Author  : S'to                                     ;
;          Fungsi  : Pada saat PrtScr ditekan, program akan   ;
;                    memberikan anda kesempatan untuk         ;
;                    menyiapkan printer sebelum mulai mencetak ;
;\============================================================/;

         .MODEL SMALL
         .CODE
         ORG 100h

TData :  JMP              Res_kan
         PrtScr           EQU 05
```

```
          Addr_PrtScr_Asli  LABEL DWORD
          Addr_PrtScr       DW  ?,?
          Pesan1            DB '--> Siapkan printer anda !! Tekan'
                            DB ' sembarang tombol untuk mulai'
                            DB ' mencetak <--'
          Pesan2          DB '>> PrtScr sudah dilaksanakan,'
                          DB 'semoga anda puas dengan hasilnya <<'

Bag_Res   PROC
          PUSH   AX           ;
          PUSH   BX           ;
          PUSH   CX           ;
          PUSH   DX           ;
          PUSH   ES           ; Simpan isi semua register
          PUSH   DI           ;
          PUSH   DS           ;
          PUSH   SI           ;

          Cetak_Pesan Pesan1,80,0,12
          Readkey

          PUSHF
          CALL   Addr_PrtScr_Asli  ; Panggil handler PrtScr
                                   ; yang asli
          Cetak_Pesan Pesan2,65,5,14

          POP    SI           ;
          POP    DS           ;
          POP    DI           ;
          POP    ES           ;
          POP    DX           ; Kembalikan isi semua register
          POP    CX           ;
          POP    BX           ;
          POP    AX           ;
          IRET                ; Akhir dari interupt handler
Bag_Res   ENDP

Res_Kan :
          Ambil_Vec PrtScr,Addr_PrtScr

          MOV       AH,25h       ;
          MOV       AL,PrtScr    ; Membelokkan vektor interupsi
          LEA       DX,Bag_Res   ;
          INT       21h          ;

          LEA       DX,Res_Kan   ; Meresidenkan program Bag_Res
          INT       27h
END       TData
```

Program 24.6. Teknik memanggil handler interupsi yang asli

Bila program 24.6. dijalankan, maka setiap penekanan tombol PrtScr, komputer akan menampilkan pesan:

---> Siapkan printer anda !!

Tekan sembarang tombol untuk mulai mencetak <---

Komputer akan segera mencetak isi layar pada printer bila ditekan sembarang tombol. Pencetakan dilakukan dengan memanggil interrupt handler PrtScr yang asli dengan:

PUSHF

CALL Addr_PrtScr_Asli

Kedua perintah ini mensimulasikan perintah interrupt. Seperti yang telah kita ketahui, setiap interrupt handler selalu diakhiri dengan perintah IRET yang akan mengambil CS, IP dan Flags dari stack. Karena perintah Call hanya akan menyimpan CS dan IP dalam stack maka kita perlu menyimpan flags dalam stack secara manual. Pada variabel "Addr_PrtScr_Asli" kita mendefinisikannya sebagai label Double Word sehingga kedua word variabel "Addr_PrtScr" dapat diakses dengan tipe data Double Word(alamat CS dan IP).

24.10. MENGATASI MASALAH REENTRANTCY DOS

Pada saat pertama DOS diciptakan, ia dirancang dengan system single user, sehingga DOS sering disebut sebagai "non-reentrant operating system". Bila terjadi interupsi dari DOS sementara interupsi DOS yang lainnya sedang aktif, data-data yang tersimpan dalam stack akan menjadi berantakan dan komputer akan menjadi hang. Inilah sebabnya, mengapa pada program residen interupsi dari DOS tidak bisa digunakan. Tetapi DOS menyediakan banyak fungsi yang sangat berguna dan tidak terdapat pada fungsi BIOS, seperti penanganan operasi pada file, memory dan sebagainya. Banyak orang yang mengira bahwa interupsi DOS tidak boleh sama sekali dipakai dalam program residen. Benarkah demikian ? Tidak.

Seperti yang telah kita ketahui, interupsi DOS tidak boleh terjadi bersamaan, sehingga untuk menggunakan fungsi DOS pada program residen yang perlu kita lakukan ialah " Jangan menggunakan fungsi DOS bila fungsi DOS yang lain sedang aktif ". Bagaimana kita bisa mengetahui bahwa suatu fungsi DOS sedang aktif atau tidak ? Untuk itu anda bisa menggunakan kedua cara berikut:

1. Gunakan fungsi 34h dari interupsi 21h untuk mendapatkan InDOS Flag atau Bendera Aktif DOS(**BAD**). Untuk menggunakan fungsi ini, masukkan 34h pada AH, kemudian laksanakan interupsi 21h. Setelah interupsi dilaksanakan , pasangan register ES:BX akan mencatat alamat tempat BAD berada. BAD yang terdiri atas 1 byte akan bernilai nol(0) jika fungsi DOS tidak ada yang sedang aktif. Artinya pada keadaan ini kita bisa menggunakan fungsi DOS pada program residen dengan aman. Bila BAD bernilai lebih dari 0, artinya terdapat fungsi DOS yang sedang aktif. Dalam keadaan seperti ini kita tidak boleh menggunakan fungsi DOS pada program residen, karena akan meyebabkan komputer menjadi hang.
2. DOS mungkin saja dalam keadaan aman, walaupun BAD bernilai lebih dari

0(biasanya 1). Untuk menandakan keadaan aman ini DOS akan selalu mengaktifkan interupsi 28h. Handler asli dari interupsi 28h ini hanyalah berupa perintah IRET. Bila interupsi 28h ini diaktifkan, kita bisa menggunakan fungsi DOS dengan aman pada program residen. Interupsi 28h ini biasanya diaktifkan DOS pada saat DOS sedang menunggu masukan dari keyboard dengan fungsi 01h-0Ch. Untuk menggunakan interupsi ini, buatlah suatu handler baru untuknya.

Pada program 24.7. akan ditunjukkan bagaimana interupsi dari DOS, yaitu interupsi 21h fungsi 09h digunakan pada program residen. Dengan teknik yang sama, anda bisa menggunakan segala fungsi dari interupsi DOS dalam program residen tanpa perlu takut program anda menjadi hang.

```
Cetak_Pesan MACRO   Pesan           ; Mencetak kalimat dengan
            MOV     AH,09           ; interupsi dari DOS
            PUSH    CS              ;
            POP     DS              ; Samakan nilai CS dan DS
            LEA     DX,Pesan        ;
            INT     21h             ; Interupsi DOS
            ENDM

Ambil_Vec   MACRO   NoInt,Alamat
            MOV     AH,35h          ; Servis untuk mencari vektor
            MOV     AL,NoInt        ; No interupsi
            INT     21h             ; Laksanakan
            MOV     Alamat,BX       ; Offset
            MOV     Alamat[2],ES    ; Segment
            ENDM

;/============================================================\;
;          Program : BAD.ASM                                 ;
;          Author  : S'to                                    ;
;          Fungsi  : Menggunakan fungsi DOS pada program residen;
;                    Program ini akan menunjukkan bagaimana    ;
;                    memecahkan masalah reentrancy DOS, dengan  ;
;                    mengecek BAD (Bendera Aktif DOS)          ;
;\============================================================/;

         .MODEL SMALL
         .CODE
         ORG 100h

TData :  JMP              Res_kan
         PrtScr           EQU 05
         Addr_PrtScr_Asli LABEL DWORD
         Addr_PrtScr      DW  ?,?
         Pesan            DB ' Kalimat ini dicetak dengan fungsi'
                          DB ' dari DOS. ',13,10
                          DB '      Pemecahan masalah '
                          DB 'Reentrancy DOS !!!$'
         Offset_BAD       DW ?
         Segmen_BAD       DW ?

Res05    PROC
         PUSH   AX           ;
         PUSH   BX           ;
         PUSH   CX           ;
         PUSH   DX           ;
         PUSH   ES           ; Simpan isi semua register
         PUSH   DI           ;
```

```
          PUSH   DS            ;
          PUSH   SI            ;

          MOV    AX,Segmen_BAD
          MOV    ES,AX
          MOV    BX,Offset_BAD       ; ES:BX = alamat BAD
          CMP    BYTE PTR ES:[BX],0  ; Apakah ada fungsi DOS yang
                                     ; sedang aktif?
          JNE    Pulihkan          ; Ya, jangan lakukan
   Aman :                          ; interupsi DOS
          Cetak_Pesan Pesan        ; Tidak, lakukan interupsi DOS
Pulihkan:
          POP    SI            ;
          POP    DS            ;
          POP    DI            ;
          POP    ES            ;
          POP    DX            ; Kembalikan isi semua register
          POP    CX            ;
          POP    BX            ;
          POP    AX            ;
          IRET                 ; Akhir dari interupt handler
Res05     ENDP

Res_Kan :
          Ambil_Vec PrtScr,Addr_PrtScr

          MOV       AH,25h       ;
          MOV       AL,PrtScr    ; Membelokkan vektor interupsi
          LEA       DX,Res05     ;
          INT       21h          ;

          MOV       AH,34h         ;
          INT       21h            ; Ambil alamat InDOS flag
          MOV       Segmen_BAD,ES  ; atau BAD
          MOV       Offset_BAD,BX  ;

          LEA       DX,Res_Kan   ; Meresidenkan program Bag_Res
          INT       27h
END       TData
```

Program 24.7. Menggunakan fungsi DOS dalam Program Residen

Bila program 24.7. dijalankan, maka setiap kali terjadi penekanan terhadap tombol PrtScr, program akan melihat keadaan aman atau tidak untuk menggunakan interupsi DOS. Bila BAD bernilai nol atau keadaan aman, program akan mencetak kalimat dengan fungsi dari DOS. Pesan yang tercetak adalah:

Kalimat ini dicetak dengan fungsi dari DOS.

Pemecahan masalah Reentrancy DOS !!!

Bila BAD bernilai lebih dari nol atau keadaan tidak aman, program tidak akan menggunakan fungsi dari DOS dan akan segera keluar.

BAB XXV
GRAFIK

25.1. MODUS GRAFIK

Pada modus text, layar dibagi menjadi kotak-kotak yang membentuk karakter. Pada modus default Dos, layar terbagi menjadi 80 kotak horisontal dan 25 kotak vertical. Kita bisa saja membentuk suatu gambar pada modus teks, akan tetapi hasilnya tentu saja sangat kasar.

Pada modus grafik, layar dibagi menjadi titik-titik yang disebut sebagai Pixel. Untuk memrogram pada modus grafik ini, tentunya anda harus mengaktifkan mode layar grafik terlebih dahulu (Lihat 18.10 tentang mode layar). Misalkan anda mengaktifkan mode 13h, maka layar akan dibagi menjadi 320 X 200 pixel, atau sama dengan 64000 titik. Bila diaktifkan mode 06h, maka layar akan dibagi menjadi 640 X 200 pixel atau sama 128000 pixel. Tentunya pada mode 06 ini gambar akan tampak lebih halus. Anda harus ingat bahwa tidak semua mode didukung oleh monitor anda. Anda harus mengetahui dengan jelas jenis monitor dan modus apa saja yang didukungnya.

25.2. MENGGAMBAR SATU PIXEL

Bila anda menggunakan komputer, seperti Hercules dan Macintosh maka dengan mudah anda dapat menggambar lingkaran, garis, serta mewarnai gambar tersebut. Bagaimana pada komputer IBM PC dan kompatiblenya?

Pada komputer IBM PC dan kompatiblenya, kemampuan menggambar pada modus grafik hanyalah satu, yaitu menggambar pixel(titik). Kemampuan ini tampaknya sangatlah kurang, tetapi pada bagian ini akan kita lihat bagaimana menggunakan fasilitas ini untuk menggambar berbagai gambar yang menarik.

Untuk menggambar pixel ini aktifkanlah modus grafik terlebih dahulu. Setelah itu anda bisa menggambar pixel dengan fungsi 0Ch, dengan aturan pemakaian:

```
INPUT:
AH = 0Ch
AL = Atribut dari pixel. Jika bit ke 7-nya 1, maka pixel akan
     di Xor dengan gambar layar.
CX = Posisi kolom(X) tempat pixel akan digambar
DX = Posisi baris(Y) tempat pixel akan digambar
BH = Nomor halaman, jika modus video yang digunakan mempunyai
     halaman tampilan melebihi satu. Jika modus yang digunakan
```

hanya menggunakan 1 halaman tampilan, maka isi BH akan diabaikan.

Setelah semuanya anda persiapkan, laksanakanlah interupsi 10h. Contoh dari macro untuk menggambar pixel:

```
Pixel     MACRO   X,Y,Warna
          PUSH    AX
          PUSH    BX
          PUSH    CX
          PUSH    DX

          MOV     AH,0Ch
          MOV     CX,X        ; Posisi kolom atau X
          MOV     DX,Y        ; Posisi baris atau Y
          MOV     AL,Warna    ; Atribut Pixel
          INT     10h         ; Gambar pixel tersebut !

          POP     DX
          POP     CX
          POP     BX
          POP     AX
          ENDM
```

25.3. MENDAPATKAN INFORMASI WARNA PIXEL

Kebalikan dari fungsi 0Ch, fungsi 0Dh dari interupsi 10h digunakan untuk mendapatkan warna dari suatu pixel. Aturan untuk menggunakan fungsi ini adalah:

INPUT:

AH = 0Dh

CX = Posisi kolom(X) dari pixel

DX = Posisi baris(Y) dari pixel

BH = Nomor halaman, jika modus video yang digunakan mempunyai halaman tampilan melebihi satu. Jika modus yang digunakan hanya menggunakan 1 halaman tampilan, maka isi BH akan diabaikan.

OUTPUT:

AL = Atribut dari pixel pada kolom CX dan baris DX.

25.4. MENGGAMBAR GARIS LURUS

Bila suatu benda dibagi-bagi terus, maka akan anda dapatkan apa yang dinamakan sebagai atom, atau bagian terkecil dari suatu benda. Demikian halnya pada gambar, bila dilihat secara seksama, setiap gambar terbentuk atas titik-titik. Makin banyak titik yang membentuk suatu gambar, makin haluslah gambar tersebut. Dengan prinsip yang sama, kita bisa membuat bermacam gambar yang menarik.

- Untuk menggambar garis vertical maupun horisontal adalah cukup mudah. Anda hanya perlu menggambar titik-titik secara berurutan untuk menghasilkan sebuah garis.
- Untuk menggambar garis Vertical ke bawah, anda hanya perlu menambah posisi baris(Y) dengan posisi kolom(X) yang tetap. Sedangkan untuk menggambar garis Horisontal ke kanan, anda hanya perlu menambah posisi kolom(X) dengan posisi baris(Y) yang tetap(Lihat program 25.1).

```
Readkey     MACRO
            MOV     AH,00
            INT     16h
            ENDM


SetCRT      MACRO   Mode
            MOV     AH,00
            MOV     AL,Mode
            INT     10h
            ENDM


PutPixel    MACRO       X,Y,Warna
            PUSH        AX
            PUSH        BX
            PUSH        CX
            PUSH        DX
            MOV         AH,0C     ; Servis menggambar pixel
            MOV         CX,X      ; Posisi kolom atau X
            MOV         DX,Y      ; Posisi baris atau Y
            MOV         AL,Warna  ; Atribut Pixel
            INT         10h       ; Gambar pixel tersebut !
            POP         DX
            POP         CX
            POP         BX
            POP         AX
            ENDM


GarisV      MACRO       X1,Y1,Panjang,Warna
            LOCAL       Ulang
            PUSH        DX
            PUSH        CX
            MOV         DX,Y1
            MOV         CX,Panjang
    Ulang:
            PutPixel    X1,DX,Warna
            INC         DX
```

```
            LOOP        Ulang
            POP         CX
            POP         DX
            ENDM

GarisH      MACRO       X1,Y1,Panjang,Warna
            LOCAL       Ulang
            PUSH        CX
            PUSH        DX
            MOV         DX,X1
            MOV         CX,Panjang
    Ulang:
            PutPixel    DX,Y1,Warna
            INC         DX
            LOOP        Ulang
            POP         DX
            POP         CX
            ENDM

;/=============================================\;
;           Program : GRAPH0.ASM                ;
;           Author  : S'to                      ;
;           Fungsi  : Menggambar garis vertical ;
;                     dan horisontal            ;
;\=============================================/;

        .MODEL  SMALL
        .CODE
        ORG 100h

Proses:
        SetCRT   13h             ; Aktifkan mode grafik 13h
        GarisV   150,50,50,12    ; Gambar garis Vertikal
        Readkey                  ; Tunggu penekanan keyboard
        GarisH   135,60,30,12    ; Gambar garis Horisontal
        Readkey                  ; Tunggu penekanan keyboard

        SetCRT   03h             ; Kembali pada mode Dos
        INT      20h
END     Proses
```

Program 25.1. Menggambar garis Vertical dan Horisontal

Bila program 25.1. dieksekusi, maka pada layar akan ditampilkan sebuah gambar salib yang terdiri atas garis vertical dan horisontal.

<<<< Gbr251.PIX >>>>

Gambar 25.1. Hasil eksekusi program 25.1.

25.5. MENGGAMBAR GARIS MIRING

Untuk menggambar sebuah garis miring, prinsip yang digunakan tidaklah jauh berbeda dengan menggambar garis lurus. Untuk menggambar sebuah garis miring 45 derajat ke kiri bawah, anda hanya perlu menggambar pixel sambil

mengurangi posisi kolom(X) dan menambah posisi baris(Y). Sedangkan untuk menggambar sebuah garis miring 45 derajat ke kanan bawah, anda bisa menggambar pixel sambil menambahi posisi kolom(X) dan menambah posisi baris(Y).

```
Readkey     MACRO
            MOV     AH,00
            INT     16h
            ENDM

SetCRT      MACRO   Mode
            MOV     AH,00
            MOV     AL,Mode
            INT     10h
            ENDM

PutPixel    MACRO   X,Y,Warna
            PUSH    AX
            PUSH    BX
            PUSH    CX
            PUSH    DX
            MOV     AH,12       ; Servis menggambar pixel
            MOV     CX,X        ; Posisi kolom atau X
            MOV     DX,Y        ; Posisi baris atau Y
            MOV     AL,Warna    ; Atribut Pixel
            INT     10h         ; Gambar pixel tersebut !
            POP     DX
            POP     CX
            POP     BX
            POP     AX
            ENDM

M_Kanan     MACRO     X1,Y1,Panjang,Warna
            LOCAL     Ulang
            PUSH      BX
            PUSH      CX
            PUSH      DX
            MOV       DX,X1
            MOV       BX,Y1
            MOV       CX,Panjang
      Ulang:
            PutPixel  DX,BX,Warna
            INC       DX
            INC       BX
            LOOP      Ulang
            POP       DX
            POP       CX
            POP       BX
            ENDM

M_Kiri      MACRO     X1,Y1,Panjang,Warna
            LOCAL     Ulang
            PUSH      BX
            PUSH      CX
            PUSH      DX
            MOV       DX,X1
            MOV       BX,Y1
            MOV       CX,Panjang
      Ulang:
            PutPixel  DX,BX,Warna
            DEC       DX
```

```
            INC       BX
            LOOP      Ulang
            POP       DX
            POP       CX
            POP       BX
            ENDM

;/=================================================\;
;         Program : GRAPH1.ASM                     ;
;         Author  : S'to                           ;
;         Fungsi  : Menggambar garis miring 45 derajat ;
;                   ke kiri dan kanan              ;
;\=================================================/;

        .MODEL  SMALL
        .CODE
        ORG 100h
Proses:
        SetCRT   13h
        M_Kiri   150,0,50,83        ; Garis miring kiri
        Readkey
        M_Kanan  150,0,50,83        ; Garis miring kanan
        Readkey

        SetCRT   03h
        INT      20h
END     Proses
```

Program 25.2. Menggambar garis miring

Bila program 25.2. dieksekusi, maka pada layar akan ditampilkan sebuah gambar atap rumah yang terdiri atas garis miring kekanan dan kiri.

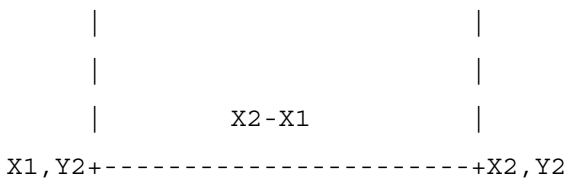
<<<< Gbr252.PIX >>>>

Gambar 25.2. Hasil eksekusi program 25.2.

25.6. MENGGAMBAR KOTAK

Sebuah kotak terdiri atas 2 garis vertical dan 2 garis horisontal. Untuk itu anda bisa menggunakan macro dari GarisV dan GarisH untuk menggambar kotak ini. Kita hanya menentukan posisi
X1,Y1 dan X2,Y2(Gambar 25.3) untuk menggambar sebuah kotak. Perhatikan, bahwa X2 > X1 dan Y2 > Y1.

```
X1,Y1+-----------------------+X2,Y1
     |        X2-X1          |
     |                        |
     |                        |
     |Y2-Y1            Y2-Y1|
```



Gambar 25.3. Teknik menggambar kotak

Dari gambar 25.3. dapat anda lihat, dengan mudah sebuah kotak dapat digambar dengan bantuan macro untuk menggambar garis vertikal dan horisontal.

```
Readkey     MACRO
            MOV     AH,00
            INT     16h
            ENDM

SetCRT      MACRO   Mode
            MOV     AH,00
            MOV     AL,Mode
            INT     10h
            ENDM

PutPixel    MACRO   X,Y,Warna
            PUSH    AX
            PUSH    BX
            PUSH    CX
            PUSH    DX
            MOV     AH,0C       ; Servis menggambar pixel
            MOV     CX,X        ; Posisi kolom atau X
            MOV     DX,Y        ; Posisi baris atau Y
            MOV     AL,Warna    ; Atribut Pixel
            INT     10h         ; Gambar pixel tersebut !
            POP     DX
            POP     CX
            POP     BX
            POP     AX
            ENDM

GarisV      MACRO     X1,Y1,Panjang,Warna
            LOCAL     Ulang
            PUSH      DX
            PUSH      CX
            MOV       DX,Y1
            MOV       CX,Panjang
      Ulang:
            PutPixel  X1,DX,Warna
            INC       DX
            LOOP      Ulang
            POP       CX
            POP       DX
            ENDM
```

```
GarisH      MACRO       X1,Y1,Panjang,Warna
            LOCAL       Ulang
            PUSH        CX
            PUSH        DX
            MOV         DX,X1
            MOV         CX,Panjang
      Ulang:
            PutPixel    DX,Y1,Warna
            INC         DX
            LOOP        Ulang
            POP         DX
            POP         CX
            ENDM


Kotak       MACRO     X1,Y1,X2,Y2,Warna
            GarisH    X1,Y1,X2-X1,Warna
            GarisV    X1,Y1,Y2-Y1,Warna
            GarisV    X2,Y1,Y2-Y1,Warna
            GarisH    X1,Y2,X2-X1+1,Warna
            ENDM

;/=============================================\;
;          Program : GRAPH2.ASM                ;
;          Author  : S'to                      ;
;          Fungsi  : Menggambar sebuah kotak   ;
;\=============================================/;

            .MODEL  SMALL
            .CODE
            ORG 100h
Proses:
         SetCRT   13h
         Kotak    120,30,180,100,12  ; Gambar kotak
         Readkey

         SetCRT   03h
         INT      20h
END      Proses
```

Program 25.3. Menggambar Kotak

Bila program 25.3. dieksekusi, maka pada layar akan ditampilkan sebuah gambar kotak persegi empat(Gambar 25.4).

<<<< Gbr254.PIX >>>>

Gambar 25.4. Hasil eksekusi program 25.3.

25.7. MEWARNAI KOTAK

Pada program 25.3., kita hanya sekedar menggambar sebuah kotak tanpa warna dasar. Untuk memberi warna pada kotak kita harus menggambar pixel-pixel pada seluruh daerah kotak yang kosong. Dengan demikian kotak akan menjadi berwarna.

Untuk menggambar daerah kotak ini, bisa anda gunakan bermacam-maca cara, misalkan dengan menggambarkan garis vertical ataupun garis horisontal. Pada program 25.4. kita akan mewarnai daerah yang kosong pada kotak dengan dengan garis-garis Horisontal.

```
Readkey     MACRO
            MOV     AH,00
            INT     16h
            ENDM

SetCRT      MACRO   Mode
            MOV     AH,00
            MOV     AL,Mode
            INT     10h
            ENDM

PutPixel    MACRO   X,Y,Warna
            PUSH    AX
            PUSH    BX
            PUSH    CX
            PUSH    DX
            MOV     AH,12       ; Servis menggambar pixel
            MOV     CX,X        ; Posisi kolom atau X
            MOV     DX,Y        ; Posisi baris atau Y
            MOV     AL,Warna    ; Atribut Pixel
            INT     10h         ; Gambar pixel tersebut !
            POP     DX
            POP     CX
            POP     BX
            POP     AX
            ENDM

GarisV      MACRO     X1,Y1,Panjang,Warna
            LOCAL     Ulang
            PUSH      DX
            PUSH      CX
            MOV       DX,Y1
            MOV       CX,Panjang
      Ulang:
            PutPixel  X1,DX,Warna
            INC       DX
            LOOP      Ulang
            POP       CX
            POP       DX
            ENDM

GarisH      MACRO      X1,Y1,Panjang,Warna
            LOCAL      Ulang
            PUSH       CX
            PUSH       DX
            MOV        DX,X1
            MOV        CX,Panjang
      Ulang:
            PutPixel   DX,Y1,Warna
            INC        DX
            LOOP       Ulang
            POP        DX
            POP        CX
            ENDM
```

```
Kotak       MACRO    X1,Y1,X2,Y2,Warna
            GarisH   X1,Y1,X2-X1,Warna
            GarisV   X1,Y1,Y2-Y1,Warna
            GarisV   X2,Y1,Y2-Y1,Warna
            GarisH   X1,Y2,X2-X1+1,Warna
            ENDM

KotakW      MACRO     X1,Y1,X2,Y2,Warna
            LOCAL     Ulang1,Ulang2
            PUSH      AX
            PUSH      CX
            MOV       AX,Y1+1
            MOV       CX,Y2-Y1-1
    Ulang1:
            GarisH    X1+1,AX,X2-X1-1,Warna
            INC       AX
            LOOP      Ulang1
            POP       CX
            POP       AX
            ENDM


;/=============================================\;
;         Program : GRAPH3.ASM                  ;
;         Author  : S'to                        ;
;         Fungsi  : Menggambar dan mewarnai kotak ;
;\=============================================/;

        .MODEL  SMALL
        .CODE
        ORG 100h
Proses:
        SetCRT   13h
        Kotak    120,30,180,100,12  ; Gambar kotak
        Readkey
        KotakW   120,30,180,100,09  ; Warnai kotak
        Readkey

        SetCRT   03h
        INT      20h
END     Proses
```

Program 25.4. Mewarnai Kotak

Bila program 25.4. dieksekusi, maka pada layar akan ditampilkan sebuah gambar kotak yang telah diwarnai (Gambar 25.5).

<<<<< Gbr255.PIX >>>>>

Gambar 25.5. Hasil eksekusi program 25.4.

Pada program 25.4. ini, kotak akan diwarnai seluruhnya. Cobalah anda membuat kreasi sendiri, misalkan kotak akan diwarnai dengan garis-garis vertical, garis-garis harisontal atau kotak- kotak kecil.

25.8. MENGGAMBAR HELIKOPTER

Pada program sebelumnya, kita selalu menggambar bentuk gambar yang linear. Kini, bila anda ingin menggambar sebuah gambar tak tentu, seperti manusia, tengkorak, tank, bunga atau helikopter, dengan rumus adalah tidak mungkin.

Untuk itu salah satu cara yang praktis adalah membentuk suatu tabel gambar. Dari tabel ini kemudian anda lihat secara perBITnya. Bila bit pada data gambar bernilai satu, maka gambarlah sebuah pixel, sebaliknya bila Bit pada data gambar bernilai nol maka pixel tidak digambar. Setelah itu pindahkan posisi X(Kolom) dan test bit berikutnya.

Dengan cara demikian anda bisa membuat gambar dalam ukuran yang berapapun, sesuai resolusi monitor anda. Pada program 25.5. akan ditunjukkan, bagaimana membuat sebuah gambar helikopter dengan ukuran 32 bit X 32 bit.

```
Readkey MACRO               ; Untuk menunggu masukan dari
        MOV     AH,00       ; Keyboard
        INT     16h
        ENDM

SetCRT  MACRO   Mode        ; Untuk merubah mode layar
        MOV     AH,00       ;
        MOV     AL,Mode     ;
        INT     10h
        ENDM


Pixel   MACRO   X,Y,Warna   ; Untuk menggambar pixel
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DX

        MOV     AH,12        ; Servis menggambar pixel
        MOV     CX,X         ; Posisi kolom atau X
        MOV     DX,Y         ; Posisi baris atau Y
        MOV     AL,Warna     ; Atribut Pixel
        INT     10h          ; Gambar pixel tersebut !

        POP     DX
        POP     CX
        POP     BX
        POP     AX
        ENDM

;/=================================================\;
;            Program : ANIMATE1.ASM                 ;
;            Author  : S'to                         ;
;            Fungsi  : Menggambar helikopter        ;
;\=================================================/;

        .MODEL  SMALL
        .CODE
        ORG 100h

TData : JMP     Proses
        Gambar  DW 0000000000000000b,0000000000000000b
```

```
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000001110000000b
                DW 0000000000000000b,0000001100000000b
                DW 0000000011111111b,1111111111111110b
                DW 0000000000000000b,0000001100000000b
                DW 0000000000000000b,0111111111000000b
                DW 1110000000000000b,1111111111100000b
                DW 0100000000111111b,1111001100110000b
                DW 0111111111111111b,1111001100011000b
                DW 0000000000000011b,1111001111111000b
                DW 0000000000000000b,0111111111100000b
                DW 0000000000000000b,0010000100001000b
                DW 0000000000111111b,1111111111110000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
        PosX    DW 100          ; Posisi awal X
        PosY    DW 30           ; Posisi awal Y

Proses:
        SetCRT  13h             ; Aktifkan mode grafik

        SUB     BX,BX           ;
        MOV     CX,32           ; CX=banyaknya baris
Ulang1:
        PUSH    CX
        MOV     CX,2            ; CX=banyaknya Word dalam 1 baris
Ulang2:
        PUSH    CX
        MOV     CX,16           ; CX=Banyaknya bit dalam 1 word
        MOV     AX,1000000000000000b
Ulang3:
        PUSH    AX
        AND     AX,Gambar[BX]   ; Test bit Gambar yang ke AX
        JZ      Nol             ; Jika nol, lompat
        Pixel   PosX,PosY,83    ; Jika tidak, gambar pixel
   Nol:
        POP     AX              ;
        SHR     AX,1            ;
        INC     PosX            ; Tambah posisi X
        LOOP    Ulang3          ; Test bit Gambar berikutnya

        ADD     BX,2            ; Akses word berikutnya
        POP     CX
        LOOP    Ulang2          ; Test word berikutnya
        INC     PosY            ;
        SUB     PosX,32         ; Kembalikan posisi X mula-mula
        POP     CX
        LOOP    Ulang1          ; Test word pada baris berikutnya
  Exit:
        Readkey
        SetCRT  03h             ; Aktifkan Mode default Dos
```

```
        INT     20h
END     TData
```

Program 25.5. Menggambar Helikopter

Bila program 25.5. dieksekusi, maka pada layar akan ditampilkan sebuah gambar helikopter (Gambar 25.6). Helikopter ini digambar berdasarkan data gambar pada variabel "Gambar".

<<<<< Gbr256.PIX >>>>>

Gambar 25.6. Hasil eksekusi program 25.5.

Catatan:

Dengan teknik yang tidak jauh berbeda, anda bisa membuat program yang dapat menampilkan bermacam format gambar, seperti GIF, PIX, BMP, dan sebagainya.

25.9. TERBANGKAN HELIKOPTER ANDA

Pada grafik, yang paling menarik adalah membuat sebuah animasi. Seperti dinosaurus yang sedang berjalan, bunga yang sedang berkembang, pesawat yang meledak dan sebagainya. Dibalik pembuatan animasi ini terdapat berpuluh-puluh cara yang dapat digunakan.

Salah satu cara yang paling praktis dan mudah, walaupun tidak begitu bagus adalah dengan teknik gambar hapus. Yaitu animasi dengan cara menggambar sebuah gambar, kemudian dihapus dan digambar lagi pada posisi atau bentuk gambar yang berbeda. Dengan cara ini sebuah gambar akan tampak seperti sedang bergerak (Program 25.6).

```
Delay   MACRO
        LOCAL Ulang
        PUSH  CX
        SUB      CX,CX
  Ulang:
        LOOP    Ulang
        POP  CX
        ENDM

SetCRT  MACRO   Mode
        MOV     AH,00
        MOV     AL,Mode
        INT     10h
        ENDM
```

```
Pixel   MACRO   X,Y,Warna
        PUSH AX
        PUSH BX
        PUSH CX
        PUSH DX
        MOV     AH,12       ; Servis menggambar pixel
        MOV     CX,X        ; Posisi kolom atau X
        MOV     DX,Y        ; Posisi baris atau Y
        MOV     AL,Warna    ; Atribut Pixel
        INT     10h         ; Gambar pixel tersebut !
        POP  DX
        POP  CX
        POP  BX
        POP  AX
        ENDM

Heli    MACRO   Gambar,Warna
        LOCAL   Ulang1,Ulang2,Ulang3,Nol
        PUSH    AX              ;
        PUSH    BX              ; Simpan semua register yang
        PUSH    CX              ;        digunakan
        PUSH    DX              ;

        SUB     BX,BX           ;
        MOV     CX,32           ; CX = banyaknya baris
Ulang1:
        PUSH    CX
        MOV     CX,2            ; CX = banyaknya Word satu baris
Ulang2:
        PUSH    CX
        MOV     CX,16           ; CX = Banyaknya bit pada 1 word
        MOV     AX,1000000000000000b
Ulang3:
        PUSH    AX
        AND     AX,Gambar[BX]   ; Apakah bit gambar ke AX=1 ?
        JZ      Nol             ; Tidak, lompat
        Pixel   PosX,PosY,Warna ; Ya, gambar pixel
   Nol:
        POP     AX              ;
        SHR     AX,1            ; Untuk men-test bit Gambar
        INC     PosX            ;
        LOOP    Ulang3          ; Ulangi test bit berikutnya

        ADD     BX,2            ; Untuk mengakses word berikutnya
        POP     CX              ;
        LOOP    Ulang2          ; Ulangi test word berikutnya

        INC     PosY            ; Tambah posisi Y
        SUB     PosX,32         ; Kembalikan posisi X mula-mula
        POP     CX              ;
        LOOP    Ulang1          ; Test word pada baris berikutnya

        SUB     PosY,32         ; Kembalikan posisi Y mula-mula

        POP     DX              ;
        POP     CX              ; Ambil kembali semua nilai
        POP     BX              ;  register yang disimpan
        POP     AX              ;
        ENDM

;/================================================\;
;          Program : ANIMATE2.ASM                 ;
;          Author  : S'to                         ;
;          Fungsi  : Membuat animasi helikopter yang ;
;                    sedang terbang               ;
;\================================================/;
        .MODEL  SMALL
```

```
        .CODE
        ORG 100h

TData : JMP     Proses
        Heli1   DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000001110000000b
                DW 0000000000000000b,0000000100000000b
                DW 0000000011111111b,1111111111111110b
                DW 0000000000000000b,0000000100000000b
                DW 0000000000000000b,0111111111000000b
                DW 1110000000000000b,1111111111100000b
                DW 0100000000111111b,1111000100110000b
                DW 0111111111111111b,1111000100011000b
                DW 0000000000000011b,1111000111111000b
                DW 0000000000000000b,0111111111100000b
                DW 0000000000000000b,0010000100001000b
                DW 0000000000111111b,1111111111110000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b

        PosX    DW 00       ; Posisi awal X
        PosY    DW 50       ; Posisi awal Y

Proses:
        SetCRT  13h         ; Aktifkan mode video grafik

        MOV     CX,0
 Ulang:
        Heli    Heli1,50    ; Gambar heli dengan warna 50
        Delay               ;
        Heli    Heli1,00    ; Hapus heli dengan warna hitam
        INC     PosX        ; Tambah posisi X

        INC     CX          ;
        CMP     CX,290      ; Ulangi sebanyak 290 kali
        JE      Exit        ;
        JMP     Ulang       ;
Exit:
        SetCRT  03h         ; Kemabli ke mode video default dos
        INT     20h
END     TData
```

Program 25.6. Animasi Helikopter yang sedang terbang

Bila program 25.6. dieksekusi, maka pada layar akan ditampilkan sebuah gambar helikopter (Gambar 25.6) yang akan terbang melintasi monitor anda.

25.10. ANIMASI DENGAN HALAMAN LAYAR

Pada bagian ini, akan kita lihat teknik lain dalam pembuatan suatu animasi. Seperti yang telah anda ketahui, dalam suatu modus mungkin saja terdapat beberapa halaman layar. Halaman layar ini bisa kita manfaatkan untuk pembuatan suatu animasi.

Untuk itu, gambarlah bentuk gambar yang diinginkan pada masing-masing halaman layar. Setelah itu anda tinggal mengaktifkan halaman layar untuk mendapatkan suatu efek gerakan.

```
Readkey MACRO             ; Macro untuk menuggu
        PUSH    AX        ; penekanan tombol keyboard
        MOV     AH,00
        INT     16h
        POP     AX
        ENDM

Ak_Page MACRO No          ; Macro ini digunakan untuk
        MOV     AH,5      ; mengaktifkan halaman layar
        MOV     AL,No
        INT     10h
        ENDM

SetCRT  MACRO   Mode      ; Macro untuk mengganti mode layar
        MOV     AH,00
        MOV     AL,Mode
        INT     10h
        ENDM

Pixel   MACRO   X,Y,Warna,Hlm
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DX
        MOV     AH,12       ; Servis menggambar pixel
        MOV     CX,X        ; Posisi kolom atau X
        MOV     DX,Y        ; Posisi baris atau Y
        MOV     AL,Warna    ; Atribut Pixel
        MOV     BH,Hlm      ; Halaman layar
        INT     10h         ; Gambar pixel tersebut !
        POP     DX
        POP     CX
        POP     BX
        POP     AX
        ENDM

Hallo   MACRO   Gambar,Warna,Hlm
        LOCAL   Ulang1,Ulang2,Ulang3,Nol
        PUSH    AX              ;
        PUSH    BX              ; Simpan semua register yang
        PUSH    CX              ;        digunakan
        PUSH    DX              ;

        SUB     BX,BX           ;
        MOV     CX,32           ; CX = banyaknya baris
Ulang1:
        PUSH    CX
        MOV     CX,2            ; CX = banyaknya Word satu baris
Ulang2:
```

```
        PUSH    CX
        MOV     CX,16           ; CX = Banyaknya bit pada 1 word
        MOV     AX,1000000000000000b
Ulang3:
        PUSH    AX
        AND     AX,Gambar[BX]   ; Apakah bit gambar ke AX=1 ?
        JZ      Nol             ; Tidak, lompat
        Pixel   PosX,PosY,Warna,Hlm ; Ya, gambar pixel
   Nol:
        POP     AX              ;
        SHR     AX,1            ; Untuk men-test bit Gambar
        INC     PosX            ;
        LOOP    Ulang3          ; Ulangi test bit berikutnya

        ADD     BX,2            ; Untuk mengakses word berikutnya
        POP     CX              ;
        LOOP    Ulang2          ; Ulangi test word berikutnya

        INC     PosY            ; Tambah posisi Y
        SUB     PosX,32         ; Kembalikan posisi X mula-mula
        POP     CX              ;
        LOOP    Ulang1          ; Test word pada baris berikutnya

        SUB     PosY,32         ; Kembalikan posisi Y mula-mula

        POP     DX              ;
        POP     CX              ; Ambil kembali semua nilai
        POP     BX              ;  register yang disimpan
        POP     AX              ;
        ENDM

;/=====================================================\;
;          Program : ANIMATE3.ASM                      ;
;          Author  : S'to                              ;
;          Fungsi  : Membuat animasi dengan memanfaatkan ;
;                    halaman layar                     ;
;\=====================================================/;

        .MODEL  SMALL
        .CODE
        ORG 100h

TData : JMP     Proses
        Hall1   DW 0000000000000001b,1000000000000000b
                DW 0000000000000011b,1000000000000000b
                DW 0000000000000110b,0000000000000000b
                DW 0000000000000111b,1000000000000000b
                DW 0000000001111100b,1111110000000000b
                DW 0000000110000100b,1000001100000000b
                DW 0000001000001000b,0100000010000000b
                DW 0000010000100000b,0011000001100000b
                DW 0000100000100000b,0000100000010000b
                DW 0001000000000000b,0000000000001000b
                DW 0010000110000000b,0000000011000100b
                DW 0100001111000000b,0000011111000100b
                DW 0100000111100000b,0000011110000010b
                DW 0100000110111000b,0001101100000010b
                DW 0100000001111110b,0011111000000001b
                DW 0100000000000000b,0000000000000001b
                DW 0100000000000001b,1000000000000010b
                DW 0100000000000011b,1100000000000010b
                DW 0100000000000010b,1100000000000010b
                DW 0010000100000000b,0000000010000010b
                DW 0010000111000000b,0000001110000010b
                DW 0010000011111100b,1100111100000100b
                DW 0001000011111100b,1100111100000100b
                DW 0001000001111111b,1111111000000100b
```

```
        DW 0000100000011111b,1111100000001000b
        DW 0000010000000111b,1110000000010000b
        DW 0000001000000000b,0000000001100000b
        DW 0000000110000000b,0000001110000000b
        DW 0000000001110000b,1000010000000000b
        DW 0000000000001111b,0111100000000000b
        DW 0000000000000000b,0000000000000000b
        DW 0000000000000000b,0000000000000000b

Hall2   DW 0000000000000011b,0000000000000000b
        DW 0000000000000111b,0000000000000000b
        DW 0000000000000110b,0000000000000000b
        DW 0000000000000111b,1000000000000000b
        DW 0000000001111100b,1111110000000000b
        DW 0000000110000100b,1000001100000000b
        DW 0000001000001000b,0100000010000000b
        DW 0000010000100000b,0011000001100000b
        DW 0000100000100000b,0000100000010000b
        DW 0001000000000000b,0000000000001000b
        DW 0010000100000000b,0000000010000100b
        DW 0100001110000000b,0000011100000100b
        DW 0100000111000000b,0000011100000010b
        DW 0100000111100000b,0001111000000010b
        DW 0100000001111000b,0011100000000001b
        DW 0100000000000000b,0000000000000001b
        DW 0100000000000001b,1000000000000010b
        DW 0100000000000011b,1100000000000010b
        DW 0100000000000010b,1100000000000010b
        DW 0010000000000000b,0000000000000010b
        DW 0010000000000000b,0000000000000010b
        DW 0010000010000000b,0000000100000100b
        DW 0001000011100000b,0000111100000100b
        DW 0001000001111101b,1111111000000100b
        DW 0000100000011111b,1111100000001000b
        DW 0000010000000111b,1110000000010000b
        DW 0000001000000000b,0000000001100000b
        DW 0000000110000000b,0000001110000000b
        DW 0000000001110000b,1000010000000000b
        DW 0000000000001111b,0111100000000000b
        DW 0000000000000000b,0000000000000000b
        DW 0000000000000000b,0000000000000000b

Hall3   DW 0000000000111100b,0000000000000000b
        DW 0000000000001111b,0000000000000000b
        DW 0000000000000110b,0000000000000000b
        DW 0000000000000111b,1000000000000000b
        DW 0000000001111100b,1111110000000000b
        DW 0000000110000100b,1000001100000000b
        DW 0000001000001000b,0100000010000000b
        DW 0000010000100000b,0011000001100000b
        DW 0000100000100000b,0000100000010000b
        DW 0001000000000000b,0000000000001000b
        DW 0010000000000000b,0000000000000100b
        DW 0100000000000000b,0000001100000100b
        DW 0100000100000000b,0000011000000010b
        DW 0100000111000000b,0000110000000010b
        DW 0100000001111100b,0001100000000001b
        DW 0100000000000000b,0000000000000001b
        DW 0100000000000001b,1000000000000010b
        DW 0100000000000011b,1100000000000010b
        DW 0100000000000010b,1100000000000010b
        DW 0010000000000000b,0000000000000010b
        DW 0010000000000000b,0000000000000010b
        DW 0010000011000000b,0000001000000100b
        DW 0001000011111000b,0100111000000100b
        DW 0001000000011100b,0111111000000100b
        DW 0000100000001111b,1111100000001000b
```

```
                DW 0000010000000000b,0000000000010000b
                DW 0000001000000000b,0000000001100000b
                DW 0000000110000000b,0000001110000000b
                DW 0000000001110000b,1000010000000000b
                DW 0000000000001111b,0111100000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b

        Hall4   DW 0000000001110000b,0000000000000000b
                DW 0000000011011110b,0000000000000000b
                DW 0000000000000111b,0000000000000000b
                DW 0000000000000111b,1000000000000000b
                DW 0000000001111100b,1111110000000000b
                DW 0000000110000100b,1000001100000000b
                DW 0000001000001000b,0100000010000000b
                DW 0000010000100000b,0011000001100000b
                DW 0000100000100000b,0000100000010000b
                DW 0001000000000000b,0000000000001000b
                DW 0010000000000000b,0000000000000100b
                DW 0100000000000000b,0000000000000100b
                DW 0100000000000000b,0000000000000010b
                DW 0100000000000000b,0000000000000010b
                DW 0100000000000000b,0000000000000001b
                DW 0100000000000000b,0000000000000001b
                DW 0100000000000001b,1000000000000010b
                DW 0100000000000011b,1100000000000010b
                DW 0100000000000010b,1100000000000010b
                DW 0010000000000000b,0000000000000010b
                DW 0010000000000000b,0000000000000010b
                DW 0010000000000000b,0000000000000100b
                DW 0001000000000000b,0000000000000100b
                DW 0001000000000000b,0000000000000100b
                DW 0000100000000000b,1000000000001000b
                DW 0000010000000100b,0010000000010000b
                DW 0000001000000011b,1100000001100000b
                DW 0000000110000000b,0000001110000000b
                DW 0000000001110000b,1000010000000000b
                DW 0000000000001111b,0111100000000000b
                DW 0000000000000000b,0000000000000000b
                DW 0000000000000000b,0000000000000000b

        PosX    DW 150          ; Posisi awal X
        PosY    DW 70           ; Posisi awal Y

Proses:
        SetCRT  13              ; Aktifkan mode video grafik

        Hallo   Hall1,20,0      ; Gambar hall1 pada halaman 0
        Hallo   Hall2,21,1      ; Gambar hall2 pada halaman 1
        Hallo   Hall3,22,2      ; Gambar hall3 pada halaman 2
        Hallo   Hall4,23,3      ; Gambar hall4 pada halaman 3

        MOV     AL,4
Ulang1:
        DEC     AL
        Ak_Page AL              ; Aktifkan halaman layar ke AL
        Readkey
        CMP     AL,0
        JNE     Ulang1
Exit:
        SetCRT  03h             ; Kemabli ke mode video default dos
        INT     20h
END     TData
```

Program 25.7. Animasi dengan halaman layar

Bila program 25.7. dieksekusi, maka pada layar akan ditampilkan gambar pumpkin yang akan berubah mimik wajahnya setiap ditekan sembarang tombol(Gambar 25.6).

<<<<< Gbr257.PIX >>>>>

Gambar 25.7. Hasil eksekusi program 25.7.

Pada awal program kita mangaktifkan modus grafik 13 yang mempunyai halaman tampilan sebanyak 4 buah(0, 1, 2 dan 3). Karena modus yang digunakan mempunyai halaman tampilan lebih dari satu, maka pada macro yang menggambar Pixel, ditambahkan register BH yang berisi nomor halaman yang akan digambari pixel.

BAB XXVI

MEMBUAT PROGRAM EXE

26.1. PROGRAM EXE

Seperti program COM, program EXE juga merupakan suatu bentuk program yang dapat langsung dijalankan pada prompt DOS. Bentuk program EXE tidaklah dibatasi oleh satu segment, seperti halnya pada program COM. Oleh karenanya besarnya file untuk program EXE bisa melebihi 64 KB.

Program EXE merupakan program yang lebih lengkap dibandingkan dengan program COM, selain itu penggunaan memory juga lebih mudah pada program EXE.

Catatan:

Pada program EXE, segala bentuk JUMP dan CALL digunakan jenis FAR(Program COM berbentuk NEAR). Hal ini dikarenakan program EXE yang bisa mengakses segment lain, sehingga perintah JUMP dan CALL membutuhkan informasi alamat dari segment(CS) selain offset(IP).

26.2. MODEL PROGRAM EXE

Pada program COM, kita tidak perlu mendefinisikan tempat tertentu untuk segment DATA dan STACK karena program COM hanya menggunakan 1 segment. Dengan demikian segment untuk DATA, STACK dan CODE pada program COM adalah sama, stack akan menggunakan akhir dari segment yang digunakan oleh segment CODE.

Berbeda dengan program COM, pada program EXE anda harus mendefinisikan tempat untuk segment DATA, CODE dan STACK. Untuk membuat program EXE ini, anda bisa menggunakan model pada gambar 26.1.

```
-------------------------------------------------------------
          .MODEL   SMALL
          .STACK   200h
          .DATA
           +--------------+
           |    Tempat    |
           | Data Program |
           +--------------+
          .CODE
   Label1:
           MOV    AX,@DATA
           MOV    DS,AX
           +---------------+
```

```
            |               |
            |    Tempat     |
            |    Program    |
            |               |
            +---------------+
            MOV    AX,4C00h
            INT    21h
     END    Label1
```

Gambar 26.1. Model program EXE

Pada program EXE, kita tidak perlu menggunakan perintah:ORG 100h, karena program EXE bisa menempatkan dirinya pada alamat yang telah ditentukan.

Pada program EXE, register segment CS dan SS diinialisasi secara otomatis, tetapi register segment DS dan ES tidaklah demikian. Register segment DS dan ES pada awalnya menunjuk pada awal program, yaitu PSP. Karenanya register DS perlu kita inialisasi secara manual agar menunjuk pada segment data melalui perintah:

```
MOV   AX,@DATA
MOV   DS,AX
```

Pada program EXE, kita perlu mendefinisikan tempat untuk stack. Pendefinisian tempat untuk stack digunakan tanda directive: .STACK yang diikuti dengan banyaknya stack yang didefinisikan untuk program. Pada model yang kita gunakan didefinisikan tempat untuk stack sebanyak 200h Word yang tentunya sudah lebih dari cukup untuk digunakan oleh program-program pada umumya.

Berbeda dengan program COM, yang selalu kita akhiri dengan interupsi 20h, pada program EXE interupsi 20h tidak bisa digunakan. Pada program EXE digunakan interupsi 21h fungsi 4Ch dengan register AL berisi kode return. Interupsi 21h fungsi 4Ch ini lebih fleksibel untuk digunakan, karena selain kontrol akan dikembalikan kepada DOS, file-file yang terbuka juga akan ditutup oleh fungsi ini. Fungsi ini juga akan mengembalikan vektor interupsi default 22h, 23h dan 24h. Anda juga bisa mengakhiri program COM dengan fungsi ini.

26.3. MEMBUAT PROGRAM EXE

Sesuai dengan model program EXE pada gambar 26.1. akan kita buat sebuah program yang sederhana. Program berikut akan memperlihatkan kepada anda, bagaimana caranya merubah huruf kecil menjadi huruf besar.

```
;/=================================================\;
;           Program : UPCASE.ASM                    ;
;           Author  : S'to                          ;
;           Fungsi  : Merubah huruf kecil menjadi   ;
;                     huruf besar                   ;
;\=================================================/;

       .MODEL   SMALL
       .STACK   200h
       .DATA

Klm  DB 'Cinta menyebabkan rindu yang paling sengsara $'

       .CODE
Proses:
        MOV     AX,@DATA            ; Inialisasi, supaya
        MOV     DS,AX               ; DS menunjuk pada data

        XOR     BX,BX
        MOV     CX,47
  Ulang:
        CMP     Klm[BX],'a'         ; Apakah huruf kecil ?
        JB      Tidak               ; Tidak, cek berikutnya
        CMP     Klm[BX],'z'         ; Apakah huruf kecil ?
        JA      Tidak               ; Tidak, cek berikutnya
        SUB     Klm[BX],'a'-'A'     ; Jadikan huruf besar
  Tidak:
        INC     BX                  ; Akses karakter berikutnya
        LOOP    Ulang               ; Ulangi
  Cetak:
        MOV     AH,09               ; Cetak kalimat yang telah
        LEA     DX,Klm              ; dirubah menjadi huruf besar
        INT     21h                 ; semuanya
   Exit:
        MOV     AX,4C00h            ; Selesai,
        INT     21h                 ; kembali ke DOS
END     Proses
```

Program 26.1. Merubah huruf kecil menjadi huruf besar

Setelah program 26.1. anda ketikkan, compilelah menjadi bentuk EXE, dengan cara:

```
C:\>TASM UPCASE
Turbo Assembler  Version 2.0  Copyright (c) 1988, 1990
Borland International

Assembling file:   UPCASE.ASM
Error messages:    None
Warning messages:  None
```

```
Passes:           1
Remaining memory: 326k
```

Setelah source program di TASM, anda bisa me-LINK objek file dengan:

```
C:\>TLINK UPCASE
Turbo Link  Version 3.0 Copyright (c) 1987, 1990
Borland International
```

Setelah kedua proses ini selesai, maka anda akan mendapatkan sebuah program EXE yang siap dijalankan.

Bila program 26.1. dieksekusi, maka pada layar akan ditampilkan kalimat yang semua karakternya telah dirubah menjadi huruf besar, seperti:

CINTA MENYEBABKAN RINDU YANG PALING SENGSARA

BAB XXVII
TURBO DEBUGGER

27.1. PROGRAM ANDA SALAH ?

Dalam membuat program, kita biasanya akan selalu mengalami suatu kesalahan. Kesalahan dalam pembuatan sebuah program dapat dibagi menjadi dua, yaitu kesalahan syntax dan kesalahan logika. Kesalahan syntak biasanya berupa kesalahan tata cara atau aturan penulisan seperti perintah: "MOV AH,AX" (kedua operand tidak sama besar bitnya). Kesalahan syntax adalah mudah untuk dicari dan diperbaiki. Bentuk kesalahan yang kedua, yaitu kesalahan logika adalah kesalahan program yang berjalan tidak sesuai dengan yang diinginkan. Kesalahan seperti ini tidaklah ditampilkan oleh compiler seperti pada kesalahan syntax. Ingatlah, program akan selalu berjalan sesuai dengan yang diperintahkan bukannya seperti yang diinginkan.

Untuk mencari kesalahan logika, Turbo Debugger merupakan salah satu debugger yang paling canggih pada saat ini. Dengan Turbo Debugger anda bisa melihat jalannya program perintruksi, melihat isi register, melihat isi stack, melihat dan mengubah isi flags register, mengubah isi memory, mentrace program residen, mentrace device driver dan sebagainya. Dengan mengusai Turbo debugger banyak yang dapat anda lakukan, seperti melihat program orang lain, mengubah suatu program yang sudah jadi, membongkar suatu kode proteksi, serta membongkar virus.

Karena Turbo Debugger cukup besar untuk kita bicarakan secara lengkap dan mendetail, maka pada bagian ini hanya akan dibahas beberapa hal penting yang anda perlukan saja.

27.2. MENYIAPKAN PROGRAM UNTUK DILACAK

Jika anda ingin melacak kesalahan ataupun jalannya program anda, pada saat source program di TASM dan di LINK gunakan parameter sebagai berikut:

TASM/ZI UPCASE

TLINK/V UPCASE

Dengan cara ini, pada akhir file EXE yang dihasilkan akan ditambahkan informasi yang dapat digunakan oleh Turbo Debugger. Informasi ini antara lain, nama variabel dan lokasi baris dari source file yang akan dieksekusi. Jika parameter ZI dan V tidak disertakan, Turbo Debugger hanya akan menampilkan intruksi yang dijalankan tanpa embel-embel lain sehingga banyak kelebihan dari Turbo Debugger yang tidak bisa kita manfaatkan.

Untuk menggunakan Turbo Debugger, anda dapat langsung menyertakan nama file yang akan dilacak malalui parameter, seperti:

TD UPCASE

Jika Ektensi file tidak anda sertakan, maka secara default Turbo Debugger akan mencari file tersebut dengan ektensi EXE terlebih dahulu.

Bagi anda pemakai Turbo Pascal, tentunya tidak akan terlalu asing dengan tampilan Turbo Debugger ini. Tampilan pada Turbo Debugger hampir sama dengan tampilan pada Turbo Pascal, selain itu banyak pula tombol fungsi yang sama persamaanya(Gambar 27.1.)

<<<<< Gbr271.PIX >>>>>>

Gambar 27.1. Tampilan Menu Turbo Debugger

Pada lingkungan Turbo Debugger ini, mouse dapat anda gunakan dengan mudahnya tetapi bila anda tidak mempunyainya, keyboard juga dapat digunakan.

27.3. MENGGUNAKAN TURBO DEBUGGER

Pada bagian ini akan kita coba menggunakan Turbo Debugger untuk melihat jalannya program UPCASE.EXE(Program 26.1). Ingatlah untuk menyertakan parameter ZI dan V pada saat source program UPCASE.ASM anda jadikan EXE.

Setelah anda masuk pada lingkungan Turbo Debugger dengan perintah:

C:\TD UPACASE.EXE

Kini, cobalah pilih menu "View" kemudian pilih "Variabel", maka semua isi variabel program akan ditampilkan. Mungkin penempatan window variabel ini agak mengganggu anda, untuk itu anda dapat memindahkannya dengan menekan tombol Ctrl+F5 dan digeser dengan tombol panah. Setelah pergeseran selesai dilakukan tekanlah enter, maka window variabel tersebut akan segera berpindah.

Setelah itu dari menu utama, pilihlah menu "View" kemudian pilih "Registers", maka seluruh nilai register dan flag akan dapat anda lihat. Bila anda ingin juga melihat nilai dari register 32 bit, aktifkanlah menu local dari registers dengan menekan tombol ALT+F10 kemudian pilihlah "Registers 32-Bit". Bila pada saat itu, register 32 Bit belum ditampilkan maka akan segera ditampilkan, sebaliknya jika register 32 Bit sudah ditampilkan maka register 32 bit tersebut akan segera dihilangkan.

<<<<< Gbr272.PIX >>>>>

Gambar 27.2. Tampilan Turbo Debugger

Cobalah anda Trace jalannya program anda dengan menekan tombol F7 atau dengan memilih menu Run kemudian Trace. Setiap penekanan tombol F7 akan melaksanakan satu intruksi yang ditunjukkan oleh tanda panah(CS:IP) dan tanda panah akan segera bergeser pada intruksi selanjutnya yang siap dieksekusi. Tekanlah terus tombol F7 dan perhatikan perubahan nilai register dan perubahan huruf-huruf pada variabel "Klm". Setelah program selesai dieksekusi anda bisa melihat tampilan dari program dengan menekan tombol ALT+F5. Untuk keluar dari lingkungan Turbo Debugger, tekanlah tombol ALT+X.

Bila anda masih ingin mencoba, tekanlah tombol Ctrl+F2 maka file akan segera direset atau diLoad kembali dan siap diTrace lagi dari permulaan.

27.3.1. MELIHAT DAN MENGUBAH ISI REGISTER

Anda bisa melihat dan mengubah semua nilai register pada komputer, termasuk flags register dengan mudahnya. Caranya adalah:

1. Aktifkan Turbo Debugger.
2. Pilih menu "View".
3. Pilih menu "Registers", maka semua nilai register bisa anda lihat. Dengan tombol panah, anda bisa menggerakkan sorotan pada register dan dengan tombol Tabulasi anda bisa berpindah dari sorotan register ke sorotan Flags. Anda bisa mengubah nilai flags dengan menyorot flag yang ingin diganti dan tekan tombol Enter.

<<<<<< Gbr273.PIX >>>>>>>

Gambar 27.3. Menu Registers

Kini aktifkanlah Menu Local dari menu Registers dengan menekan tombol ALT+F10 yang terdiri atas:

- Increment: Untuk menambah nilai register yang sedang disorot dengan satu.
- Decrement: Untuk mengurangai nilai register yang sedang disorot dengan satu.
- Zero : Untuk menolkan nilai register yang sedang disorot.
- Change : Pilihan ini akan menampilkan sebuah kotak yang akan meminta anda untuk memasukkan angka baru untuk

register yang sedang disorot. Selain mengganti nilai register dengan cara ini, anda bisa mengganti nilai register yang sedang disorot dengan secara langsung mengetikan nilainya tanpa mengaktifkan SubMenu.

- Register 32-Bit: Pilihan ini akan menampilkan register 32 bit bila belum ditampilkan, sebaliknya register 32 bit akan dihilangkan bila sebelumnya telah ditampilkan.

27.3.2. MANIPULASI DATA MEMORY

Dengan Turbo Debugger, anda bisa melihat, mengganti, merubah, mencatat maupun menulisi data dimemory dengan mudahnya. Adapun caranya adalah:

1. Aktifkan Turbo Debugger.
2. Pilih menu "View".
3. Pilih menu "Dump", maka secara default akan ditampilkan data pada segment data(DS). Besarkanlah windownya dengan menekan tombol F5(Penekanan F5 sekali lagi akan mengecilkan windownya). Anda bisa menggerakkan posisi kursor dengan tombol panah. Data pada posisi kursor bisa diubah dengan langsung mengetikkan angka barunya atau mengetikkan karakter dengan diapit oleh tanda petik. Anda bisa menekan tombol Ctrl+G untuk menampilkan alamat baru, misalkan alamat stack, dengan memasukkan kode SS:SP pada input Box. Anda juga bisa menggunakan angka secara langsung pada input Box, seperti 0000:0000 untuk menampilkan Interrupt Vektor Table.

<<<<< Gbr274.PIX >>>>>>

Gambar 27.4. Manipulasi Data Memory

4. Kini aktifkanlah Menu Local dari menu Dump dengan menekan tombol ALT+F10. Anda akan akan melihat menu:

- Goto : Untuk menampilkan data pada suatu alamat. Perintah ini sama dengan penekanan tombol Ctrl+G yang telah kita lakukan sebelumnya.
- Search : Untuk mencari byte atau string pertama yang cocok dengan deskripsi yang kita berikan.
- Next : Untuk mencari byte atau string selanjutnya yang cocok, setelah anda mencari dengan Search.

- Change : Untuk mengganti data 1 byte atau lebih data pada posisi kursor.
- Follow : Untuk mengganti alamat code atau data tampilan dengan data pada posisi kursor.
- Previous: Untuk mengembalikan tampilan pada alamat sebelum terjadi perubahan. Perintah ini biasanya digunakan untuk membatalkan hasil dari perintah Follow dan Goto. Pekerjaan ini dapat dilakukan karena Turbo Debugger menyimpan alamat terakhir pada saat terjadi perubahan alamat.
- Display As: Untuk merubah format tampilan yang secara default adalah Byte. Perintah ini mempunyai pilihan: Byte, Word, LongInt(4 byte), Composite(8 byte), Float(Format data C), Real(6 byte), Double(Format data C) dan Extended(10 byte).
- Block : Untuk manipulasi blok memory. Perintah ini mempunyai pilihan:
    * Clear : Untuk menolkan data pada suatu block memory yang ditentukan.
    * Move : Untuk memindahkan suatu block memory pada alamat baru.
    * Set : Untuk merubah semua nilai pada suatu block memory.
    * Read : Untuk membaca file ke suatu block memory yang ditentukan. Anda harus menentukan nama file, alamat awal penyimpanan dan banyaknya byte yang akan dibaca.
    * Write : Untuk membaca suatu Data blok yang ditentukan dan ditulis ke file yang ditentukan. Pada menu ini, anda juga harus menentukan alamat awal dari data yang akan dicatat, banyaknya byte serta nama file.

27.3.3. MENTRACE PROGRAM

Pada bagian 27.3. anda telah menggunakan tombol F7 untuk mentrace program. Pada bagian ini akan kita lihat fungsi-fungsi lain yang disediakan oleh Turbo Debugger untuk mentrace program. Macam-macam fungsi trace ini umumnya terdapat pada menu Run(Gambar 27.5.).

<<<<< Gbr275.PIX >>>>>

Gambar 27.5. Menu Run

Pada menu Run ini, dapat anda lihat terdapat lagi submenu:

1. Run : Untuk menjalankan program dengan kecepatan penuh. Program akan dijalankan sampai selesai atau pada posisi Break Point. Anda bisa menggunakan fungsi ini dengan menekan tombol F9 secara langsung.

2. Go To Cursol : Untuk menjalankan program sampai posisi kursor. Untuk menggunakan fungsi ini, gerakkanlah posisi kursor dengan tombol panah menuju posisi yang diiginkan, kemudian pilihlah menu Run dan Go To Cursol atau dengan menekan tombol F4. Maka program akan dijalankan sampai posisi kursor tersebut.

3. Trace Into : Untuk menjalankan satu intruksi. Bila terdapat pemanggilan terhadap suatu procedure, fungsi ini juga akan melompat dan menjalankan intruksi pada procedure itu. Fungsi ini dapat digunakan secara langsung dengan menekan tombol F7.

4. Step Over : Sama dengan fungsi F7 perbedaannya bila terdapat pemanggilan terhadap suatu procedure, fungsi ini akan langsung menyelesaikan seluruh intruksi pada procedure tersebut. Fungsi ini dapat digunakan secara langsung dengan menekan tombol F8.

5. Execute To : Untuk menjalankan program sampai pada posisi yang ditentukan. Anda bisa menentukan alamat segment:offset ataupun dengan nama label program sebagai tempat berhenti. Fungsi ini dapat digunakan secara langsung dengan menekan tombol Alt+F9.

6. Until Return : Untuk menjalankan program sampai akhir dari suatu procedure atau procedure kembali pada pemanggilnya. Fungsi ini biasanya digunakan pada saat anda mentrace program dengan tombol F7 dan masuk pada suatu procedure, ternyata anda ingin segera melewatinya. Fungsi ini dapat digunakan secara langsung dengan menekan tombol Alt+F8.

7. Animate : Untuk menjalankan program secara otomatis sampai program selesai atau ditekan sembarang tombol. Fungsi ini sama dengan penekanan tombol F7 secara otomatis. Fungsi ini biasanya digunakan oleh orang yang mempunyai mata elang atau pada program yang terdapat intruksi yang akan mengunci keyboard sementara.

8. Back Trace : Untuk menjalankan intruksi sebelumnya yang telah dijalankan. Fungsi ini dapat digunakan secara langsung dengan menekan tombol Alt+F4.

9. Instruktion Trace : Sama dengan tombol F7, perbedaannya bila terdapat suatu interupsi maka fungsi ini akan masuk dan menjalankan program interupt handler perintruksi. Dengan fungsi ini anda dapat melihat interrupt handler suatu interupsi. Fungsi ini dapat digunakan secara langsung dengan menekan tombol Alt+F7.

10. Argumen : Untuk merubah atau mengganti command line.

11. Program Reset : Untuk mereset program. Fungsi ini biasanya digunakan bila anda ingin mentrace kembali program dari permulaaan. Fungsi ini dapat digunakan secara langsung dengan menekan tombol Ctrl+F2.

27.3.4. MELIHAT VARIABEL PROGRAM

Semua isi ataupun nilai dari variabel program bisa anda lihat dengan tombol Ctrl+F7 ataupun dengan memilih menu Data kemudian pilih menu Add Watch. Setelah itu isilah dengan nama variabel yang ingin anda lihat isinya(Gambar 27.6.).

<<<<<< Gbr276.PIX >>>>>>

Gambar 27.6. Melihat Variabel Program

Bila anda memasukkan nama dari suatu label program, maka alamat dari label tersebut akan ditampilkan. Bila anda merasa wimdow watch ini terlalu kecil, tombol F6 dapat anda gunakan untuk berpindah ke window watch ini dan membesarkannya(F5). Penekanan tombol F5 akan membesarkan atau mengecilkan window.

27.3.5. MEMBERIKAN TITIK BREAKPOINTS

Breakpoints adalah titik-titik tempat dimana program akan dihentikan. Untuk itu anda bisa melakukannya dengan:

1. Aktifkan Turbo Debugger

2. Tempatkan kursor pada posisi yang ingin anda beri tanda breakpoints.

3. Pilih menu BreakPoints kemudian Toggle, atau langsung menekan tombol F2. Maka pada posisi tersebut akan ditandai dengan sebuah sorotan garis merah(Gambar 27.7). Bila anda ingin membatalkan titik brekpoints, tekanlah tombol F2 sekali lagi pada tempat yang sama.

<<<<<< Gbr277.PIX >>>>>>

Gambar 27.7. Memberikan posisi BreakPoints

Selain pemberian posisi BreakPoints dengan F2, anda bisa juga memberikan posisi BreakPoints melalui alamatnya. Untuk itu tekanlah tombol Alt+F2 dan berikan alamat untuk titik BreakPoints tersebut.

4. Tekanlah tombol F9 untuk mengeksekusi program anda. Setiap mencapai titik BreakPoints program akan segera berhenti, untuk melanjutkannya kembali tekanlah tombol F9 lagi.

27.4. JIKA ANDA MEMPUNYAI PROGRAM YANG BESAR

Jika anda ingin melacak suatu program yang besar, kebutuhan akan memory komputer menjadi penting sekali. Pada kasus seperti ini, Turbo Debugger menyediakan 2 cara yang dapat memecahkan masalah ini, yaitu:

1. Untuk anda yang memiliki komputer 80286, disediakan program TD286.EXE yang dapat berjalan pada modus proteksi. TD286.EXE akan menyimpan dirinya pada High Area, sehingga memory konvensional hanya digunakan sedikit sekali. Program TD286 dapat juga berjalan pada semua prosesor diatasnya, seperti 80386 dan 80486.

2. Untuk anda yang memiliki komputer 80386 dengan memory extended minimal 640 KB, disediakan TD386.EXE dengan Driver TDH386.SYS. Driver TDH386.SYS akan membuat Turbo Debugger dijalankan pada modus Virtual sehingga membebaskan memory konvensional. Anda bisa menggunakan driver ini melalui CONFIG.SYS dengan menyertakan: DEVICEHIGH=TDH386.SYS. Ingatlah, driver TDH386.SYS ini biasanya akan konflik dengan memory manager yang menggunakan virtual memory, seperti QEMM.SYS. Baik TD286.EXE maupun TD386.EXE dapat dijalankan persis dengan program TD.EXE.

Selain dengan cara diatas, masih terdapat cara lain yaitu dengan menggunakan TDREMOTE.EXE. Program ini memungkinkan pelacakan sebuah program dengan menggunakan 2 komputer yang dihubungkan melalui serial port. Dengan cara ini, sebuah program dapat menggunakan memory konvensional secara optimal sementara TDREMOTE menggunakan memory dari komputer yang lain.

LAMPIRAN I

DAFTAR INSTRUKSI ASSEMBLY

Mnemonic : **AAA (ASCII Adjust For Addition)**
Tersedia pada : 8088 keatas
Syntax : AAA
Pengaruh flag : AF, CF
Fungsi : Mengatur format bilangan biner/hexa ke bentuk BCD setelah dilakukan operasi penjumlahan dua bilangan BCD. AAA hanya dapat dilakukan untuk bilangan sebesar 4 bit, maksimal hexa F dan diletakkan di register AL. Bila AL bernilai lebih dari 9, maka AL akan dikurangi 10 dan 4 bit tinggi dari AL akan dijadikan 0. Setelah itu AH akan ditambah dengan 1. CF dan AF akan diset 1.
Contoh:

```
Bilangan BCD 8 + 6 = ...
MOV AL,8h
MOV AH,6h
ADD AL,AH   ; AX = 060Eh
AAA         ; AX = 0704h
```

Jadi bilangan 0E dijadikan BCD menjadi 14, dimana bilangan di AX dibaca BCD 14 --> AH = 1(7-6), AL = 4

Mnemonic : **AAD (ASCII Adjust For Division)**
Tersedia pada : 8088 keatas
Syntax : AAD
Pengaruh flag : SF, ZF, PF
Fungsi : Mengkonversi bilangan BCD ke biner atau hexa. Adapun cara yang dilakukan adalah mengalikan AH dengan 10 dan menambahkan isi AL dengan hasilkali AH. Hasil pertambahan tersebut akan diletakkan di register AL kemudian AH akan dinolkan.
Contoh:

```
Hexa dari BCD 53 adalah ...
MOV AH,05
MOV AL,03
AAD        ; AL=0035h yaitu hexa dari BCD 53
```

Mnemonic : **AAM (ASCII Adjust For Multiplication)**
Tersedia pada : 8088 keatas
Syntax : AAM
Pengaruh flag : OF, SF, ZF, AF, PF, CF
Fungsi : Mengkonversi bilangan biner atau hexa ke BCD. Adapun cara yang dilakukan adalah membagi AL dengan 10, kemudian hasilnya dimasukkan ke register AH sedang sisanya ke register AL.
Contoh:

```
Bilangan BCD ; 12 * 9 = ...
MOV AL,12h
MOV BL,09h
MUL BL       ; AX = 00A2h
AAM          ; AX = 1002h
```

Bilangan 1002h pada AX dibaca sebagai desimal 162 :

- AH = 10h = 16
- AL = 02h = 2

Mnemonic : **AAS (ASCII Adjust For Subtraction)**
Tersedia pada : 8088 keatas
Syntax : AAS
Pengaruh flag : AF, CF
Fungsi : Mengatur format bilangan biner/hexa hasil pengurangan ke bentuk BCD. AAS ini berlaku untuk hasil pengurangan yang tidak lebih dari 4 bit. Jika 4 Bit rendah dari AL lebih besar dari 9, maka AL akan dikurangi dengan 6 dan register AH akan dikurangi 1. 4 bit atas register AL akan dijadikan nol sedangkan 4 bit rendahnya akan bernilai 0-9.
Contoh:

```
Bilangan BCD 11 - 5 = ...
MOV AL,11h
```

```
MOV BL,5h
SUB AL,BL  ; AX = 000C
AAS        ; AX = FF06
```

Mnemonic : **ADC (Add With Carry)**
Tersedia pada : 8088 keatas
Syntax : ADC Tujuan,Sumber
Pengaruh flag : OF, SF, ZF, AF, PF
Fungsi : Menambahkan "Sumber", "Tujuan" dan Carry Flag (1=on, 0=off), hasilnya diletakkan pada "Tujuan". Intruksi ini biasanya digunakan setelah operasi pada pertambahan atau perkalian yang menyebabkan Carry. Misalkan pertambahan yang melibatkan bilangan yang besar, seperti pada contoh dibawah ini:
Contoh:

```
12345678h + 9ABCDEF0 = ......
```

Kedua operand di atas berukuran 4 byte. Jelas sudah melebihi kapasitas register. Di sinilah digunakan mnemonic ADC.
Contoh:

```
MOV AX,1234h     ; AX = 1234
MOV BX,9ABCh     ; BX = 9ABC
MOV CX,5678h     ; BX = 5678
MOV DX,0DEF0h    ; DX = DEF0
ADD CX,DX        ; CX = 3568  CF = 1
ADC AX,BX        ; AX = AX+BX+CF = ACF1
```

Hasil penjumlahan tertampung di AX:CX yaitu ACF13568h.

Mnemonic : **ADD**
Tersedia pada : 8088 keatas
Syntax : ADD Tujuan,Sumber
Pengaruh flag : OF, SF, ZF, AF, PF, CF
Fungsi : Menambahkan "Sumber" dan "Tujuan" kemudian hasilnya disimpan pada "Tujuan". Bila hasil penjumlahan tidak tertampung seluruhnya pada

"Tujuan", maka CF akan diset 1.
Contoh:

```
ADD AX,BX      ; Jumlahkan 2 register
ADD AL,[350]   ; Jumlahkan register dengan isi memori
ADD [350],AL   ; Jumlahkan isi memory dengan register
ADD AH,10h     ; Jumlahkan register dengan immediate
ADD [350],10h  ; Jumlahkan isi memori dengan immediate
```

Mnemonic : **AND**
Tersedia pada : 8088 keatas
Syntax : AND Tujuan,Sumber
Pengaruh flag : OF, SF, ZF, PF, CF
Fungsi : Melakukan logika AND antara "Tujuan" dan "Sumber". Hasil dari operasi AND diletakkan pada "Tujuan". Instruksi AND umumnya digunakan untuk melihat kondisi suatu bit dengan menolkan bit-bit lainnya.
Contoh:

```
AND AL,00001000b  ; AL=0000?000
JZ  Nol           ; Jika bit ketiga AL=0, maka lompat
```

Mnemonic : **BOUND (Check Bounds Of Array Index)**
Tersedia pada : 8088 keatas
Syntax : BOUND Tujuan,Sumber
Pengaruh flag : Tidak ada
Fungsi : Untuk memastikan bahwa index array bertanda negatif atau positif masih masuk dalam batas limit yang didefinisikan oleh Double Word blok memory.

Mnemonic : **CALL**
Tersedia pada : 8088 keatas
Syntax : CALL nama-procedure
Pengaruh flag : Tidak ada
Fungsi : Melompat dan mengerjakan intruksi pada procedure program. Pada saat instruksi Call diberikan, maka processor akan melakukan :

- PUSH CS ke stack bila procedure yang dipanggil bertipe Far.
- PUSH IP ke stack.
- Mengganti nilai CS dengan segmen dari procedure bila procedure tersebut bertipe Far.
- Mengganti nilai IP dengan offset dari procedure. Lakukan intruksi yang terdapat pada alamat baru(CS:IP) sampai bertemu dengan intruksi RET, setelah itu:
- POP IP
- POP CS bila procedure bertipe Far.
- Kembali ke program induk/pemanggil.

Contoh:

```
    1CFE:0125  CALL N_PROC ; Push IP(=0128) ke stack, IP=1066
    1CFE:0128      .......
    1CFE:0155  CALL F_PROC ; Push CS(=1CFE)&IP(=0160) ke stack
                           ; CS=1FFF,IP=0179
    1CFE:0160      .......

    1CFE:1066  N_PROC  PROC NEAR
                   .......
                   .......
               RET         ; Pop IP(=0128)
               N_PROC  ENDP
    1FFF:0179  F_PROC  PROC FAR
                   .......
                   .......
               RET         ; Pop IP(=0160) & CS(=1CFE)
               F_PROC  ENDP
```

Mnemonic : **CBW (Convert Byte To Word)**
Tersedia pada : 8088 keatas
Syntax : CBW
Fungsi : Mengubah isi register AL menjadi AX dengan mengubah isi register AH menjadi 0 bila AL benilai positif atau AH akan bernilai FF bila AL negatif.

Contoh:

```
MOV AL,FFh
MOV BX,123Fh
CBW          ; AX = FFFF
ADD AX,BX    ; AX = 123F + (-1) = 123E
```

Pada bilangan bertanda, angka FFh pada AL adalah -1 bagi Assembler bukannya 255 desimal.

Mnemonic : **CLC (Clear Carry Flag)**
Tersedia pada : 8088 keatas
Syntax : CLC
Pengaruh flag : CF
Fungsi : Membuat carry flag menjadi 0.
Contoh:

Untuk menjaga agar dalam operasi RCR, rotasi pertamanya yang masuk adalah 0 maka digunakan CLC dahulu.

```
CLC
RCR AX,1
```

Mnemonic : **CLD (Clear Direction Flag)**
Tersedia pada : 8088 keatas
Syntax : CLD
Pengaruh flag : DF
Fungsi : Membuat direction flag berisi 0. Bila direction flag berisi 0 maka pembacaan string akan berlangsung dari memory rendah ke tinggi. Sebaliknya bila direction flag bernilai 1 maka string akan diproses dari memory tinggi ke rendah.
Contoh:

```
CLD          ; Arah Operasi string ke kanan
MOV  CX,0Fh  ; Bandingkan 16 byte dari string
REPE CMPSB   ; sampai ada satu yang tidak sama
```

Mnemonic : **CLI (Clear Interrupt Flag)**
Tersedia pada : 8088 keatas
Syntax : CLI
Pengaruh flag : IF
Fungsi : Membuat interrupt flag menjadi 0. Bila IF berisi 0 maka semua interupsi akan diabaikan oleh komputer, kecuali Nonmaskable Interrupt(NMI). Umumnya CLI diberikan pada saat akan dilakukan proses yang fatal, dimana terjadinya interupsi akan menghancurkan proses tersebut.
Contoh:

Kita akan mengubah alamat sebuah stack, dengan mengubah SS dan SP. Selama SS dan SP diubah, interupsi tidak boleh terjadi. Hal ini dikarenakan pada saat terjadi interupsi, register CS, IP dan Flags disimpan pada stack sebagai alamat kembali nantinya.

```
MOV  AX,AlmStack
MOV  DX,AlmOffset
CLI
MOV  SP,DX
MOV  SS,AX
STI
```

Mnemonic : **CMC (Complement Carry Flag)**
Tersedia pada : 8088 keatas
Syntax : CMC
Pengaruh flag : CF
Fungsi : Mengubah Carry flag menjadi kebalikan dari isi semulanya, seperti dari 0 menjadi 1 dan sebaliknya.

Contoh:

Pada kebanyakan operasi, Carry flag dijadikan sebagai tanda berhasil atau tidaknya operasi tersebut. Biasanya Carry flag akan bernilai 0 bila operasi berhasil dan bernilai 1 bila operasi mengalami kegagalan. Dengan menggunakan perintah CMC disertai dengan ADC(pertambahan dengan carry flag), anda dapat memanfaatkannya untuk menghitung banyaknya keberhasilan operasi

yang dilakukan, seperti:

```
                MOV   CX,Counter
                XOR   AX,AX
        Ulang:
                PUSH  AX
                Operasi
                POP   AX
                CMC
                ADC   AX,0
                LOOP  Ulang
```

Pada hasil akhir dari proses ini register AX akan berisi banyaknya operasi yang berhasil dilakukan.

Mnemonic : **CMP (Compare)**
Tersedia pada : 8088 keatas
Syntax : CMP operand1,operand2
Pengaruh flag : OF, SF, ZF, AF, PF, CF
Fungsi : Membandingkan "operand1" dengan "operand2". Adapun cara yang dilakukan adalah dengan mengurangkan "operand1" dengan "operand2" (operand1-operand2). "Operand1" dan "operand2" yang dibandingkan harus mempunyai tipe data yang sama, seperti byte dengan byte (AL,AH,BL,BH,..) atau word dengan word (AX,BX,CX,..). Perintah CMP hanya mempengaruhi flags register tanpa merubah isi "operand1" dan "operand2".
Contoh:

```
        Ulang:
              CMP    CX,AX
              JE     Exit
              LOOP   Ulang
```

Mnemonic : **CMPSB (Compare Strings Byte)**
Tersedia pada : 8088 keatas
Syntax : CMPSB
Pengaruh flag : OF, SF, ZF, AF, PF, CF
Fungsi : Untuk membandingkan satu byte pada alamat DS:SI dengan ES:DI.

Jika direction flag bernilai 1 maka setiap selesai perbandingan register SI dan DI akan ditambah dengan 1, sebaliknya jika direction flag bernilai 0 maka setiap selesai perbandingan register SI dan DI akan dikurang dengan 1.

Mnemonic : **CMPSW (Compare Strings Word)**
Tersedia pada : 8088 keatas
Syntax : CMPSW
Pengaruh flag : OF, SF, ZF, AF, PF, CF
Fungsi : Untuk membandingkan satu word pada alamat DS:SI dengan ES:DI. Jika direction flag bernilai 1 maka setiap selesai perbandingan register SI dan DI akan ditambah dengan 2, sebaliknya jika direction flag bernilai 0 maka setiap selesai perbandingan register SI dan DI akan dikurang dengan 2.

Mnemonic : **CWD (Convert Word To Doubleword)**
Tersedia pada : 8088 keatas
Syntax : CWD
Pengaruh flag : Tidak ada
Fungsi : Mengubah tipe word(AX) menjadi double word(DX). Bila AX positif maka DX akan berisi 0000, bila AX negatif maka DX berisi FFFF.

Contoh:

Anda dapat memanfaatkan fungsi CWD ini untuk mendapatkan bilangan absolute.

```
Absolut   MACRO  Bil
          MOV
          TEST   AX,10000000b  ; Apakah AX negatif?
          JZ     Selesai       ; Ya, selesai
          CWD                   ;
          XOR    AX,DX          ; Jadikan positif
          SUB    AX,DX          ;
Selesai:
          ENDM
```

Mnemonic : **DAA (Decimal Adjust After Addition)**

Tersedian pada : 8088 keatas

Syntax : DAA

Pengaruh flag : OF, SF, ZF, AF, PF, CF

Fungsi : Mengubah hasil penjumlahan 2 bilangan bukan BCD pada register AL menjadi bentuk BCD. Jika 4 bit rendah dari AL lebih besar dari 9 maka AL akan dikurangi dengan 10 dan AF diset menjadi 1, sebaliknya jika 4 bit rendah AL lebih kecil atau sama dengan 9 maka AF akan dijadikan 0. DAA sebenarnya adalah sama dengan AAA kecuali dalam hal bahwa DAA dapat mengatur baik bilangan 8 bit maupun 4 bit pada AL, sementara AAA hanya 4 bit.

Contoh:

```
Bilangan BCD : 27h + 45h = ...

MOV AH,45h
MOV AL,27h
ADD AL,AH    ; AL = 6C
DAA          ; AL = 72
```

Mnemonic : **DAS (Decimal Adjust After Substraction)**

Tersedia pada : 8088 keatas

Syntax : DAS

Pengaruh flag : OF, SF, ZF, AF, PF, CF

Fungsi : Mengubah hasil pengurangan 2 bilangan pada AL menjadi bentuk BCD. Jika 4 bit rendah dari AL lebih besar dari 9 maka AL akan dikurangi dengan 6 dan AF diset menjadi 1, sebaliknya jika 4 bit rendah dari AL lebih kecil atau sama dengan 9 maka AF akan dijadikan 0.

Contoh:

```
Bilangan BCD:  50h - 23h = ...

MOV AX,50h
SUB AX,23h  ; AX = 002D
DAS         ; AX = 0027
```

Mnemonic : **DEC (Decrement)**
Tersedia pada : 8088 keatas
Syntax : DEC Tujuan
Pengaruh flag : OF, SF, ZF, AF, PF
Fungsi : Untuk mengurangi "Tujuan" dengan 1. "Tujuan" dapat berupa register 8 bit, 16 bit, 32 bit maupun memory. Bila anda ingin mengurangi suatu register ataupun memory dengan 1, gunakanlah perintah DEC ini karena selain lebih cepat, perintah DEC juga menggunakan memory lebih sedikit dibandingkan dengan perintah SUB.
Contoh:

Kita dapat mengimplementasikan perintah Loop dengan menggunakan DEC. Di bawah ini kita akan menjumlahkan bilangan BX sampai 1. Misalnya bila BX = 5 maka dijumlahkan 5+4+3+2+1 = ...

```
        XOR   AX,AX
Loop1 :
        ADD   AX,BX
        DEC   BX
        CMP   BX,0
        JNZ   Loop1
```

Mnemonic : **DIV (Divide)**
Tersedia pada : 8088 keatas
Syntax : DIV Sumber
Pengaruh flag : OF, SF, ZF, AF, PF, CF
Fungsi : Bila "sumber" bertipe 8 bit maka dilakukan pembagian AX dengan "Sumber" (AX / Sumber). Hasil pembagian akan disimpan pada register AL sedangkan sisa pembagian akan disimpan pada regiser AH.
Jika "sumber" bertipe 16 bit maka dilakukan pembagian DX:AX dengan "Sumber" (DX:AX / Sumber). Hasil pembagian akan disimpan pada register AX sedangkan sisa pembagian akan disimpan pada regiser DX.
Contoh:

Untuk memeriksa apakah suatu bilangan merupakan kelipatan 3 atau bukan,

anda bisa membaginya dengan tiga. Bila merupakan kelipatan 3, maka sisa pembagian akan 0, sebaliknya jika bukan kelipatan tiga, sisa pembagian tidak akan 0. Macro ini akan menjadikan AL=1 bila bilangan yang ditest merupakan kelipatan tiga dan sebaliknya akan bernilai 0.

```
Lipat3   MACRO   Bil
         MOV     AX,Bil
         MOV     BX,3
         DIV     BX
         CMP     AX,0   ; Apakah ada sisa pembagian ?
         JE      Tiga   ; Tidak ada sisa , kelipatan 3
         MOV     AL,0
   Tiga :
         MOV     AL,1
         ENDM
```

Mnemonic : **ENTER (Make Stack Frame)**
Tersedia pada : 8088 keatas
Syntax : Enter Operand1,operand2
pengaruh flag : Tidak ada
Fungsi : Untuk memesan tempat pada stack yang dibutuhkan oleh bahasa tingkat tinggi.

Mnemonic :**ESC (Escape)**
Tersedia pada : 8088 keatas
Syntax : ESC Operand1,Operand2
Pengaruh flag : Tidak ada
Fungsi : Perintah ini digunakan terutama pada komputer yang mempunyai procesor lebih dari satu dan dirangkai secara paralel. Perintah ESC akan menyebabkan procesor yang sedang aktif dinonaktifkan sehingga procesor yang lain dapat digunakan.

Mnemonic : **HLT (Halt)**
Tersedia pada : 8088 keatas
Syntax : HLT
Pengaruh flag : Tidak ada

Fungsi : Untuk menghentikan program atau membuat procesor dalam keadaan tidak aktif. Setelah mendapat perintah ESC ini, procesor harus mendapat interrupt dari luar atau direset untuk berjalan secara normal kembali.

Mnemonic : **IDIV (Signed Divide)**
Tersedia pada : 8088 keatas
Syntax : IDIV Sumber
Pengaruh flag : OF, SF, ZF, AF, PF, CF
Fungsi : IDIV digunakan untuk pembagian pada bilangan bertanda. Bila "sumber" bertipe 8 bit maka dilakukan pembagian AX dengan "Sumber" (AX / Sumber). Hasil pembagian akan disimpan pada register AL sedangkan sisa pembagian akan disimpan pada regiser AH.
Jika sumber bertipe 16 bit maka dilakukan pembagian DX:AX dengan Sumber(DX:AX / Sumber). Hasil pembagian akan disimpan pada register AX sedangkan sisa pembagian akan disimpan pada regiser DX.
Contoh:

```
MOV   BL,10h
MOV   AX,-10h
IDIV  BL          ; AX = 00FFh(-1)
```

Mnemonic : **IMUL (Signed Multiply)**
Tersedia pada : 8088 keatas
Syntax : IMUL Sumber
Khusus 80386:
IMUL Tujuan,Sumber
IMUL Tujuan,Pengali,Sumber
Pengaruh flag : OF, SF, ZF, AF, PF, CF
Fungsi : IMUL digunakan untuk perkalian pada bilangan bertanda. Bila "sumber" bertipe 8 bit maka akan dilakukan perkalian pada register AL dengan "sumber" kemudian hasilnya disimpan pada AX. Bila "sumber" bertipe 16 bit maka akan dilakukan perkalian pada register AX dengan "sumber" kemudian hasilnya disimpan pada DX:AX.
Contoh:

```
MOV   AX,100h    ; AX=100h
MOV   BX,-2      ; BX=FFFEh
IMUL  BX         ; DX=FFFFh,  AX=FE00h
```

Mnemonic : **IN (Input From Port)**
Tersedia pada : 8088 keatas
Syntax : IN Operand,NoPort
Pengaruh flag : Tidak ada
Fungsi : Untuk mengambil data pada port. Jika "Operand" merupakan register AL maka akan diambil data pada port sebanyak 1 byte, bila "operand" merupakan register AX maka akan diambil data pada port sebanyak 1 word. "NoPort" mencatat nomor port yang akan dibaca datanya. "NoPort" bisa langsung diberi nilai bila nomor port dibawah 255. Bila nomor port melebihi 255 maka "NoPort" harus berupa register DX yang mencatat nomor port tersebut.
Contoh:

Interrupt dari keyboard diatur oleh PIC(Programmable Interrupt Controller) yang berada pada port 21h. Jika bit ke 1 dari port 21h bernilai 1, maka interupsi dari keyboard akan diabaikan.

```
NoKey  MACRO
       IN    AL,21h
       OR    AL,00000010b
       OUT   21h,AL
       ENDM
```

Mnemonic : **INC (Increment)**
Tersedia pada : 8088 keatas
Syntax : INC Tujuan
Pengaruh flag : OF, SF, ZF, AF, PF
Fungsi : Untuk menambah "Tujuan" dengan 1. Bila anda ingin menambah suatu register ataupun memory dengan 1, gunakanlah perintah INC ini karena selain lebih cepat, perintah INC juga menggunakan memory lebih sedikit dibandingkan dengan perintah ADD.
Contoh:

Untuk membuat suatu pengulangan, seperti pada perintah 'FOR I:=1 TO 10 DO' pada bahasa tingkat tinggi:

```
       XOR  AX,AX
 Ulang:
       CMP  AX,10
       JE   Selesai
       INC  AX
       JMP  Ulang
```

Mnemonic : **INS (Input From Port To String)**
Tersedia pada : 8088 keatas
Syntax : INS Operand,NoPort
Pengaruh flag : Tidak ada
Fungsi : Untuk mengambil data dari "NoPort" yang dicatat oleh register DX sebanyak 1 byte atau 1 word, sesuai dengan tipe "operand". Jadi "operand" hanya berfungsi sebagai penunjuk besarnya data yang akan dibaca dari port. Data yang diambil dari port akan disimpan pada lokasi ES:DI.

Jika Direction flag bernilai 0(dengan CLD) maka setelah intruksi INS dijalankan register DI akan ditambah secara otomatis, sebaliknya jika Direction flag bernilai 1(dengan STD) maka register DI akan dikurang secara otomatis. Anda bisa menggunakan intruksi pengulangan pada string disertai dengan perintah INS ini.
Contoh:

```
       MOV   DX,123h
       MOV   CX,30
 Ulang:
       INS   AX,DX
       LOOP  Ulang
```

Mnemonic : **INSB (Input String Byte From Port)**
Tersedia pada : 8088 keatas
Syntax : INSB

Pengaruh flag : Tidak ada

Fungsi : Untuk mengambil data dari nomor port yang dicatat oleh register DX sebanyak 1 byte. Data yang diambil dari port akan disimpan pada lokasi ES:DI.

Jika Direction flag bernilai 0(dengan CLD) maka setelah intruksi INS dijalankan register DI akan ditambah dengan 1 secara otomatis, sebaliknya jika Direction flag bernilai 1(dengan STD) maka register DI akan dikurang dengan 1 secara otomatis. Anda bisa menggunakan intruksi pengulangan pada string disertai dengan perintah INSB ini.

Contoh:

```
REP   INSB
```

Mnemonic : **INSW**

Tersedia pada : 8088 keatas

Syntax : INSW

Pengaruh Flag : Tidak ada

Fungsi : Untuk mengambil data dari nomor port yang dicatat oleh register DX sebanyak 1 Word. Data yang diambil dari port akan disimpan pada lokasi ES:DI.

Jika Direction flag bernilai 0(dengan CLD) maka setelah intruksi INS dijalankan register DI akan ditambah dengan 2 secara otomatis, sebaliknya jika Direction flag bernilai 1(dengan STD) maka register DI akan dikurang dengan 2 secara otomatis. Anda bisa menggunakan intruksi pengulangan pada string disertai dengan perintah INSW ini.

Contoh:

```
REP   INSW
```

Mnemonic : **INT (Interrupt)**

Tersedia pada : 8088 keatas

Syntax : INT NoInt

Pengaruh flag : IF, TF

Fungsi : Untuk membangkitkan software interrupt yang bernomor 0 sampai 255. Setiap terjadi suatu interupsi data flags, CS dan IP akan disimpan pada

stack. Data ini selanjutmya digunakan sebagai alamat kembali setelah komputer melakukan suatu rutin atau interrupt handler.

Mnemonic : **INTO (Interrupt If Overflow)**
Tersedia pada : 8088 keatas
Syntax : INTO
Pengaruh Flag : Tidak ada
Fungsi : Jika Overflow flag bernilai 1, maka INTO akan melaksanakan interrupt 04h, sebaliknya jika Overflow flag bernilai 0 maka interrupt 04h tidak akan dilaksanakan. INTO hampir sama dengan INT hanya INTO khusus untuk membangkitkan interrupt 04h jika OF=1.

Mnemonic : **IRET (Interrupt Return)**
Tersedia pada : 8088 keatas
Syntax : IRET
Pengaruh Flag : OF, DF, IF, TF, SF, ZF, AF, PF, CF
Fungsi : Digunakan untuk mengakhiri suatu interrupt handler. IRET akan mengambil IP, CS dan Flags yang disimpan pada stack pada saat terjadi suatu interupsi(INT).

Mnemonic : **JA (Jump If Above)**
Tersedia pada : 8088 keatas
Syntax : JA Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Melakukan suatu loncatan menuju "tujuan" bila CF=0 dan ZF=0. "Tujuan" dapat berupa nama label ataupun alamat memory. Pada lompatan bersyarat ini, besarnya lompatan tidak bisa melebihi -128 dan +127 byte kecuali pada 80386 yang mampu mencapai -32768 dan +32767. JA identik dengan perintah JNBE yang biasanya digunakan setelah dilakukan suatu perbandingan dengan CMP.
Catatan : JA dan JNBE melakukan operasi pada bilangan tidak bertanda.
Contoh:

```
CMP   AX,BX
JA    Besar
```

Pada perintah diatas, loncatan menuju label "besar" akan dilakukan bila pada perintah CMP diatasnya register AX lebih besar dari register BX. Perintah JA beroperasi pada bilangan tidak bertanda atau bilangan yang tidak mengenal tanda minus.

Mnemonic : **JAE (Jump If Above or Equal)**
Tersedia pada : 8088 keatas
Syntax : JAE Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Melakukan suatu loncatan menuju "tujuan" bila CF=0. "Tujuan" dapat berupa nama label ataupun alamat memory. Pada lompatan bersyarat ini, besarnya lompatan tidak bisa melebihi -128 dan +127 byte. JAE identik dengan perintah JNB yang biasanya digunakan setelah dilakukan suatu perbandingan dengan CMP.
Catatan : JAE dan JNB melakukan operasi pada bilangan tidak bertanda.
Contoh:

```
CMP   AX,BX
JAE   BesarSama
```

Pada perintah diatas, loncatan menuju label "BesarSama" akan dilakukan bila pada perintah CMP diatasnya register AX lebih besar atau sama dengan register BX. Perintah JAE beroperasi pada bilangan tidak bertanda atau bilangan yang tidak mengenal tanda minus.

Mnemonic : **JB (Jump If Bellow)**
Tersedia pada : 8088 keatas
Syntax : JB Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Melakukan suatu loncatan menuju "tujuan" bila CF=1. "Tujuan" dapat berupa nama label ataupun alamat memory. Pada lompatan bersyarat ini, besarnya lompatan tidak bisa melebihi -128 dan +127 byte. JB identik dengan

perintah JNAE dan JC .
Catatan : JB dan JNAE melakukan operasi pada bilangan tidak bertanda.

Contoh:

```
CMP   AX,BX
JB    Kecil
```

Pada perintah diatas, loncatan menuju label "Kecil" akan dilakukan bila pada perintah CMP diatasnya register AX lebih kecil dibandingkan dengan register BX.

Mnemonic : **JBE (Jump If Below or Equal)**
Tersedia pada : 8088 keatas
Syntax : JBE Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Melakukan suatu loncatan menuju "tujuan" bila CF=1 atau ZF=1. "Tujuan" dapat berupa nama label ataupun alamat memory. Pada lompatan bersyarat ini, besarnya lompatan tidak bisa melebihi -128 dan +127 byte. JBE identik dengan perintah JNA.
Catatan : JBE dan JNA melakukan operasi pada bilangan tidak bertanda.
Contoh:

```
CMP   AX,BX
JBE   KecilSama
```

Pada perintah diatas, loncatan menuju label "KecilSama" akan dilakukan bila pada perintah CMP diatasnya register AX lebih kecil atau sama dengan register BX.

Mnemonic : **JC (Jump On Carry)**
Tersedia pada : 8088 keatas
Syntax : JC Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Sama dengan JB dan JNAE.

Mnemonic : **JCXZ (Jump If CX = 0)**
Tersedia pada : 8088 keatas
Syntax : JCXZ Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Melakukan suatu loncatan menuju "tujuan" bila register CX=0. "Tujuan" dapat berupa nama label ataupun alamat memory. Pada lompatan bersyarat ini, besarnya lompatan tidak bisa melebihi -128 dan +127 byte.
Contoh:

```
MOV   AH,00          ;
LEA   DX,Buffer      ; Masukan string dari keyboard
INT   21h            ;

MOV   CX,Buffer[1]   ; CX = banyaknya masukan string
JCXZ  Tdk            ; Jika tidak ada, lompat ke "Tdk"
```

Mnemonic : **JE (Jump Equal)**
Tersedia pada : 8088 keatas
Syntax : JE Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Melakukan suatu loncatan menuju "Tujuan" bila ZF=1. "Tujuan" dapat berupa nama label ataupun alamat memory. Pada lompatan bersyarat ini, besarnya lompatan tidak bisa melebihi -128 dan +127 byte. JE identik dengan JZ.
Contoh:

```
CMP   AX,BX
JE    Sama
```

Pada perintah diatas, loncatan menuju label "Sama" akan dilakukan bila pada perintah CMP diatasnya register AX sama dengan register BX yang menyebabkan zero flag bernilai 1.

Mnemonic : **JG (Jump If Greater)**
Tersedia pada : 8088 keatas

Syntax : JG Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Melakukan suatu loncatan menuju "tujuan" bila ZF=0 dan SF=OF. "Tujuan" dapat berupa nama label ataupun alamat memory. Pada lompatan bersyarat ini, besarnya lompatan tidak bisa melebihi -128 dan +127 byte. JG identik dengan perintah JNLE.
Catatan : JG dan JNLE melakukan operasi pada bilangan bertanda.
Contoh:

```
CMP   AX,BX
JG    Besar
```

Pada perintah diatas, loncatan menuju label "Besar" akan dilakukan bila pada perintah CMP diatasnya register AX lebih besar dibandingkan dengan register BX.

Mnemonic : **JGE (Jump If Greater or Equal)**
Tersedia pada : 8088 keatas
Syntax : JGE Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Melakukan suatu loncatan menuju "tujuan" bila SF=OF. "Tujuan" dapat berupa nama label ataupun alamat memory. Pada lompatan bersyarat ini, besarnya lompatan tidak bisa melebihi -128 dan +127 byte. JGE identik dengan perintah JNL.
Catatan : JGE dan JNL melakukan operasi pada bilangan bertanda.
Contoh:

```
CMP   AX,BX
JGE   BesarSama
```

Pada perintah diatas, loncatan menuju label "BesarSama" akan dilakukan bila pada perintah CMP diatasnya register AX lebih besar atau sama dengan register BX.

Mnemonic : **JL (Jump If Less Than)**
Tersedia pada : 8088 keatas
Syntax : JL Tujuan

Pengaruh flag : Tidak Ada
Fungsi : Melakukan suatu loncatan menuju "tujuan" bila SF tidak sama dengan OF. "Tujuan" dapat berupa nama label ataupun alamat memory. Pada lompatan bersyarat ini, besarnya lompatan tidak bisa melebihi -128 dan +127 byte. JL identik dengan perintah JNGE.
Catatan : JL dan JNGE melakukan operasi pada bilangan bertanda.
Contoh:

```
CMP   AX,BX
JL    Kecil
```

Pada perintah diatas, loncatan menuju label "Kecil" akan dilakukan bila pada perintah CMP diatasnya register AX lebih kecil dibandingkan dengan register BX.

Mnemonic : **JLE (Jump If Less Than or Equal)**
Tersedia pada : 8088 keatas
Syntax : JLE Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Melakukan suatu loncatan menuju "tujuan" bila ZF=1 atau SF tidak sama dengan OF. "Tujuan" dapat berupa nama label ataupun alamat memory. Pada lompatan bersyarat ini, besarnya lompatan tidak bisa melebihi -128 dan +127 byte. JLE identik dengan perintah JNG.
Catatan : JLE dan JNG melakukan operasi pada bilangan bertanda.
Contoh:

```
CMP   AX,BX
JLE   KecilSama
```

Pada perintah diatas, loncatan menuju label "KecilSama" akan dilakukan bila pada perintah CMP diatasnya register AX lebih kecil atau sama dengan register BX.

Mnemonic : **JMP (Jump)**
Tersedia pada : 8088 keatas

Syntax : JMP Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Melakukan lompatan menuju "Tujuan" yang dapat berupa suatu label maupun alamat memory. Tidak seperti lompatan bersyarat, perintah JMP dapat melakukan lompatan ke segment lain.
Contoh:

```
        JMP   Proses
```

Mnemonic : **JNA (Jump If Not Above)**
Tersedia pada : 8088 keatas
Syntax : JNA Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Identik dengan JBE.

Mnemonic : **JNAE (Jump If Not Above or Equal)**
Tersedia pada : 8088 keatas
Syntax : JNAE Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Identik dengan JB.

Mnemonic : **JNB (Jump If Not Bellow)**
Tersedia pada : 8088 keatas
Syntax : JNB Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Identik dengan JAE.

Mnemonic : **JNBE (Jump If Not Bellow or Equal)**
Tersedia pada : 8088 keatas
Syntax : JNBE Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Identik dengan JA.

Mnemonic : **JNC (Jump Not Carry)**
Tersedia pada : 8088 keatas
Syntax : JNC Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Identik dengan JNB.

Mnemonic : **JNE (Jump Not Equal)**
Tersedia pada : 8088 keatas
Syntax : JNE Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Melakukan suatu loncatan menuju "tujuan" bila ZF=0. "Tujuan" dapat berupa nama label ataupun alamat memory. Pada lompatan bersyarat ini, besarnya lompatan tidak bisa melebihi -128 dan +127 byte. JNE identik dengan perintah JNZ.
Contoh:

```
CMP   AX,BX
JNE   TidakSama
```

Pada perintah diatas, loncatan menuju label "TidakSama" akan dilakukan bila pada perintah CMP diatasnya register AX tidak sama dengan register BX.

Mnemonic : **JNG (Jump If Not Greater)**
Tersedia pada : 8088 keatas
Syntax : JNG Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Identik dengan JLE.

Mnemonic : **JNGE (Jump If Not Greater or Equal)**
Tersedia pada : 8088 keatas
Syntax : JNGE Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Identik dengan JL.

Mnemonic : **JNL (Jump If Not Less Than)**
Tersedia pada : 8088 keatas
Syntax : JNL Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Identik dengan JGE.

Mnemonic : **JNLE (Jump If Not Less or Equal)**
Tersedia pada : 8088 keatas
Syntax : JNLE Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Identik dengan JG.

Mnemonic : **JNO (Jump On Not Overflow)**
Tersedia pada : 8088 keatas
Syntax : JNO Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Melakukan suatu loncatan menuju "tujuan" bila OF=0. "Tujuan" dapat berupa nama label ataupun alamat memory. Pada lompatan bersyarat ini, besarnya lompatan tidak bisa melebihi -128 dan +127 byte.
Catatan : JNO melakukan operasi pada bilangan bertanda.

Mnemonic : **JNP (Jump On No Parity)**
Tersedia pada : 8088 keatas
Syntax : JNP Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Melakukan suatu loncatan menuju "tujuan" bila PF=0. "Tujuan" dapat berupa nama label ataupun alamat memory. Pada lompatan bersyarat ini, besarnya lompatan tidak bisa melebihi -128 dan +127 byte. JNP identik dengan perintah JPO.
Catatan : JNP dan JPO melakukan operasi pada bilangan tidak bertanda.

Mnemonic : **JNS (Jump On No Sign)**
Tersedia pada : 8088 keatas
Syntax : JNS Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Melakukan suatu loncatan menuju "tujuan" bila SF=0. "Tujuan" dapat berupa nama label ataupun alamat memory. Pada lompatan bersyarat ini, besarnya lompatan tidak bisa melebihi -128 dan +127 byte.
Catatan : JNS melakukan operasi pada bilangan bertanda.

Mnemonic : **JNZ (Jump On Not Zero)**
Tersedia pada : 8088 keatas
Syntax : JNZ Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Identik dengan JNE.

Mnemonic : **JO (Jump On Overflow)**
Tersedia pada : 8088 keatas
Syntax : JO Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Melakukan suatu loncatan menuju "tujuan" bila OF=1. "Tujuan" dapat berupa nama label ataupun alamat memory. Pada lompatan bersyarat ini, besarnya lompatan tidak bisa melebihi -128 dan +127 byte.
Catatan : JO melakukan operasi pada bilangan bertanda.

Mnemonic : **JP (Jump On Parity)**
Tersedia pada : 8088 keatas
Syntax : JP Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Melakukan suatu loncatan menuju "tujuan" bila PF=1. "Tujuan" dapat berupa nama label ataupun alamat memory. Pada lompatan bersyarat ini, besarnya lompatan tidak bisa melebihi -128 dan +127 byte. JP identik dengan perintah JPE.

Mnemonic : **JPE (Jump On Parity Even)**
Tersedia pada : 8088 keatas
Syntax : JPE Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Identik dengan JP.

Mnemonic : **JPO (Jump On Parity Odd)**
Tersedia pada : 8088 keatas
Syntax : JPO Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Identik dengan JNP.

Mnemonic : **JS (Jump On Sign)**
Tersedia pada : 8088 keatas
Syntax : JS Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Melakukan suatu loncatan menuju "tujuan" bila SF=1. "Tujuan" dapat berupa nama label ataupun alamat memory. Pada lompatan bersyarat ini, besarnya lompatan tidak bisa melebihi -128 dan +127 byte.
Catatan : JS melakukan operasi pada bilangan bertanda.

Mnemonic : **JZ (Jump On Zero)**
Tersedia pada : 8088 keatas
Syntax : JZ Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Identik dengan JE.

Mnemonic : **LAHF (Load Flags Into AH Register)**
Tersedia pada : 8088 keatas
Pengaruh flag : Tidak Ada
Syntax : LAHF

Fungsi : Untuk mengcopykan CF, PF, AF, ZF dan SF yang terletak pada bit ke 0, 2, 4, 6 dan 7 dari flags register menuju register AH pada bit yang sesuai(0, 2, 4, 6 dan 7).
Contoh:

Dengan fungsi ini, anda bisa menyimpan 8 bit rendah dari flags register untuk menghindari perubahan akibat dari suatu proses, seperti:

```
LAHF          ; Simpan flags pada AH
PUSH AX       ; Nilai AH disimpan ke stack
  +---------+
  | Proses |
  +---------+
POP AX        ; Keluarkan AH dari stack
SAHF          ; Masukkan ke flag register
```

Mnemonic : **LDS (Load Pointer Using DS Register)**
Tersedia pada : 8088 keatas
Syntax : LDS Operand,Mem32
Pengaruh flag : Tidak Ada
Fungsi : Untuk menyimpan Double Word dari memory. 16 bit rendah akan disimpan pada "Operand" sedangkan 16 bit tingginya akan disimpan pada register DS. "Mem32" mencatat lokasi dari suatu memory yang didefinisikan dengan tipe Double Word(DD).
Contoh:

```
TData:  JMP     Proses
        Tabel   DD ?,?,?
Proses:
        LDS     BX,Tabel
```

Setelah intruksi diatas dijalankan, maka pasangan register DS:BX akan mencatat alamat dari variabel "Tabel".

Mnemonic : **LEA (Load Effective Address)**

Tersedia pada : 8088 keatas
Syntax : LEA Operand,Mem16
Pengaruh flag : Tidak Ada
Fungsi : Untuk mendapatkan alamat efektif atau offset dari "Mem16" dimana "Operand" merupakan suatu register 16 bit. Perintah LEA hampir sama dengan perintah OFFSET yang juga digunakan untuk mendapatkan offset dari suatu memory. Perintah LEA lebih fleksibel untuk digunakan dibandingkan dengan perintah OFFSET karena dengan LEA kita bisa memberikan tambahan nilai pada "Mem16".
Contoh:

```
TData :JMP    Proses
       Kal    '0123456789abcd'
Proses:
       MOV  SI,10
       LEA  BX,Kal[SI]
```

Dengan perintah diatas, register BX akan mencatat offset ke 10 dari Kal.

Mnemonic : **LES (Load Pointer Using ES Register)**
Tersedia pada : 8088 keatas
Syntax : LES Operand,Mem32
Pengaruh flag : Tidak Ada
Fungsi : Untuk menyimpan Double Word dari memory. 16 bit rendah akan disimpan pada "Operand" sedangkan 16 bit tingginya akan disimpan pada register ES. "Mem32" mencatat lokasi dari suatu memory yang didefinisikan dengan tipe Double Word(DD).
Contoh:

```
TData:  JMP     Proses
        Tabel   DD ?,?,?
Proses:
        LES     BX,Tabel
```

Setelah intruksi diatas dijalankan, maka pasangan register ES:BX akan mencatat alamat dari variabel "Tabel".

Mnemonic : **LOCK (Lock The BUS)**
Tersedia pada : 8088 keatas
Syntax : LOCK Operand
Pengaruh flag : Tidak Ada
Fungsi : Perintah ini digunakan terutama pada komputer yang mempunyai procesor lebih dari satu. Perintah LOCK mengunci suatu area terhadap pemakaian oleh mikroprocesor lainnya.
Catatan: Perintah XCHG selalu mengaktifkan LOCK.

Mnemonic : **LODSB (Load A Byte From String Into AL)**
Tersedia pada : 8088 keatas
Syntax : LODSB
Pengaruh flag : Tidak Ada
Fungsi : Untuk mengcopy data 1 byte dari alamat DS:SI ke register AL.

Mnemonic : **LOOP**
Tersedia pada : 8088 keatas
Syntax : LOOP Tujuan
Pengaruh flag : Tidak Ada
Fungsi : Untuk melakukan suatu proses yang berulang (Looping) dengan CX sebagai counternya. Perintah LOOP akan mengurangkan CX dengan 1 kemudian dilihat apakah CX telah mencapai nol. Proses looping akan selesai jika register CX telah mencapai nol. Oleh karena inilah maka jika kita menjadikan CX=0 pada saat pertama kali, kita akan mendapatkan suatu pengulangan yang terbanyak karena pengurangan 0-1=-1(FFFF).
Catatan: LOOP hanya dapat melakukan lompatan menuju "Tujuan" bila jaraknya tidak lebih dari -128 dan +127 byte.
Contoh:

```
        XOR   CX,CX ; Melakukan pengulangan
Ulang:              ; sebanyak FFFF kali
        LOOP  Ulang ;
```

Mnemonic : **LOOPE (Loop While Equal)**

Tersedia pada : 8088 keatas

Syntax : LOOPE Tujuan

Pengaruh flag : Tidak Ada

Fungsi : Untuk melakukan pengulangan selama register CX tidak sama dengan 0 dan ZF=1. LOOPE hanya dapat melakukan lompatan menuju "Tujuan" bila jaraknya tidak lebih dari -128 dan +127 byte. LOOPE identik dengan LOOPZ.

Mnemonic : **LOOPNE (Loop While Not Equal)**

Tersedia pada : 8088 keatas

Syntax : LOOPNE Tujuan

Pengaruh flag : Tidak Ada

Fungsi : Untuk melakukan pengulangan selama register CX tidak sama dengan 0 dan ZF=0. LOOPNE hanya dapat melakukan lompatan menuju "Tujuan" bila jaraknya tidak lebih dari -128 dan +127 byte. LOOPE identik dengan LOOPNZ.

Mnemonic : **LOOPNZ (Loop While Not Zero)**

Tersedia pada : 8088 keatas

Syntax : LOOPNZ Tujuan

Pengaruh flag : Tidak Ada

Fungsi : Identik dengan LOOPNE.

Mnemonic : **LOOPZ**

Tersedia pada : 8088 keatas

Syntax : LOOPZ Tujuan

Pengaruh flag : Tidak Ada

Fungsi : Identik dengan LOOPE.

Mnemonic : **MOV (Move)**

Tersedia pada : 8088 keatas

Syntax : MOV Tujuan,Sumber

Pengaruh flag : Tidak Ada

Fungsi : Untuk mengcopykan isi "Sumber" ke "Tujuan". Antara "Sumber" dan "Tujuan" harus mempunyai tipe data yang sama, seperti AL dan BL, AX dan BX. Pada perintah MOV ini harus anda perhatikan bahwa:

- Segment register tidak bisa langsung diisi nilainya, seperti:

```
MOV   ES,0B800h
```

Untuk masalah seperti ini anda bisa menggunakan register general purpose sebagai perantara, seperti:

```
MOV   AX,0B800h
MOV   ES,AX
```

- Register CS sebaiknya tidak digunakan sebagai "Tujuan" karena hal ini akan mengacaukan program anda.

- Pengcopyan data antar segment register tidak bisa digunakan, seperti:

```
MOV   ES,DS
```

Untuk masalah seperti ini anda bisa menggunakan register general purpose atau stack sebagai perantara, seperti:

```
MOV   AX,DS
MOV   ES,AX
```

atau:

```
PUSH  DS
POP   ES
```

- Pengcopyan data antar lokasi memory, seperti:

```
MOV  A,B
```

Untuk masalah seperti ini anda bisa menggunakan suatu register sebagai perantara, seperti:

```
MOV  AX,B
MOV  A,AX
```

Mnemonic : **MOVSW (Move String Byte By Byte)**
Tersedia pada : 8088 keatas
Syntax : MOVSB
Pengaruh flag : Tidak Ada
Fungsi : Untuk mengcopy data 1 byte dari alamat DS:SI ke alamat ES:DI. Bila DF=0(CLD) maka setelah intruksi MOVSB dijalankan, register SI dan DI akan ditambah dengan 1 sebaliknya jika DF=1(STD) maka register SI dan DI akan dikurang dengan 1.

Mnemonic : **MOVSW (Move String Word By Word)**
Tersedia pada : 8088 keatas
Syntax : MOVSW
Pengaruh flag : Tidak Ada
Fungsi : Untuk mengcopy data 1 Word dari alamat DS:SI ke alamat ES:DI. Bila DF=0(CLD) maka setelah intruksi MOVSW dijalankan, register SI dan DI akan ditambah dengan 2 sebaliknya jika DF=1(STD) maka register SI dan DI akan dikurang dengan 2.

Mnemonic : **MUL (Multiply)**
Tersedia pada : 8088 keatas
Syntax : MUL Sumber
Pengaruh flag : OF, CF
Fungsi : Bila "Sumber" bertipe 8 bit maka akan dilakukan perkalian antara "Sumber" dengan AL. Hasilnya disimpan pada register AX. Bila "Sumber" bertipe 16 bit maka akan dilakukan perkalian antara "Sumber" dengan AX. Hasilnya disimpan pada pasangan register DX:AX.
Contoh:

```
MUL  BH    ; AX    = BH * AL
MUL  BX    ; DX:AX = BX * AX
```

Mnemonic : **NEG (Negate)**

Tersedia pada : 8088 keatas

Syntax : NEG Operand

Pengaruh flag : OF, SF, ZF, AF, PF, CF

Fungsi : Untuk mengubah "Operand" menjadi negatif.

Contoh:

```
MOV  BH,1  ; BH = 01h
NEG  BH    ; BH = FFh
```

Mnemonic : **NOT**

Tersedia pada : 8088 keatas

Syntax : NOT Operand

Pengaruh flag : Tidak Ada

Fungsi : Membalikkan bit pada operand. Jika bit operand bernilai 0 akan dijadikan 1 sebaliknya jika 1 akan dijadikan 0.

Contoh:

```
MOV  AL,10010011b
NOT  AL          ; AL = 01101100b
```

Mnemonic : **OR**

Tersedia pada : 8088 keatas

Syntax : OR Tujuan,Sumber

Pengaruh flag : OF, SF, ZF, PF, CF

Fungsi : Melakukan logika OR antara "Tujuan" dan "Sumber". Hasil dari operasi OR diletakkan pada "Tujuan". Instruksi OR umumnya digunakan untuk menjadikan suatu bit menjadi 1.

Contoh:

```
OR  AL,00000001b  ; Jadikan bit ke 0 dari AL menjadi 1
```

Mnemonic : **OUT (Output to Port)**

Tersedia pada : 8088 keatas

Syntax : OUT NoPort,Operand

Pengaruh flag : Tidak ada
Fungsi : Untuk memasukkan 1 byte atau 1 word dari "operand" ke "NoPort" sesuai dengan tipe data "operand". Jika "Operand" merupakan register AL maka akan dikirimkan data pada port sebanyak 1 byte, bila "operand" merupakan register AX maka akan dikirimkan data pada port sebanyak 1 word. "NoPort" bisa langsung diberi nilai bila nomor port dibawah 255. Bila nomor port melebihi 255 maka "NoPort" harus berupa register DX yang mencatat nomor port tersebut.

Mnemonic : **OUTS (Output String To Port)**
Tersedia pada : 8088 keatas
Syntax : OUTS NoPort,Operand
Pengaruh flag : Tidak ada
Fungsi : Untuk mengirimkan data dari "operand" ke "NoPort" yang dicatat oleh register DX sebanyak 1 byte atau 1 word, sesuai dengan tipe "operand". Jadi "operand" hanya berfungsi sebagai penunjuk besarnya data yang akan dikirimkan menuju port. Data yang akan dikirimkan menuju port disimpan pada lokasi ES:SI.

Jika Direction flag bernilai 0(dengan CLD) maka setelah intruksi OUTS dijalankan register SI akan ditambah secara otomatis, sebaliknya jika Direction flag bernilai 1(dengan STD) maka register SI akan dikurang secara otomatis. Anda bisa menggunakan intruksi pengulangan pada string disertai dengan perintah OUTS ini.

Mnemonic : **OUTSB (Output String Byte To Port)**
Tersedia pada : 8088 keatas
Syntax : OUTSB
Pengaruh flag : Tidak ada
Fungsi : Untuk memasukkan data dari lokasi DS:SI ke nomor port yang dicatat oleh register DX sebanyak 1 byte.

Jika Direction flag bernilai 0(dengan CLD) maka setelah intruksi OUTSB dijalankan register SI akan ditambah dengan 1 secara otomatis, sebaliknya jika Direction flag bernilai 1(dengan STD) maka register SI akan dikurang dengan 1 secara otomatis. Anda bisa menggunakan intruksi pengulangan pada string disertai dengan perintah OUTSB ini.

Mnemonic : **OUTSW (Output String Wyte To Port)**
Tersedia pada : 8088 keatas
Syntax : OUTSW
Pengaruh flag : Tidak ada
Fungsi : Untuk memasukkan data dari lokasi DS:SI ke nomor port yang dicatat oleh register DX sebanyak 1 word.

Jika Direction flag bernilai 0(dengan CLD) maka setelah intruksi OUTSW dijalankan register SI akan ditambah dengan 2 secara otomatis, sebaliknya jika Direction flag bernilai 1(dengan STD) maka register SI akan dikurang dengan 2 secara otomatis. Anda bisa menggunakan intruksi pengulangan pada string disertai dengan perintah OUTSW ini.

Mnemonic : **POP (Pop A Word From Stack)**
Tersedia pada : 8088 keatas
Syntax : POP Tujuan
Pengaruh flag : Tidak ada
Fungsi : Untuk mengambil data 1 word dari stack(SS:SP) dan disimpan pada "Tujuan". Setelah intruksi POP dijalankan register SP akan ditambah dengan 2.
Contoh:

```
PUSH  DS
POP   ES
```

Mnemonic : **POPA (Pop All General Purpose Registers)**
Tersedia pada : 8088 keatas
Syntax : POPA
Pengaruh flag : Tidak ada
Fungsi : Untuk mengambil 8 word dari stack menuju register berturut-turut DI, SI, BP, SP, BX, DX, CX dan AX. Dengan PUSHA program anda akan berjalan lebih cepat dan efesian dibandingkan bila anda menyimpannya dengan cara:

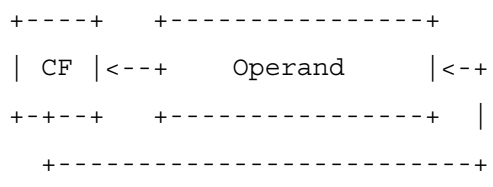
```
    POP  DI
    POP  SI
    POP  BP
    POP  SP
    POP  BX
    POP  DX
    POP  CX
    POP  AX
```

Mnemonic : **POPF (Pop Flags Off Stack)**
Tersedia pada : 8088 keatas
Syntax : POPF
Pengaruh flag : OF, DF, IF, TF, SF, ZF, AF, PF, CF
Fungsi : Untuk mengambil 1 word dari stack ke flags registers. Setelah intruksi POPF dijalankan register SP akan ditambah dengan 2.

Mnemonic : **PUSH (Push Operand Onto Stack)**
Tersedia pada : 8088 keatas
Syntax : PUSH Sumber
Pengaruh flag : Tidak ada
Fungsi : Untuk menyimpan data 1 word dari "Sumber" ke stack(SS:SP). "Sumber" merupakan operand yang bertipe 16 bit. Setelah intruksi PUSH dijalankan register SP akan dikurang dengan 2.

Mnemonic : **PUSHF (Push Flags Onto Stack)**
Tersedia pada : 8088 keatas
Syntax : PUSHF
Pengaruh flag : Tidak ada
Fungsi : Untuk menyimpan flags registers ke stack. Intruksi ini merupakan kebalikan dari intruksi POPF. Setelah intruksi PUSHF dijalankan register SP akan dikurang dengan 2.

Mnemonic : **RCL (Rotate Through Carry Left)**
Tersedia pada : 8088 keatas
Syntax : RCL Operand,Reg
Pengaruh flag : OF, CF
Fungsi : Untuk memutar "Operand" sebanyak "Reg" kali ke kiri melalui CF. Bit yang tergeser keluar dari kiri akan dimasukkan pada CF dan nilai CF akan dimasukkan pada bit terkanan dari "Operand".

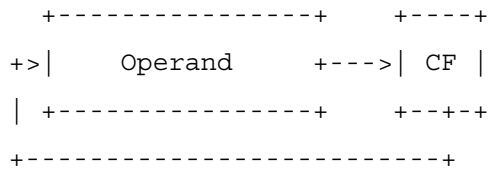


Pada mikroprosesor 8088 "Reg" haruslah berupa register CL atau CX bila perputaran yang dilakukan terhadap "Operand" lebih dari 1.

Contoh:

```
MOV  BL,00100110b    ; Misalkan CF=1
RCL  BL,1            ; BL = 01001101b  CF=0
```

Mnemonic : **RCR (Rotate through Carry Right)**
Tersedia pada : 8088 keatas
Syntax : RCR Operand,Reg
Pengaruh flag : OF, CF
Fungsi : Untuk memutar "Operand" sebanyak "Reg" kali ke kanan melalui CF. Bit yang tergeser keluar dari kanan akan dimasukkan pada CF dan nilai CF akan dimasukkan pada bit terkiri dari "Operand".



Pada mikroprosesor 8088 "Reg" haruslah berupa register CL atau CX bila perputaran yang dilakukan terhadap "Operand" lebih dari 1.

Contoh:

```
MOV  BL,00100110b    ; Misalkan CF=1
RCR  BL,1            ; BL = 10010011b  CF=0
```

Mnemonic : **REP (Repeat)**
Tersedia pada : 8088 keatas
Syntax : REP Instruksi
Pengaruh flag : Tidak ada
Fungsi : Melakukan "intruksi" sebanyak CX kali. Intruksi REP biasanya diikuti oleh Intruksi yang berhubungan dengan operasi pada string.

Mnemonic : **REPE (Repeat While Equal)**
Tersedia pada : 8088 keatas
Syntax : REPE Intruksi
Pengaruh flag : Tidak ada
Fungsi : Melakukan "intruksi" sebanyak CX kali atau sampai zero flag bernilai 1. Intruksi REPE biasanya diikuti oleh Intruksi yang berhubungan dengan operasi pada string.
Contoh:

```
MOV    CX,200
REPE   CMPS  String1,String2
```

Intruksi diatas akan membandingkan string1 dengan string2 sebanyak 200 kali atau sampai ditemukan adanya ketidaksamaan antara String1 dan String2 yang membuat zero flag bernilai 1.

Mnemonic : **REPNE (Repeat While Not Equal)**
Tersedia pada : 8088 keatas
Syntax : REPNE Instruksi
Pengaruh flag : Tidak ada
Fungsi : Melakukan "intruksi" sebanyak CX kali atau sampai zero flag bernilai 0. Intruksi REPNE biasanya diikuti oleh Intruksi yang berhubungan dengan operasi pada string.
Contoh:

```
MOV     CX,200
REPNE   CMPS  String1,String2
```

Intruksi diatas akan membandingkan string1 dengan string2 sebanyak 200 kali atau sampai ditemukan adanya kesamaan antara String1 dan String2 yang membuat zero flag bernilai 0.

Mnemonic : **REPNZ (Repeat While Not Zero)**
Tersedia pada : 8088 keatas
Syntax : REPNZ Instruksi
Pengaruh flag : Tidak ada
Fungsi : Identik dengan REPNE.

Mnemonic : **REPZ (Repeat While Zero)**
Tersedia pada : 8088 keatas
Syntax : REPZ Instruksi
Pengaruh flag : Tidak ada
Fungsi : Identik dengan REPE.

Mnemonic : **RET (Return From Procedure)**
Tersedia pada : 8088 keatas
Syntax : RET [Size]

Pengaruh flag : Tidak ada

Fungsi : RET akan mengambil alamat pada stack untuk melakukan lompatan pada alamat tersebut. RET biasanya diletakkan pada akhir dari suatu procedure yang dipanggil dengan CALL (Lihat CALL). Bila pada perintah RET diberi parameter, maka paramter itu akan digunakan sebagai angka tambahan dalam mengambil data pada stack.

Contoh:

```
RET  2
```

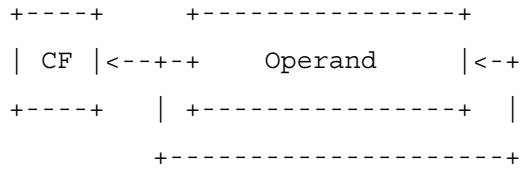
Mnemonic : **ROL (Rotate Left)**

Tersedia pada : 8088 keatas

Syntax : ROL Operand,Reg

Pengaruh flag : OF, CF

Fungsi : Untuk memutar "Operand" sebanyak "Reg" kali ke kiri. Bit yang tergeser keluar dari kiri akan dimasukkan pada CF dan pada bit terkanan dari "Operand".



Pada mikroprosesor 8088 "Reg" haruslah berupa register CL atau CX bila perputaran yang dilakukan terhadap "Operand" lebih dari 1.

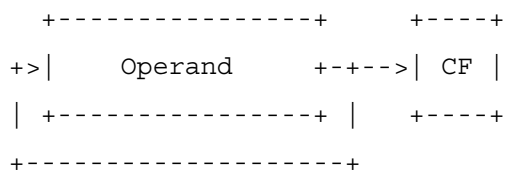
Contoh:

```
MOV  BL,00100110b    ; Misalkan CF=1
ROL  BL,1            ; BL = 01001100b  CF=0
```

Mnemonic : **ROR (Rotate Right)**

Tersedia pada : 8088 keatas

Syntax : ROR Operand,Reg
Pengaruh flag : OF, CF
Fungsi : Untuk memutar "Operand" sebanyak "Reg" kali ke kanan. Bit yang tergeser keluar dari kanan akan dimasukkan pada CF dan pada bit terkiri dari "Operand".



Pada mikroprosesor 8088 "Reg" haruslah berupa register CL atau CX bila perputaran yang dilakukan terhadap "Operand" lebih dari 1.

Contoh:

```
MOV  BL,00100110b    ; Misalkan CF=1
ROR  BL,1            ; BL = 00010011b  CF=0
```

Mnemonic : **SAHF (Store AH into Flag Register)**
Tersedia pada : 8088 keatas
Syntax : SAHF
Pengaruh flag : SF, ZF, AF, PF, CF
Fungsi : Untuk mengcopykan nilai AH pada 8 bit rendah dari flag register, yaitu: CF, PF, AF, ZF dan SF. Intruksi ini merupakan kebalikan dari Intruksi LAHF.

Mnemonic : **SAL (Arithmatic Shift Left)**
Tersedia pada : 8088 keatas
Syntax : SAL Operand,Reg
Pengaruh flag : OF, SF, ZF, PF, CF
Fungsi : Untuk menggerser "Operand" sebanyak "Reg" kali ke kiri(Lihat SHL).

Mnemonic : **SBB (Substract With Carry)**
Tersedia pada : 8088 keatas
Syntax : SBB Tujuan,Sumber
Pengaruh flag : OF, SF, ZF, AF, PF, CF
Fungsi : Untuk mengurangkan "Sumber" dengan "Tujuan" dan Carry Flag (1=on, 0=off), hasilnya diletakkan pada "Tujuan" (Tujuan=Tujuan-Sumber-CF).

Mnemonic : **SCASB (Scan String For Byte)**
Tersedia pada : 8088 keatas
Syntax : SCASB
Pengaruh flag : OF, SF, ZF, AF, PF, CF
Fungsi : Untuk membandingkan isi register AL dengan data pada alamat ES:DI.

Mnemonic : **SCASW (Scan String for Word)**
Tersedia pada : 8088 keatas
Syntax : SCASW
Pengaruh flag : OF, SF, ZF, AF, PF, CF
Fungsi : Untuk membandingkan isi register AX dengan data pada alamat ES:DI.

Mnemonic : **SHL (Shift Left)**
Tersedia pada : 8088 keatas
Pengaruh flag : OF, SF, ZF, PF, CF
Syntax : SHL Operand,Reg
Fungsi : Untuk menggeser bit dari "operand" sebanyak "Reg" kali ke kiri. Bila pergeseran yang dilakukan lebih dari 1 kali maka "Reg" harus berupa regisrer CL atau CX.
Contoh:

```
MOV  BL,00000001b
SHL  BL,1          ; BL=00000010b
```

Dengan menggeser n kali kekiri adalah sama bila bilangan tersebut dikali dengan 2 pangkat n.

Mnemonic : **SHR (Shift Right)**
Tersedia pada : 8088 keatas
Syntax : SHR Operand,Reg
Pengaruh flag : OF, SF, ZF, PF, CF
Fungsi : Untuk menggeser bit dari "operand" sebanyak "Reg" kali ke kanan. Bila pergeseran yang dilakukan lebih dari 1 kali maka "Reg" harus berupa regisrer CL atau CX.
Contoh:

```
MOV  BL,00000100b
SHR  BL,1          ; BL=00000010b
```

Dengan menggeser n kali kekanan adalah sama bila bilangan tersebut dibagi dengan 2 pangkat n.

Mnemonic : **STC (Set Carry Flag)**
Tersedia pada : 8088 keatas
Syntax : STC
Pengaruh flag : CF
Fungsi : Untuk menjadikan Carry flag menjadi 1.

Mnemonic : **STD (Set Direction Flag)**
Tersedia pada : 8088 keatas
Syntax : STD
Pengaruh flag : DF
Fungsi : Untuk menjadikan Direction flag menjadi 1. Intruksi ini merupakan kebalikan dari intruksi CLD (lihat CLD).

Mnemonic : **STI (Set Interrupt Flag)**
Tersedia pada : 8088 keatas

Syntax : STI

Pengaruh flag : IF

Fungsi : Untuk membuat Interrupt flag menjadi 1 agar interupsi dapat terjadi lagi. Intruksi ini merupakan kebalikan dari intruksi CLI (lihat CLI).

Mnemonic : **STOSB (Store AL)**

Tersedia pada : 8088 keatas

Syntax : STOSB

Pengaruh flag : Tidak ada

Fungsi : Untuk mengcopykan isi register AL pada alamat ES:DI. Jika Direction flag bernilai 0(dengan CLD) maka setelah intruksi STOSB dijalankan register DI akan ditambah dengan 1 secara otomatis, sebaliknya jika Direction flag bernilai 1(dengan STD) maka register DI akan dikurang dengan 1 secara otomatis. Anda bisa menggunakan intruksi pengulangan pada string disertai dengan perintah STOSB ini.

Mnemonic : **STOSW (Store Word in AX at String)**

Tersedia pada : 8088 keatas

Syntax : STOSW

Pengaruh flag : Tidak ada

Fungsi : Untuk mengcopykan isi register AX pada alamat ES:DI. Jika Direction flag bernilai 0(dengan CLD) maka setelah intruksi STOSW dijalankan register DI akan ditambah dengan 2 secara otomatis, sebaliknya jika Direction flag bernilai 1(dengan STD) maka register DI akan dikurang dengan 2 secara otomatis. Anda bisa menggunakan intruksi pengulangan pada string disertai dengan perintah STOSW ini.

Mnemonic : **SUB (Substract)**

Tersedia pada : 8088 keatas

Syntax : SUB Tujuan,Sumber

Pengaruh flag : OF, SF, ZF, AF, PF, CF

Fungsi : Untuk mengurangkan "Tujuan" dengan "Sumber". Hasil pengurangan akan disimpan pada "Tujuan".

Contoh:

```
SUB  BX,BX    ; BX=0
```

Mnemonic : **TEST**
Tersedia pada : 8088 keatas
Syntax : TEST Operand1,Operand2
Pengaruh flag : OF, SF, ZF, PF, CF
Fungsi : Untuk melakukan operasi AND antara "Operand1" dan "Operand2". Hasil dari operasi AND hanya mempengaruhi flag register saja dan tidak mempengaruhi nilai "operand1" maupun "operand2".

Mnemonic : **WAIT**
Tersedia pada : 8088 keatas
Syntax : WAIT
Pengaruh flag : Tidak ada
Fungsi : Untuk menghentikan CPU sampai adanya interupsi dari luar. Intruksi ini bisanya digunakan untuk bekerja sama dengan Coprosesor.

Mnemonic : **XCHG (Exchange)**
Tersedia pada : 8088 keatas
Syntax : XCHG Operand1,Operand2
Pengaruh flag : Tidak ada
Fungsi : Untuk menukar "Operand1" dengan "Operand2".
Contoh:

Potongan program di bawah akan mengatur agar AX > BX

```
        CMP  AX,BX
        JAE  Selesai
        XCHG AX,BX
Selesai :
        ..........
```

Mnemonic : **XLAT (Translate)**

Tersedia pada : 8088 keatas

Syntax : XLAT

Pengaruh flag : Tidak ada

Fungsi : Untuk mengcopykan isi dari alamat DS:[BX] ditambah dengan AL ke AL (AL=DS:[BX]+AL).

Mnemonic : **XOR**

Tersedia pada : 8088 keatas

Syntax : XOR Tujuan,Sumber

Pengaruh flag : Tidak ada

Fungsi : Melakukan logika XOR antara "Tujuan" dan "Sumber". Hasil dari operasi XOR diletakkan pada "Tujuan". Fungsi XOR yg paling populer adalah untuk menolkan suatu register. Dengan logika XOR program akan berjalan lebih cepat dan efisien.

Contoh:

```
XOR AX,AX   ; AX akan bernilai nol setelah perintah ini
```

LAMPIRAN II

DAFTAR INTERRUPT PILIHAN

INTERRUPT 05h

Print Screen

Fungsi : Mencetak seluruh isi layar ke printer.

Register Input : Tidak Ada. **Register Output :** Tidak Ada.

INTERRUPT 09h

Keyboard

Fungsi : Interupsi 09 merupakan HardWare interupsi dari keyboard. Setiap penekanan tombol keyboard akan membangkitkan interupsi 09. Handler dari interupsi 09 kemudian akan mengambil data dari tombol apa yang ditekan dari Port 60h yang berisi kode scan tombol. Dari kode scan ini kemudian akan diterjemahkan dalam kode ASCII atau Extended dan disimpan pada keyboard buffer untuk kemudian digunakan oleh interupsi lain.

INTERRUPT 10h - Service 00h

Set Video Mode

Fungsi : Mengubah Mode Video.

Register Input : **Register Output :** Tidak Ada.

AH = 00h

AL = nomor mode

Penjelasan : Setiap dilakukan pergantian mode akan menimbulkan efek CLS, kecuali nomor mode dijumlahkan dengan 128 atau bit ke-7 pada AL diset 1.

INTERRUPT 10h - Service 01h

Set Cursor Size

Fungsi : Merubah ukuran kursor pada mode teks.

Register Input : **Register Output :** Tidak Ada.

AH = 01h

CH = awal garis bentuk kursor

CL = akhir garis bentuk kursor

INTERRUPT 10h - Service 02h

Set Cursor Position

Fungsi : Meletakkan kursor pada posisi tertentu.

Register Input : **Register Output :** Tidak Ada.

AH = 02h

BH = nomor halaman tampilan

DH = nomor baris (dimulai 00)

DL = nomor kolom (dimulai 00)

INTERRUPT 10h - Service 03h

Cursor Information

Fungsi : Memperoleh Informasi posisi kursor dan ukurannya.

Register Input :	**Register Output :**
AH = 03h	CH = awal garis bentuk kursor
BH = nomor halaman tampilan	CL = akhir garis bentuk kursor
	DH = nomor baris
	DL = nomor kolom

Penjelasan : Setiap halaman tampilan memiliki kursornya sendiri- sendiri.

INTERRUPT 10h - Service 05h

Select Active Page

Fungsi : Merubah halaman tampilan aktif.

Register Input : **Register Output :** Tidak Ada.

AH = 05h

AL = nomor halaman tampilan

INTERRUPT 10h - Service 06h

Scroll Up Window

Fungsi : Menggulung jendela ke atas.

Register Input : **Register Output :** Tidak Ada.

AH = 06h

AL = jumlah baris untuk digulung

BH = atribut untuk baris kosong

CH,CL = koordinat kiri atas jendela

DH,DL = koordinat kanan bawah jendela

Penjelasan : AL diisi 00 akan mengosongkan keseluruhan jendela.

INTERRUPT 10h - Service 07h

Scroll Down Window

Fungsi : Menggulung jendela ke bawah.

Register Input : **Register Output :** Tidak Ada.

AH = 07h

AL = jumlah baris untuk digulung

BH = atribut untuk baris kosong

CH,CL = koordinat kiri atas jendela

DH,DL = koordinat kanan bawah jendela

Penjelasan : AL diisi 00 akan mengosongkan keseluruhan jendela.

INTERRUPT 10h - Service 09h

Write Character And Attribute At Cursor Position

Fungsi : Mencetak karakter dan atribut pada posisi kursor.

Register Input : **Register Output :** Tidak Ada.

AH = 09h

AL = kode ASCII karakter

BH = nomor halaman

BL = atribut karakter

CX = jumlah pengulangan pencetakan

Penjelasan : Karakter kontrol akan dicetak sebagai karakter biasa.

INTERRUPT 10h - Service 0Ah

Write Character At Cursor Position

Fungsi : Mencetak karakter pada posisi kursor.

Register Input : **Register Output :** Tidak Ada.

AH = 0Ah

AL = kode ASCII karakter

BH = nomor halaman

CX = jumlah pengulangan pencetakan

Penjelasan : Karakter kontrol akan dicetak sebagai karakter biasa, atribut yang digunakan akan sama dengan atribut yang lama.

INTERRUPT 10h - Service 0Eh

Teletype Output

Fungsi : Output karakter sederhana.

Register Input : **Register Output :** Tidak Ada.

AH = 0Eh

AL = kode ASCII karakter

BH = nomor halaman

Penjelasan : Karakter kontrol berpengaruh sesuai fungsinya. Pada ROM BIOS dengan tanggal antara 24/4/81 ke atas umumnya register BH tidak berfungsi karena setiap output akan dicetak ke halaman aktif.

INTERRUPT 10h - Service 0Fh

Get Current Video Mode

Fungsi : Mendapatkan mode video aktif.

Register Input :

AH = 0Fh

Register Output :

AL = mode video (gambar 5.1.)

AH = jumlah karakter per kolom

BH = nomor halaman tampilan

Penjelasan : Jika mode video diset dengan bit 7 on, maka AL yang didapat juga akan berisi bit 7 on.

Konflik : Driver tampilan VUIMAGE

INTERRUPT 10h - Service 11h,Subservice 00h

Load User Specific Character

Fungsi : Membuat karakter ASCII baru.

Terdapat pada : Sistem dilengkapi tampilan EGA, VGA, dan MCGA.

Register Input :

AH = 11h

AL = 00h

CX = jumlah karakter akan diubah

DX = nomor karakter mulai diubah

BL = nomor blok untuk diubah

BH = jumlah byte per karakter

ES:BP = buffer bentuk karakter

Register Output : Tidak Ada

INTERRUPT 10h - Service 11h,Subservice 03h

Set Block Specifier

Fungsi : Memberikan identitas tabel karakter untuk ditampilkan.

Terdapat pada : Sistem dilengkapi tampilan EGA, VGA, dan MCGA.

Register Input :

AH = 11h

AL = 03h

BL = penanda tabel karakter

Register Output : Tidak Ada

INTERRUPT 10h - Service 12h,Subservice 10h

Get EGA Information

Fungsi : Memperoleh karakteristik sistem video.

Terdapat pada : Sistem dilengkapi tampilan EGA ke atas.

Register Input :	**Register Output :**
AH = 12h	BH = 00h monitor warna
BL = 10h	01h monitor mono
	BL = 00h Card 64 KB
	01h Card 128 KB
	02h Card 192 KB
	03h Card 256 KB

INTERRUPT 10h - Service 13h

Write String

Fungsi : Mencetak string ke Layar

Terdapat pada : Mesin 80286 ke atas dengan tampilan EGA ke atas.

Register Input : **Register Output :** Tidak Ada.

AH = 13h

AL = 00h

BH = nomor halaman

BL = atribut untuk string

CX = jumlah karakter pada string

DH,DL = koordinat untuk memulai pencetakan

ES:BP = alamat string yang akan dicetak

Penjelasan : Memperlakukan karakter kontrol sesuai fungsinya.

INTERRUPT 10h - Service 1Ah,Subservice 00h

Get Display Combination

Fungsi : Memperoleh informasi tampilan

Terdapat pada : Sistem dengan tampilan VGA dan MCGA.

Register Input :	**Register Output :**
AH = 1Ah	AL = 1Ah jika berhasil
AL = 00h	BL = kode tampilan aktif
	BH = kode tampilan kedua

INTERRUPT 10h - Service 1Bh

VGA State Information

Fungsi : Memperoleh informasi tampilan.

Terdapat pada : Sistem dengan tampilan VGA dan MCGA.

Register Input :	**Register Output :**
AH = 1Bh	AL = 1Bh bila berhasil
BX = 0000h	ES:DI = alamat buffer 64
ES:DI = alamat buffer 64	byte informasi
byte untuk diisi	

INTERRUPT 10h - Service 4Fh,Subservice 00h

VESA SuperVga Information

Fungsi : Memperoleh informasi VESA.

Terdapat pada : Sistem dengan card tampilan VESA SuperVga.

Register Input :	**Register Output :**
AX = 4F00h	AL = 4Fh jika berhasil
ES:DI = alamat buffer 256	AH = 00h jika berhasil
byte untuk diisi	ES:DI = alamat buffer 256
	byte informasi

INTERRUPT 16h - Service 00h

Get Keystroke

Fungsi : Menunggu masukan keyboard.

Terdapat pada : Semua mesin.

Register Input :	**Register Output :**
AH = 00h	Jika AL=0 maka
	AH = kode Extended
	Jika AL<>0 maka
	AL = Kode ASCII
	AH = Kode Scan

INTERRUPT 16h - Service 01h

Check Keystroke

Fungsi : Mengecek isi keyboard buffer.

Terdapat pada : Semua mesin.

Register Input :

AH = 01h

Register Output :

ZF=0 bila buffer tidak kosong

Jika AL=0 maka

AH = kode Extended

Jika AL<>0 maka

AL = Kode ASCII

AH = Kode Scan

ZF=1 bila buffer kosong

INTERRUPT 16h - Service 10h

Get Enhanced Keystroke

Fungsi : Menunggu masukan keyboard.

Terdapat pada : Mesin AT dengan keyboard Enhanced.

Register Input :

AH = 10h

Register Output :

AH = kode scan

AL = kode ASCII

INTERRUPT 16h - Service 11h

Check Enhanced Keystroke

Fungsi : Mengecek isi keyboard buffer.

Terdapat pada : Mesin AT dengan keyboard Enhanced.

Register Input :

AH = 11h

Register Output :

ZF=0 bila buffer tidak kosong

Jika AL=0 maka

AH = kode Extended

Jika AL<>0 maka

AL = Kode ASCII

AH = Kode Scan

ZF=1 bila buffer kosong

INTERRUPT 19h

Bootstrap Loader

Fungsi : Melakukan Warm Boot.

Register Input : Tidak Ada.

Register Output : Tidak Ada.

INTERRUPT 1Bh

Control Break Handler

Fungsi : Interupsi ini terjadi setiap kali terjadi penekanan tombol Ctrl+Break.

Register Input : Tidak Ada. **Register Output :** Tidak Ada.

INTERRUPT 1Ch

Timer Tick

Fungsi : Interupsi ini disediakan untuk digunakan oleh pemakai. Interupsi 1Ch akan terjadi kurang lebih 18,2 kali setiap detik. Anda bisa membuat program residen dengan memanfaatkan timer tick ini.

Register Input : Tidak Ada. **Register Output :** Tidak Ada.

INTERRUPT 21h - Service 01h

Read Character With Echo

Fungsi : Untuk membaca masukan 1 karakter dari keyboard dan menampilkannya ke layar. Fungsi ini dapat dihentikan oleh penekanan tombol Ctrl+Break.

Register Input : **Register Output :**

AH = 10h AL = Kode ASCII

Catatan : Berbeda dengan fungsi dari BIOS, untuk membaca karakter khusus yang mempunyai kode Extended, anda harus membacanya dua kali dengan fungsi dari DOS ini.

INTERRUPT 21h - Service 02h

Write Character To Standard Output

Fungsi : Untuk mencetak satu buah karakter pada layar.

Register Input : **Register Output :** Tidak ada

AH = 02h

DL = Kode ASCII

INTERRUPT 21h - Service 07h

Direct Character Input Without Echo

Fungsi : Untuk membaca masukan 1 karakter dari keyboard. Fungsi ini tidak akan

menampilkan karakter yang ditekan pada layar, selain itu penekanan tombol Ctrl+Break juga akan diabaikan.

Register Input : **Register Output :**

AH = 07h AL = Kode ASCII

INTERRUPT 21h - Service 08h

Character Input Without Echo

Fungsi : Untuk membaca masukan 1 karakter dari keyboard. Fungsi ini tidak akan menampilkan karakter yang ditekan pada layar. Penekanan tombol Ctrl+Break akan menghentikan fungsi ini.

Register Input : **Register Output :**

AH = 08h AL = Kode ASCII

INTERRUPT 21h - Service 09h

Write String To Standard Output

Fungsi : Untuk mencetak string ke layar.

Register Input : **Register Output :** Tidak ada

AH = 09h

DS:DX = String yang diakhiri
dengan tanda "$".

INTERRUPT 21h - Service 0Ah

Input String

Fungsi : Untuk mendapatkan masukan string dari keyboard.

Register Input : **Register Output :**

AH = 0Ah Buffer terisi

DS:DX = Buffer

Spesifikasi buffer:

- Offset 00 mencatat maksimum karakter yang dapat dimasukkan.
- Offset 01 banyaknya masukan dari keyboard yang telah diketikkan. Tombol CR tidak akan dihitung.
- Offset 02 keatas, tempat dari string yang diketikkan disimpan.

INTERRUPT 21h - Service 0Bh

Get Status

Fungsi : Mengecek isi keyboard buffer.

Register Input :
AH = 0Bh

Register Output :
AL = 00 jika tidak ada karakter
AL = FFh jika ada karakter

INTERRUPT 21h - Service 0Ch

Clear Buffer dan Read Input

Fungsi : Untuk mengosongkan keyboard buffer, kemudian melaksanakan fungsi DOS 01, 06, 07, 08 atau 0Ah.

Register Input :
AH = 0Ch
AL = Fungsi yang akan dieksekusi setelah buffer dikosongkan (01,06,0,08 atau 0Ah).

Register Output : Tidak ada

INTERRUPT 21h - Service 0Fh

Open File Using FCB

Fungsi : Membuka file dengan cara FCB.

Register Input :
AH = 0Fh
DS:DX = FCB

Register Output :
AL = 00h Jika sukses
AL = FFh Jika gagal

INTERRUPT 21h - Service 10h

Closes File Using FCB

Fungsi : Untuk menutup file dengan cara FCB.

Register Input :
AH = 10h
DS:DX = FCB

Register Output :
AL = 00h Jika sukses
AL = FFh Jika gagal

INTERRUPT 21h - Service 13h

Delete File Using FCB

Fungsi: Menghapus file dengan cara FCB.

Register Input :

Register Output :

AH = 13h AL = 00h Jika sukses

DS:DX = FCB AL = FFh Jika gagal

INTERRUPT 21h - Service 14h

Sequential Read From FCB File

Fungsi : Untuk membaca file secara sekuensial.

Register Input : **Register Output :**

AH = 14h AL = status

DS:DX = FCB

INTERRUPT 21h - Service 15h

Sequential Write To FCB File

Fungsi : Untuk menulis file secara sekuential.

Register Input : **Register Output :**

AH = 14h AL = Status

DS:DX = FCB

INTERRUPT 21h - Service 16h

Create File Using FCB

Fungsi : Untuk menciptakan sebuah file baru dengan cara FCB.

Register Input : **Register Output :**

AH = 16h AL = Status

DS:DX = FCB

INTERRUPT 21h - Service 17h

Rename File Using FCB

Fungsi : Untuk mengganti nama file dengan cara FCB.

Register Input : **Register Output :**

AH = 17h AL = Status

DS:DX = FCB

INTERRUPT 21h - Service 1Ah

Set DTA

Fungsi : Untuk merubah alamat DTA. Secara default DTA terletak pada PSP offset ke 80h sebanyak 128 byte.

Register Input : **Register Output :** Tidak ada

AH = 1Ah

DS:DX = Lokasi DTA yang baru

INTERRUPT 21h - Service 21h

Read Random Record From FCB File

Fungsi : Untuk membaca record dari file FCB.

Register Input :	**Register Output :**
AH = 21h	AL = Status
DS:DX = FCB	DTA = Hasil pembacaan

INTERRUPT 21h - Service 22h

Write random Record To FCB File

Fungsi : untuk menulis record ke file FCB.

Register Input :	**Register Output :**
AH = 22h	AL = Status
DS:DX = FCB	
DTA = Data record	

INTERRUPT 21h - Service 23h

Get File Size

Fungsi : Untuk mengetahui besarnya suatu file.

Register Input :	**Register Output :**
AH = 23h	AL = Status
DS:DX = FCB	

INTERRUPT 21h - Service 24h

Set Random Record Number For FCB

Fungsi : Untuk memindahkan record untuk diakses oleh fungsi 21h dan 22h.

Register Input : **Register Output :**

AH = 24h

DS:DX = FCB

INTERRUPT 21h - Service 25h

Set Interrupt Vektor

Fungsi : Untuk merubah vektor interupsi ke suatu lokasi dengan merubah alamat awal vektor interupsi.

Register Input :	**Register Output :**
AH = 25h	
AL = Nomor Interupsi	
DS:DX = Lokasi baru	

Konflik : Phar Lap 386

INTERRUPT 21h - Service 27h

Random Block Read From FCB File

Fungsi : Untuk membaca sejumlah record dari suatu file.

Register Input :	**Register Output :**
AH = 27h	AL = Status
CX = Banyaknya record yang akan dibaca	DTA = hasil pembacaan CX = Banyaknya record yang berhasil dibaca
DS:DX = FCB	

INTERRUPT 21h - Service 28h

Random Block Write To FCB File

Fungsi : Untuk menulis sejumlah record ke suatu file

Register Input :	**Register Output :**
AH = 28h	AL = Status
CX = Banyaknya record yang akan ditulisi	CX = Banyaknya record yang berhasil ditulis
DS:DX = FCB	
DTA = Data dari record	

INTERRUPT 21h - Service 2Fh

Get DTA Address

Fungsi : Untuk mengetahui alamat dari DTA yang digunakan.

Register Input :	**Register Output :**
AH = 2Fh	ES:BX = Lokasi DTA

INTERRUPT 21h - Service 30h

Get DOS Version

Fungsi : Untuk mengetahui versi DOS yang sedang digunakan

Register Input :	**Register Output :**
AH = 30h	AL = Angka mayor
	AH = Angka minor

INTERRUPT 21h - Service 31h

Terminate And Stay Residen

Fungsi : Untuk meresidenkan suatu program.

Register Input :	**Register Output :**
AH = 31h	
AL = Kode return	
DX = Besar memory dalam paragraf	

INTERRUPT 21h - Service 33h

Extended Break Checking

Fungsi : Untuk menghidup dan matikan pengecekan tombol Ctrl+Break oleh fungsi DOS.

Register Input :	**Register Output :**
AH = 33h	Jika input AL=0
AL = 0 untuk mengambil keterangan Ctrl+Break	DL=0 Off DL=1 On
= 1 untuk merubah status Ctrl+Break DL = 0 Ctrl+Break dijadikan Off DL = 1 Ctrl+Break dijadikan On	

INTERRUPT 21h - Service 34h

GET ADDRESS OF INDOS FLAG

Fungsi : Untuk mendapatkan alamat dari BAD (Bendera Aktif DOS). Nilai BAD akan bertambah pada saat interupsi dari DOS dijalankan dan akan berkurang saat interupsi dari DOS selesai. Dengan melihat pada BAD anda dapat mengetahui apakah interupsi dari DOS sedang aktif atau tidak(Lihat bagian residen).

Register Input :	**Register Output :**
AH = 34h	ES:BX = Alamat BAD

INTERRUPT 21h - Service 35h

Get Interrupt Vektor

Fungsi : Untuk mendapatkan alamat vektor interupsi dari suatu nomor interupsi.

Register Input :	**Register Output :**
AH = 35h	ES:BX = Alamat vektor interupsi
AL = Nomor Interupsi	

INTERRUPT 21h - Service 3Ch

Create File Handle

Fungsi : Untuk menciptakan sebuah file baru dengan metode File Handle.

Register Input :	**Register Output :**
AH = 3Ch	Jika CF=0 maka
AL = Mode, dengan bit:	AX=Nomor file handle
0 file Read only	Jika CF=1 maka
1 file Hidden	AX=Kode kesalahan
2 file System	
3 Volume label	
4 Cadangan	
5 file Archive	
DS:DX = Nama file ASCIIZ	

INTERRUPT 21h - Service 3Dh

Open Existing File

Fungsi : Untuk membuka file yang telah ada dengan metode file handle.

Register Input :

AH = 3Dh

AL = Mode, dengan bit:
- 0 untuk Read only
- 1 untuk Write only
- 2 untuk Read/Write

DS:DX = Nama file ASCIIZ

Register Output :

Jika CF=0 maka
  AX=Nomor file handle
Jika CF=1 maka
  AX=Kode kesalahan

INTERRUPT 21h - Service 3Eh

Close File Handle

Fungsi : Untuk menutup file handle

Register Input :

AH = 3Eh

BX = Nomor file handle

Register Output :

CF=0 jika sukses
CF=1 jika gagal, maka
  AX=Kode kesalahan

INTERRUPT 21h - Service 3Fh

Read From File Or Device Using File Handle

Fungsi : Untuk membaca data dari suatu file atau device.

Register Input :

AH = 3Fh

BX = Nomor file handle

CX = Banyaknya byte yang akan dibaca

DS:DX = Alamat buffer

Register Output :

CF=0 jika sukses
  AX=byte yang berhasil dibaca
CF=1 jika gagal
  AX=Kode kesalahan

INTERRUPT 21h - Service 40h

Write To File Or Device Using File Handle

Fungsi : Untuk menulisi file atau device.

Register Input :

AH = 40h

BX = Nomor file handle

CX = Banyaknya byte yang akan ditulisi

DS:DX = Alamat data

Register Output :

CF=0 jika sukses
  AX=byte yang berhasil ditulisi
CF=1 jika gagal
  AX=Kode kesalahan

INTERRUPT 21h - Service 41h

Delete File Using File Handle

Fungsi : Untuk menghapus file

Register Input :	**Register Output :**
AH = 41h	CF=0 jika sukses
CL = Nama file ASCIIZ	CF=1 jika gagal, maka
	AX=kode kesalahan

INTERRUPT 21h - Service 42h

Set Current File Position

Fungsi : Untuk memindahkan pointer dari suatu file.

Register Input :	**Register Output :**
AH = 42h	CF=0 jika sukses
AL = Mode perpindahan:	CF=1 jika gagal
00 dari awal file	AX= kode kesalahan
01 dari posisi aktif	
02 dari akhir file	
BX = Nomor file handle	
CX:DX = Banyaknya perpindahan	

INTERRUPT 21h - Service 43h

Set And Get File Atribut

Fungsi : Untuk mengetahui dan merubah atribut dari suatu file.

Register Input :	**Register Output :**
AH = 43h	Jika input AL=0, maka:
AL = 0 untuk mendapatkan atribut file	jika CF=0
	CX = atribut
1 untuk merubah atribut file	jika CF=1
CX = atribut baru dengan bit:	AX = Kode kesalahan
0 = Read Only	
1 = Hidden	
2 = System	
5 = Archive	
DS:DX = Nama file ASCIIZ	

INTERRUPT 21h - Service 4Ch

Terminate With Return Code

Fungsi : Untuk menghentikan program dan mengembalikan kendali kepada DOS. Fungsi ini lebih efektif untuk digunakan dibandingkan dengan interupsi 20h.

Register Input : **Register Output :** Tidak ada

AH = 4Ch

AL = Kode return

INTERRUPT 21h - Service 56h

Rename File

Fungsi : Untuk mengganti nama file. Fungsi ini juga bisa memindahkan file ke directory lain.

Register Input :	**Register Output :**
AH = 56h	CF=0 jika sukses
DS:DX = Nama file ASCIIZ lama	CF=1 jika gagal
ES:BX = Nama file ASCIIZ baru	AX=Kode kesalahan

INTERRUPT 27h

Terminate And Stay Residen

Fungsi : Untuk meresidenkan program.

Register Input :	**Register Output :**
DS:DX = Batas alamat residen	

Lampiran II

Tabel Kode Scan Keyboard

KODE SCAN								
TOMBOL	SCAN KODE TEKAN	SCAN KODE LEPAS	TOMBOL	SCAN KODE TEKAN	SCAN KODE LEPAS	TOMBOL	SCAN KODE TEKAN	SCAN KODE LEPAS
Esc	01	81	S	1F	9F	F3	3D	BD
1	02	82	D	20	A0	F4	3E	BE
2	03	83	F	21	A1	F5	3F	BF
3	04	84	G	22	A2	F6	40	C0
4	05	85	H	23	A3	F7	41	C1
5	06	86	J	24	A4	F8	42	C2
6	07	87	K	25	A5	F9	43	C3
7	08	88	L	26	A6	F10	44	C4
8	09	89	;	27	A7	Num Lock	45	C5
9	0A	8A	'	28	A8	Scrol		
0	0B	8B	~	29	A9	Lock	46	C6
-	0C	8C	ShifKiri	2A	AA	Home	47	C7
=	0D	8D	\	2B	AB	_ <8>	48	C8
Back			Z	2C	AC	PgUp	49	C9
Space	0E	8E	X	2D	AD	<->KeyPad	4A	CA
Tab	0F	8F	C	2E	AE	<_>KeyPad	4B	CB
Q	10	90	V	2F	AF	<5>KeyPad	4C	CC
W	11	91	B	30	B0	<_>KeyPad	4D	CD
E	12	92	N	31	B1	<+>KeyPad	4E	CE
R	13	93	M	32	B2	<1>KeyPad	4F	CF
T	14	94	<	33	B3	<_>KeyPad	50	D0
Y	15	95	>	34	B4	PgDn	51	D1
U	16	96	?	35	B5	Ins	52	D2
I	17	97	Shif			Del	53	D3
O	18	98	Kanan	36	B6	F11	57	D7
P	19	99	PrtSc	37	B7	F12	58	D8
{	1A	9A	LftAlt	38	B8	RgtAlt	E038	E0B8
}	1B	9B	Space	39	B9	RgtCtrl	E01B	E09D
Enter	1C	9C	CapsLock	3A	BA	Enter	E01C	E09C
LftCtrL	1D	9D	F1	3B	BB	<KeyPad>		
A	1E	9E	F2	3C	BC			

Lampiran III

Tabel Kode Extended Keyboard

KODE EXTENDED					
TOMBOL	KODE <DES>	TOMBOL	KODE <DES>	TOMBOL	KODE <DES>
Shift+ Tab	15	F5	63	Ctrl + F7	100
Alt + Q	16	F6	64	Ctrl + F8	101
Alt + W	17	F7	65	Ctrl + F9	102
Alt + E	18	F8	66	Ctrl + F10	103
Alt + R	19	F9	67	Alt + F1	104
Alt + T	20	F10	68	Alt + F2	105
Alt + Y	21	Home	71	Alt + F3	106
Alt + U	22	Cursol Up	72	Alt + F4	107
Alt + I	23	Page Up	73	Alt + F5	108
Alt + O	24	Cursol Left	75	Alt + F6	109
Alt + P	25	Cursol Right	77	Alt + F7	110
Alt + A	30	Cursol Down	80	Alt + F8	111
Alt + S	31	Page Down	81	Alt + F9	112
Alt + D	32	Insert	82	Alt + F10	113
Alt + F	33	Delete	83	Ctrl+Cursol.lf	115
Alt + G	34	Shif + F1	84	Ctrl+Cursol.Rg	116
Alt + H	35	Shif + F2	85	Ctrl+End	117
Alt + J	36	Shif + F3	86	Ctrl+Page Down	118
Alt + K	37	Shif + F4	87	Ctrl+Home	119
Alt + L	38	Shif + F5	88	Alt + 1	120
Alt + Z	44	Shif + F6	89	Alt + 2	121
Alt + X	45	Shif + F7	90	Alt + 3	122
Alt + C	46	Shif + F8	91	Alt + 4	123
Alt + V	47	Shif + F9	92	Alt + 5	124
Alt + B	48	Shif + F10	93	Alt + 6	125
Alt + N	49	Ctrl + F1	94	Alt + 7	126
Alt + M	50	Ctrl + F2	95	Alt + 8	127
F1	59	Ctrl + F3	96	Alt + 9	128
F2	60	Ctrl + F4	97	Alt + 0	129
F3	61	Ctrl + F5	98	Alt + -	130
F4	62	Ctrl + F6	99	Alt + '	131