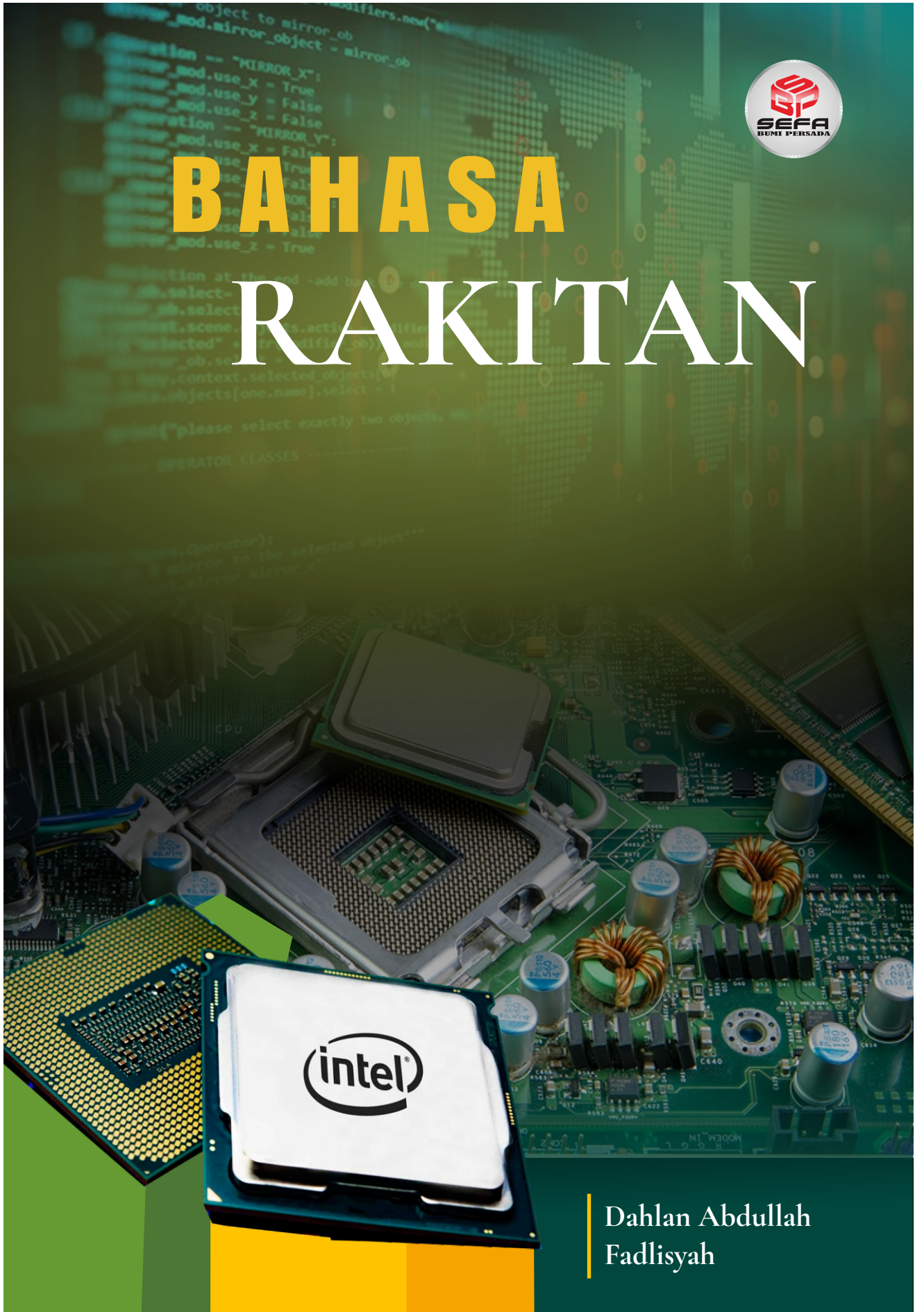
BAHASA RAKITAN



Dahlan Abdullah
Fadlisyah

Dahlan Abdullah
Fadlisyah

BAHASA RAKITAN

Diterbitkan Oleh:

CV. SEFA BUMI PERSADA - ACEH

2020

BAHASA RAKITAN

Penulis : **Dahlan Abdullah**
Fadlisyah

Penerbit:
SEFA BUMI PERSADA
Anggota Ikapi N0.021/DIA/2018
Jl. Malikussaleh No. 3
email: www.sefabumipersada.com
Telp. 085260363550

Cetakan I : Desember 2020–Aceh
ISBN: 978-623-6983-04-1
Halaman. 80
Ukuran 16,8 x 23 cm

KATA PENGANTAR

Assembly dikategorikan sebagai Bahasa tingkat rendah (low level language), yang memiliki sedikit kerumitan untuk dipahami secara selintas, sehingga saat ini Assembly seakan-akan telah terlupakan, tertutup oleh perkembangan perangkat lunak yang mudah dipelajari, yang lebih interaktif dan menarik. Walaupun keadaan membuat Assembly cenderung untuk ditinggalkan, tetapi di Universitas-Universitas yang menyelenggarakan kuliah Ilmu Komputer, Assembly tetap menjadi bagian vital dari pemrograman mikroprosesor, bahkan pada sebagian Universitas tertentu, Assembly dipelajari dalam mata kuliah khusus seperti : Bahasa Rakitan. Buku ini memfasilitasi kebutuhan tersebut, dengan pengalaman penulis sebagai staff pengajar pertama mata kuliah Mikroprosesor untuk Teknik Informatika.

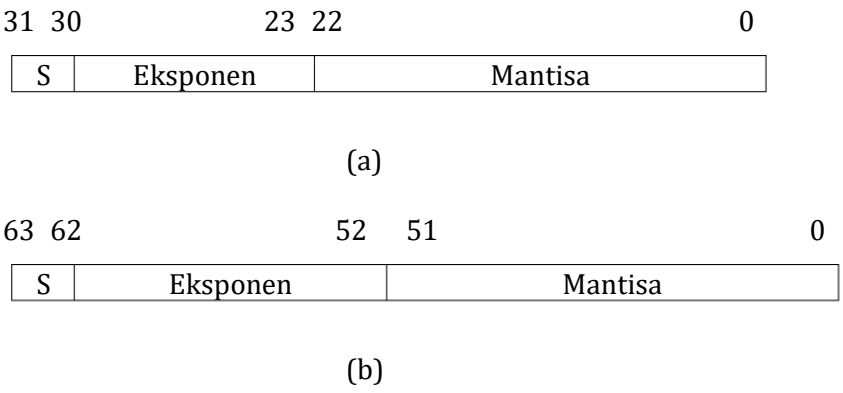
Adapun materi yang hadir di dalam buku ini meliputi : Bilangan floating point, Mikroprosesor, arsitektur, dan mode pengalamatan, Bahasa mesin, Manipulasi bit dan operasi logika, Stack, dan Pemrograman modular.

Penulis

DAFTAR ISI

BAB I
BILANGAN FLOATING POINT

Bilangan floating-point, berisikan 2 bagian, yaitu mantisa (signifikan atau pecahan), dan eksponen. Gambar 1.1 memperlihatkan bentuk 4 byte dan 8 byte dari bilangan floating-point yang disimpan di dalam sistem Intel. Bilangan floating-point atau bilangan real 4 byte disebut *single-precision*, dan Bilangan floating-point atau bilangan real 8 byte disebut *double-precision*.



Gambar 1.1 (a) Presisi tunggal menggunakan bias 7FH, dan (b) Presisi Ganda menggunakan bias 3FFH

Mantisa 24-bit memiliki bit satu yang tersembunyi (implied), yang memungkinkan mantisa untuk mewakili 24-bit walau hanya disimpan 23-bit. Bit yang disembunyikan adalah bit pertama dari bilangan yang dinormalisasi. Pada saat menormalisasi bilangan, bit ini diatur sehingga nilainya sekurang-kurangnya 1, tetapi kurang dari 2. sebagai contoh, jika 12 diubah ke biner (1102_2), maka dinormalisasi dan hasilnya $1{,}1\times 2^3$. Bit satu yang terdapat di depan tanda koma

akan disembunyikan, atau tidak disimpan di dalam bagian mantisa.

Eksponen disimpan dalam eksponen terbias (*biased exponent*). Untuk presisi tunggal, Bias 7FH (127) akan dijumlahkan ke eksponen sebelum disimpan ke dalam tempat eksponen, dan untuk presisi ganda sebesar 3FFH (1023).

Ada 2 pengecualian mengenai aturan-aturan yang diterapkan mengenai bilangan floating-point. Angka 0,0 disimpan semuanya sebagai nol. Bilangan tak hingga disimpan dalam eksponen sebagai satu, dan dalam mantisa semuanya sebagai nol. Bit tanda menunjukkan bilangan tak hingga tersebut bernilai positif atau negatif.

Kasus 1

Bagaimana bilangan +12 disimpan dalam bentuk presisi tunggal.

Penyelesaian :

Desimal	Biner	Normal	Tanda
+12	1100	$1{,}1 \times 2^3$	0

s	Eksponen							
0	1	0	0	0	0	0	1	0
31	30	29	28	27	26	25	24	23

Mantisa																							
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Penjelasan :

+ = 0, pangkat pada normalisasi adalah 3 atau dalam bentuk biner sama dengan 11_2. Selanjutnya untuk presisi

tunggal eksponen dapat diperoleh dengan menambahkan 11_2 +7FH = $11_2 + 1111111_2 = 10000010_2$.

Kasus 2

Bagaimana bilangan -12 disimpan dalam bentuk presisi tunggal.

Penyelesaian :

Desimal	Biner	Normal	Tanda
-12	1100	$-1{,}1 \times 2^3$	1

s	Eksponen							
1	1	0	0	0	0	0	1	0
31	30	29	28	27	26	25	24	23

Mantisa																							
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Kasus 3

Bagaimana bilangan +100 disimpan dalam bentuk presisi tunggal.

Penyelesaian :

Desimal	Biner	Normal	Tanda
+100	1100100	$1{,}1001 \times 2^6$	0

s	Eksponen							
0	1	0	0	0	0	1	0	1
31	30	29	28	27	26	25	24	23

	Mantisa																						
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Penjelasan :

Konversi bilangan 100 menjadi bilangan biner adalah sebagai berikut :

100/2	Sisa 0	3/2	Sisa 1
50/2	Sisa 0	1/2	Sisa 1
25/2	Sisa 1	0	
12/2	Sisa 0		
6/2	Sisa 0		1100100

<u>Kasus 4</u>

Bagaimana bilangan -1,75 disimpan dalam bentuk presisi tunggal.

Penyelesaian :

Desimal	Biner	Normal	Tanda
-1,75	1,11	$-1{,}11\times 2^{0}$	1

s	Eksponen							
1	0	1	1	1	1	1	1	1
31	30	29	28	27	26	25	24	23

	Mantisa																						
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Penjelasan :

Konversi bilangan -1,75 menjadi bilangan biner adalah sebagai berikut: (kita ambil 0,75, 1 desimal = 1 biner), sampai di sini kita sudah dapat menyatakan sebagai 1,...

0,75 x 2		
0,5 x 2	Digit 1	
1,0	Digit 1	
		1,11

Kasus 5

Bagaimana bilangan +0,25 disimpan dalam bentuk presisi tunggal.

Penyelesaian :

Desimal	Biner	Normal	Tanda
+0,25	0,01	$1,\times 2^{-2}$	0

s	Eksponen							
0	0	1	1	1	1	1	0	1
31	30	29	28	27	26	25	24	23

Mantisa

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Penjelasan :

Konversi bilangan +0,25 menjadi bilangan biner adalah sebagai berikut: (kita ambil 0,25, 0 desimal = 0 biner), sampai di sini kita sudah dapat menyatakan sebagai 0,...

0,25 x 2		
0,5 x 2	Digit 0	
1,0	Digit 1	
		0,01

Eksponen terbias kita peroleh dengan menjumlahkan :

7FH	1111111
-2	-
	0000010
	1111101

Berbagai kasus yang lain, ditinggalkan untuk ujicoba para mahasiswa,
Andai diketahui :

Ujicoba 1

s	Eksponen							
0	1	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23

Mantisa

1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Maka ubahlah bilangan floating-point di atas ke dalam bilangan desimal.

Ujicoba 2

s	Eksponen							
1	0	1	1	1	1	1	1	1
31	30	29	28	27	26	25	24	23

Mantisa

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Maka ubahlah bilangan floating-point di atas ke dalam bilangan desimal.

Ujicoba 3

s	Eksponen							
0	1	0	0	0	0	0	1	0
31	30	29	28	27	26	25	24	23

Mantisa																							
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Maka ubahlah bilangan floating-point di atas ke dalam bilangan desimal.

Keseluruhan bahan dan berbagai kasus yang menyertai bab ini, disadur dari buku Brey, Barry B. 2000. *Mikroprosesor Intel 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, dan Pentium-Pro : Arsitektur, Pemrograman, Antarmuka (Edisi kelima jilid 1)*, Penerbit Erlangga, yang diterjemahkan oleh Ir. N.R. Poespawati, M.T.

BAB II
MIKROPROSESOR, ARSITEKTUR DAN MODE PENGALAMATAN

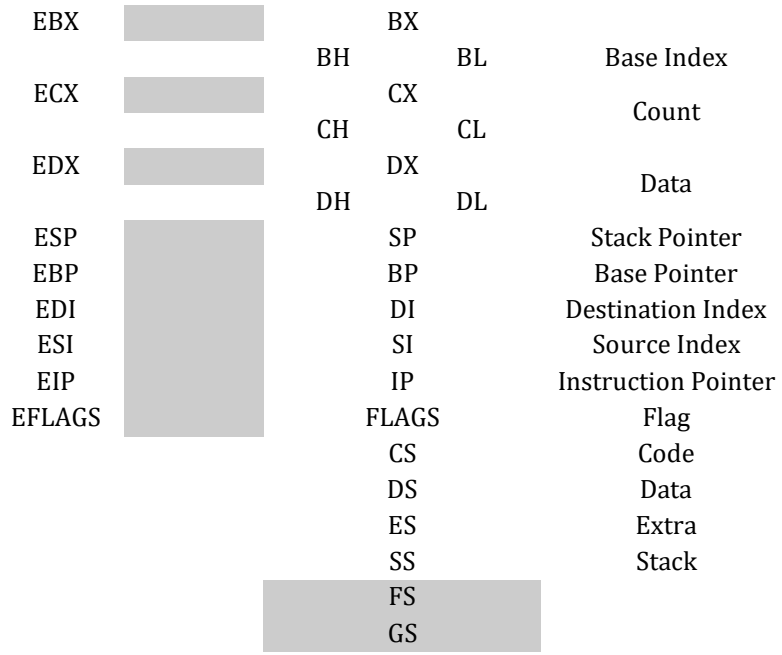
A. Arsitektur Internal Mikroprosesor

1. Model Pemrograman

Model pemrograman 8086 sampai Pentium termasuk visibel program (terlihat-program) karena register-registernya digunakan selama pemrograman aplikasi dan ditentukan oleh instruksi. Register lainnya merupakan invisibel program (tak terlihat-program) karena register-register itu tidak mampu dialamati selama pemrograman aplikasi, namun dapat diganakan tak langsung selama penmrograman sistem. Hanya mikroprosessor 80286 ke atas yang berisi register-register invisibel program yang digunakan untuk mengontrol dan mengoperasikan sistem memori terlindung.

Gambar 2.1 menguraikan model pemrograman mikroprosessor 8086 sampai Pentium. 8486, 8088, dan 80286 sebelumnya berisi arsitekiur internal 16-bit, subset dari register yang diperlihatkan dalam Gambar 2.1. Mikroprosesor 84386, 80486, Pentium, Pentium Pro, dan Pentium II terdiri dari arsitektur internal 32-bit penuh. Arsitektur 8086 sampai 80286 sebelumnya sepenuhnya kompatibel ke atas dengan 80386 sampai Pentium 4. Area yang diarsir dalam ilustrasi ini mewakili register-register yang tidak tersedia pada mikroprosesor 8086, 8088, atau 80286.

Nama 32-bit		**Nama 16 bit**		**keterangan**
		8 bit	8 bit	
EAX		AX		Akumulator
		AH	AL	



Gambar 2.1 Model pemrograman mikroprosesor Intel, register-register yang diarsir hanya terdapat pada mikroprosesor 80386 – Pentium 4, dan register FS dan GS tidak memiliki nama khusus.

Model pemrograman berisi register-register 8-, 16-, dan 32-bit. Register-register 8-bit adalah AH, AL, BH, BL, CH, CL, DH, DL dan dirujuk pada saat satu instruksi dibuat menggunakan perancangan 2-huruf. Sebagai contoh, instruksi ADD AL,AH menambah isi 8-bit isi register AH pada isi register AL. (Hanya isi register AL yang berubah karena instruksi ini.) Register-register 16 bit adalah AX, BX, CX, DX, SP, BP, DI, Sl, IP, FLAGS, CS, DS, ES, SS, FS, dan GS. Register-register ini juga dirujuk dengan perancangan dua-huruf. Sebagai contoh, instruksi ADD DX,CX rnenambah isi register 16-bit CX pada isi register DX. (Hanya isi register DX yang berubah karena instruksi ini.) Register-register extended 32-bit

berlabel EAX, EBX, ECX, EDX, ESP, EBP, EDI, ESI, EIP, dan EFLAGS. Register ekstended 32-bit dan register GS dan FS 16-bit hanya terdapat pada mikroprosesor 80386 ke atas. Register-register ini dirujuk oleh FS dan GS pada dua register 16-bit baru, dan dengan perancangan tiga-huruf pada register-register 32-bit. Sebagai contoh, instruksi ADD ECX,EBX menambah isi register EBX 32-bit pada isi register ECX. (Hanya isi register ECX yang berubah karena instruksi- ini.)

Ada beberapa register yang tergolong *general purpose* (serba guna) atau *multipurpose* (multiguna), sedangkan yang lain mempunyai kegunaan yang khusus. Yang termasuk dalam register-register multiguna antara lain EAX, EBX, ECX, EDX, EBP, EDI, dan ESI. Register-register tersebut mampu menyimpan data berbagai ukuran (byte, word, atau doubleword) dan digunakan hampir untuk semua tujuan, seperti tertulis di program.

EAX EAX dirujuk sebagai register 32-bit (EAX), sebagai register (akumulatar) 16-bit (AX), atau 2 register 8-bit (AH dan AL). Akumulator digunakan untuk instruksi-instruksi perkalian, pembagian dan beberapa instruksi penyesuaian. Dalam mikroprosesor 80386 ke atas, register EAX juga dapat rnenyimpan alamat offset sebuah lokasi dalam sistem memori.

EBX EBX dapat dialamati sebagai EBX, BX, BH, BL. Register (base index) BX dapat menyimpan alamat offset sebuah lokasi memori dalam sistem memori semua versi mikroprosesor. Dalam nukroprosesor 80386 ke atas, EBX juga dapat mengalamati data memori.

ECX ECX merupakan register serbaguna yang juga dapat digunakan untuk instruksi perhitungan. Dalam mikroprosesor 80386 ke atas, register ECX dapat juga menyimpan alamat offset data memori. Instruksi-instruksi yang menggunakan penghitungan merupakan instruksi string yang diulang (REP/REPE/REPNE); dan instruksi pergeseran, perputaran, dan instruksi LOOP/LOOPD. Instruksi pergeseran dan perputaran menggunakan CL, instruksi string berulang

memakai CX, dan instruksi LOOP/LOOPD menggunakan CX atau ECX.

EDX EDX merupakan register serbaguna yang dapat digunakan untuk (data) menyimpan hasil perkalian dan menyimpan sisa dalam pembagian. Dalam mikroprosesor 80386 ke atas, register ini juga mampu mengalamati data memori.

EBP EBP menunjuk sebuah lokasi memori yang dalam semua versi mikroprosesar digunakan untuk mentransfer data memori, Register ini dialamati sebagai EBP dan BP.

EDI EDI sering mengalamati data tujuan string untuk beberapa instruksi string. EDI juga berfungsi sebagai register serbaguna 32-bit (EDI) atau 16-bit (DI).

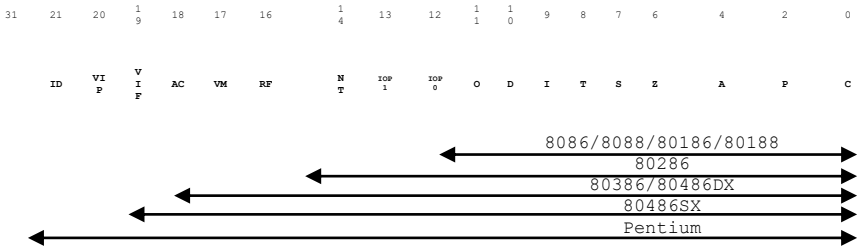
ESI ESI digunakan sebagai ESI dan sebagai SI. Register indeks sumber sering mengalamati sumber data string untuk beberapa instruksi string. Seperti EDI, ESI juga berfungsi sebagai register serbaguna. Sebagai register 16-bit dialamati sebagai SI; sebagai register 32-bit dialamati sebagai ESI.

EIP EIP mengalamati instruksi selanjutnya dalam sebuah bagian memori yang didefinisikan sebagai segmen kode. Register ini merupakan 1P (16-bit) ketika mikroprosesor dioperasikan dalam mode real namun menjadi EIP (32-bit) bila mikroprosesor 80386 ke atas dioperasikan dalam mode terlindung. Penunjuk instruksi, yang menunjuk ke instruksi selanjutnya dalam satu program, digunakan oleh mikroprosesor untuk mendapatkan instruksi berurutan berikutnya dalam suatu program yang lokasinya terletak di dalam segmen kode. Penunjuk instruksi dapat dimodifikasi dengan instruksi jump atau call.

ESP ESP mengalamati sebuah lokasi memori yang disebut stack. Memori stack menyimpan data melalui penunjuk ini dan akan dijelaskan dengan instruksi yang mengalamati data stack. Register disebut SP jika digunakan sebagai register l6-bit dan disebut register ESP jika digunakan sebagai register 32-bit.

EFLAGS EFLAGS menunjukkan kondisi mikroprosesor dan mengontrol operasinya. Gambar 2.2 memperlihatkan register-register flag semua versi mikroprosesor. Register FLAG (l6-bit) terdapat dalam mikroprosesor 8086-80286 dan register EFLAG (32-bit) yaitu extended FLAG terdapat dalam mikroprosesor 80386 ke atas.

Lima bit flag yang paling kanan dan flag overflownya berubah setelah beberapa instruksi aritmetik dan logik dijalankan. Flag tidak akan berubah dengan adanya transfer data atau operasi kontrol program. Beberapa flag juga digunakan untuk mengontrol beberapa fitur yang terdapat dalam mikroprosesor. Selanjutnya dijelaskan secara singkat fungsi-fungsi setiap bit flag. Instruksi-instruksi diperkenalkan dalam pembahasannya berikutnya, rincian tambahan tentang bit flag akan disediakan. Lima flag terkanan dan flag overflow diubah oleh sebagian besar operasi aritmatika dan logik, sedangkan transfer data tidak mempengaruhinya.



Gambar 2.2 EFLAG dan FLAG count register untuk seluruh kerabat mikroprosesor 80x86 dan pentium

C (Carry) Carry menyimpan carry setelah penambahan, atau borrow setelah pengurangan. Flag carry juga menunjukkan kondisi error scperti yang diperintahkan oleh program dan prosedur.

P (Paritas) Paritas merupakan 0 logika untuk paritas ganjil dan 1 logika untuk paritas genap. Paritas adalah jumlah angka satu dalam bilangan yang menyatakan genap atau ganjil. Jika sebuah bilangan terdiri 3 bilangan biner bit satu, maka termasuk paritas ganjil. Jika terdiri dari nol bit satu, maka termasuk paritas genap. Flag paritas jarang ditemukan dalam aplikasi pemrograman

	modern dan telah diimplementasikan dalam mikroprosessor Intel generasi pertama untuk mengecek data dalam komunikasi data. Dewasa ini pengecekan paritas tidak dilakukan oleh mikro-prosesor, tetapi oleh peralatan komunikasi data.
A (Auxiliary Carry)	Carry tambahan (auxiliary carry) menampung carry (carry setengah) setelah penambahan, atau borrow setelah pengurangan antar posisi bit 3 dan 4. Flag bit khusus ini diuji oleh instruksi DAA dan DAS untuk menyesuaikan nilai AL setelah penambahan atau pengurangan suatu BCD. Di luar itu bit flag A tidak digunakan oleh mikroprosesor atau instruksi lain.
Z (Zero)	Flag not menunjukkan bahwa jumlah dari suatu operasi aritmetika atau logik adalah nol. Jika Z = 1, jumlahnya adalah nol, jika Z = 0 maka jumlahnya adalah bukan nol.
S (Sign)	Flag tanda akan menampung tanda aritmetika dari hasil setelah instruksi aritmatika atau instruksi logik dieksekusi. Jika S = 1, tanda bit negatif, jika S = 0, tanda bit positif.
T (Trap)	Flag trap memungkinkan trapping melalui suatu chip debugging. (Suatu program didebug untuk mencari error atau bug). Jika flag T enabel (= 1), mikroprosesor akan menghentikan alur program pada keadaan yang diindikasikan oleh register debug dan register kontrol. Jika flag T-nya nol logik, fitur trappingnya (debugging) adalah disabel. Program CodeView dapat menggunakan fitur trap dan register debug untuk mendebug kesalahan software.
I (Interrupt)	Flag interrupt ini mengendalikan operasi dari pin input INTR (interrupt request). Jika I = 1 pin INTRnya enabel; jika I = 0 pin INTRnya disabel. Kondisi dari bit flag I dikontrol oleh instruksi STI (set I flag) dan CLI (clear I flag).
D (Direction)	Flag arah ini memilih salah satu dari mode penambahan atau pengurangan untuk register DI dan/atau register SI selama instruksi string. Jika D = 1, register secara otomatis akan dikurangi; jika D = 0, register secara otomatis akan ditambah. Flag D diset dengan instruksi STD (set direction) dan diclear dengan instruksi CLD (clear direction).

O (Overflow)	Overflow terjadi ketika bilangan bertanda ditambah atau dikurang. Suatu overflow menunjukkan hasilnya melebihi kapasitas dari mesin. Contohnya jika 7FH (+127) ditambah, dengan menggunakan penambahan 8 bit, pada O1H (+1) hasilnya adalah 80H (-128). Hasil ini menunjukkan suatu kondisi overflow yang ditunjukkan oleh flag overflow untuk penambahan bertanda. Untuk operasi tak bertanda, flag overflow diabaikan.
IOPL (I/O Privilege Level)	IOPL digunakan dalam operasi mode terlindung untuk memilih tingkatan operasi istimewa untuk peranti I/O. Jika arus tingkatan istimewa lebih tinggi atau lebih menjamin daripada IOPL, eksekusi I/O berjalan tanpa gangguan. Jika IOPL lebih rendah dari tingkatan istimewa sekarang, terjadi suatu interrupt yang, menyebabkan eksekusi tertunda. Perlu diperhatikan bahwa IOPL 00 paling tinggi atau paling menjamin, sedangkan IOPL 11 adalah yang paling rendah atau paling tak-terpercaya.
NT (Nested Task)	Flag NT menandakan bahwa task yang sedang dilaksanakan bersarang dalam task yang lain dalam operasi mode terlindung. Flag ini diset ketika task di-nest oleh perangkat lunak.
RF (Resume)	Flag resume digunakan bersama debugging untuk mengontrol kelanjutan eksekusi setelah instruksi berikutnya.
VM (Virtual Mode)	Bit flag VM memilih operasi mode virtual dalam suatu sistem mode terlindung. Suatu sistem mode virtual memungkinkan partisi memori dari DOS yang panjangnya 1M byte hadir bersamaan dalam sistem memori. Pada dasarnya, model ini memampukan program sistem mengeksekusi lebih dari satu program DOS.
AC (Alignment Check)	Bit flag AC aktif jika suatu word atau doubleword dialamatkan pada batas suatu nonword atau nondoubleword. Hanya mikroprosesor 80486SX berisi bit AC yang umum digunakan oleh koprosesor numerik 80487SX pendampingnya untuk sinkronisasi.
VIF (Virtual Interrupt Flag)	VIF adaiah suatu salinan bit flag interrupt yang digunakan pada prosesor Pentium atau Pentium Pro.
VIP (Virtual	VIP menyediakan informasi tentang suatu interrupt

Interrupt Pending)	mode interrupt virtual untuk mikroprosesor Pentium. Ini digunakan dalam pemakaian multitasking untuk memberikan flag virtual interrupt dan informasi pending interrupt pada sistem operasi.
ID (Identification)	Flag ID menunjukkan bahwa mikroprosesor Pentium mendukung instruksi CPUID. Instruksi CPUID memberikan informasi tentang mikroprosesor, sesuai dengan versi dan nomor seri dari pabriknya, pada sistem.

Register Segmen. Register tambahan, disebut juga register segmen, menghasilkan alamat-alamat memori saat dikombinasikan dengan register-register lainnya dalam mikroprosesor. Bisa ada empat atau enam register segmen dalam bermacam versi mikroprosesor. Fungsi register segmen berbeda dalam mode real, bila dibandingkan dengan operasi mode terlindung, dari mikroprosesor. Berikut ini daftar dari masingmasing register segmen berkaitan dengan fungsinya dalam sistem.

CS (Code)	Segmen kode adalah suatu bagian dari memori yang memuat kode (program dan prosedur) yang digunakan oleh mikroprosesor. Register CS mendefinisikan alamat awal dari bagian kode pemuatan memori. Pada operasi mode real, CS menetapkan awal dari suatu bagian memori 64K byte; dalam mode terlindung, CS memilih suatu pendeskripsi yang menggambarkan alamat awal dan panjang dari suatu bagian code pemuatan memori. Segmen kode panjangnya terbatas sampai 64K byte dalam 8086-80286 dan 4G byte dalam 80386 dan di atasnya ketika mikroprosesor ini beroperasi pada mode terlindung.
DS (Data)	Segmen data adalah bagian dari memori yang berisi sebagian besar data yang digunakan oleh program. Data diakses dalam segmen data oleh alamat offset atau isi dari register lain yang mempunyai alamat offset. Seperti halnya segmen kode dan segmen yang lain panjangnya dibatasi sampai 64K byte dalam 8086-80286 dan 4G byte

dalam mikroprosesor 80386 dan di atasnya.

ES (Extra)	Segmen ekstra adalah suatu penambahan segmen data yang digunakan oleh beberapa instruksi string untuk menyimpan data tujuan.
SS (Stack)	Segmen stack mendefinisikan area memori yang digunakan untuk stack. Posisi dari arus masuk dalam segmen stack dibatasi oleh register penunjuk stack. Register BP juga mengalamati data dalam segmen stack.
FS dan GS	Segmen FS dan GS adalah register segmen tambahan yang tersedia pada mikroprosesor 80386, 80486, Pentium dan Pentium Pro, yang memungkinkan dua segmen rnemori tambahan diakses oleh program-program yang dijalankan.

B. Segmen dan Offset

Kombinasi dari suatu alamat segmen dan alamat offset mengakses lokasi memori pada mode real. Semua alamat memori mode real terdiri dari alamat segmen dan alamat offset. Alamat segmen berada dalam satu register segmen, menetapkan alamat awal dari segmen memori 64K byte. Alamat offset memilih sembarang lokasi yang memiliki segmen memori 64K byte itu. Gambar 2.3 menunjukkan bagaimana skema pengalamatan segmen dan offset dalam memilih lokasi memori. Gambaran ini memperlihatkan suatu segmen memori yang berawal pada lokasi 10000H dan berakhir pada lokasi 1FFFFH- dengan panjang 64K; juga memperlihatkan alamat offset, yang kadang-kadang disebut displacement, F000H memilih lokasi 1F000H dalam sistem memori. Perhatikan bahwa offset atau displacement merupakan jarak di atas awal segmen, seperti ditunjukkan dalam Gambar 2.3.

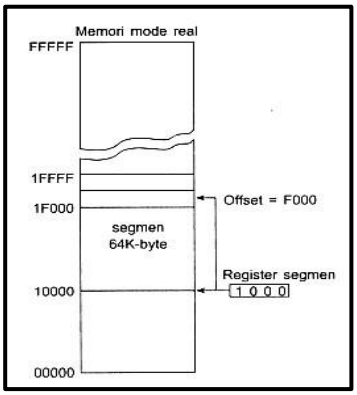
Register segmen dalam Gambar 2.3 berisi suatu alamat 1000H, tetapi alamat ini memiliki segmen awal pada lokasi 10000H. Dalam mode real, masing-masing register segmen diberi tanda dengan 0H bagian akhir sebelah kanan. Ini membentuk suatu alamat memori 24-bit, mengijinkannya untuk mengakses awal segmen. Mikroprosesor harus

menghasilkan alamat memori 20-bit untuk mengakses sebuah lokasi dalam memori 1M pertama. Contohnya jika suatu register segmen berisi 1200H, berarti segmen memori 64K byte berawal pada lokasi 12000H. Sama halnya jika register segmen berisi 1201H, berarti segmen memorinya akan berawal pada lokasi 12010H. Karena ditambahkan dengan 0H, segmen mode real hanya dapat bermula pada batasan 16 byte dalam sistem metnori. Batasan 16 byte ini sering disebut paragraf.

Karena suatu segmen mode real dari memori yang panjangnya 64K, maka begitu alamat awalnya diketahui, akan diketahui juga alamat akhirnya dengan menambahkan FFFFH. Sebagai contoh,, jika register segmen berisi 3000H, alamat pertama dari segmen ini adalah 30000H dan alamat akhirnya 30000H + FFFFH atau 3FFFFH. Tabel berikut menunjukkan beberapa contoh dari register segmen yang berawal dan berakhir pada alamat segmen memori yang dipilih oleh masing-masing alamat segmen.

Register Segmen	Alamat Awal	Alamat Akhir
2000H	20000H	2FFFFH
2001H	20010H	3000FH
2100H	21000H	30FFFH
AB00H	AB000H	BAFFFH
1234H	12340H	2233FH

Alamat offset ditambahkan pada awal segrnen untuk mengalamati memori dalam segmen memori. Sobagai contoh, jika alamat segmennya adalah 1000H dan alamat offsetnya adalah 2000H, mikroprosesor mengalamati lokasi memori ini 12000H. Alamat segmen dan offsetnya kadang-kadang ditulis sebagai 1000:2000 untuk alamat segmen dari 1000H dengan alamat offset 2000H.



Gambar 2.3 Skema pengalamatan memori mode real, menggunakan alamat segmen : offset.

Kombinasi alamat segmen dan offset 16 bit default untuk mikroprosesor 8086 – pentium, dirangkum dalam tabel berikut :

Register Segmen	Offset	Tujuan
CS	IP	Alamat Instruksi
SS	SP atau BP	Alamat Stack
DS	BX, DI, SI, bilangan 8 bit atau 16 bit	Alamat Data
ES	DI untuk instruksi string	Alamat Tujuan String

Kombinasi alamat segmen dan offset 32 bit default untuk mikroprosesor 8086 – pentium, dirangkum dalam tabel berikut :

Register Segmen	Offset	Tujuan
CS	EIP	Alamat Instruksi
SS	ESP dan EBP	Alamat Stack
DS	EAX, ECX, EBX, EDX, EDI, ESI, bilangan 8 bit atau 32 bit	Alamat Data
ES	EDI untuk instruksi string	Alamat Tujuan String

FS	Tanpa default	Alamat umum
GS	Tanpa default	Alamat umum

C. Mode Pengalamatan Data

Untuk merepresentasikan mode pengalamatan data, maka kita pilih instruksi MOV sebagai landasan dasar untuk menerangkan mode pengalamatan data, sebagai berikut :

Tipe	Instruksi	Sumber	Pembangkit alamat	Tujuan
Register	**MOV AX,BX**	Register BX		Register AX
Immediate	**MOV CH, 3AH**	Data 3AH		Register CH
Direct	**MOV [1234H],AX**	Register AX	DS x 10H + DISP 10000 H+1234H	Alamat memori 11234H
Register tidak langsung	**MOV [BX],CL**	Register CL	DS x 10H + BX 10000H+0300H	Alamat memori 10300H
Base-plus-index	**MOV [BX+SI],BP**	Register BP	DS x 10H + BX +SI 10000H+0300H+ 0200H	Alamat memori 10500H
Register relatif	**MOV CL, [BX+4]**	Alamat memori 10304H	DS x 10H + BX +4 10000H+0300H+4	Register CL
Base relatif-plus-index	**MOV ARRAY[BX+SI], DX**	Register DX	DS x 10H + ARRAY+ BX +SI 10000H+1000H+ 0300H+ 0200H	Alamat memori 11500H
Indeks berskala	**MOV [EBX+2 x ESI], AX**	Register AX	DS x 10H + EBX+2 x ESI 10000H+00000300H+00000400H	Alamat memori 10700H

Catatan : EBX = 00000300H, ESI = 00000200H, ARRAY = 1000H, DS = 1000H.

Keseluruhan bahan dan berbagai kasus yang menyertai bab ini, disadur dari buku Brey, Barry B. 2000. *Mikroprosesor Intel 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, dan Pentium-Pro : Arsitektur, Pemrograman, Antarmuka (Edisi kelima jilid 1)*, Penerbit Erlangga, yang diterjemahkan oleh Ir. N.R. Poespawati, M.T.

BAB III
BAHASA MESIN

Bahasa mesin adalah kode biner asli (native) yang dapat dimengerti oleh mikroprosesor dan digunakan sebagai instruksi-instruksi kontrol dalam operasinya. Instruksi bahasa mesin untuk 8086 hingga Pentium sangat beraneka ragam panjangnya, mulai dari satu hingga tiga belas byte. Meski bahasa mesin terlihat sulit, bahasa inilah yang digunakan dalam aplikasi mikroprosesor. Ada lebih dari 100.000 variasi dari instruksi bahasa mesin, dan tidak ada daftar lengkap dari semua variasi yang ada. Karenanya beberapa bit biner dalam instruksi bahasa mesin diberikan di sini, dan yang lainnya ditentukan dari setiap variasi instruksi.

Instruksi dalam 8086 hingga 80286 adalah mode insiruksi 16-bit yang bentuknya dapat dilihat dalam Gambar 3.1(a). Mode instruksi 16-bit akan kompatibel dengan 80386 dan di atasnya bila mikroprosesor itu diprogram untuk beroperasi dalam mode instruksi 16-bit, tetapi juga diprefikskan seperti pada Gambar 3.1(b). Dengan 80386 ke atas, diasumsikan bahwa semua mode instruksi tipe 16-bit-lah yang dieksekusi ketika mikroprosesor beroperasi dalam mode real. Dalam mode protected, byte teratas dari deskriptor (pendeskripsi) berisi D-bit yang akan memilih mode instruksi 16-bit atau 32 bit. Pada saat ini hanya Windows NT, Windows 95, Windows 98 OS/2, dan Versi Windows di atasnya-lah yang beroperasi dalam mode instruksi 32-bit. Mode instruksi 32-bit diperlihatkan dalam Gambar 3.1(b). Instruksi ini sesungguhnya terjadi dalam mode instruksi 16-bit yang mempergunakan prefiks.

Dua byte pertama dari format mode insiruksi 32-bit disebut override prefix, hal ini disebabkan karena 2 byte ini tidak selalu ada. Byte yang pertama memodifikasi ukuran

alamat operand yang digunakan oleh instruksi, dan byte kedua memodifikasi ukuran register. Apabila 80386 hingga Pentium 4 beroperasi dalam mode mesin instruksi 16-bit (mode real atau mode protected) dan register 32-bit digunakan, maka ditambahkanlah register-size prefix (66H) di depan baris instruksi. Bila beroperasi dalam mode instruksi 32-bit (mode protected saja), dan register 32-bit yang digunakan, maka. register-size prefix absen. Jika register 16-bit muncul dalam sebuah instruksi dalam mode instruksi 32-bit, maka register-size prefix muncul untuk memilih register 16-bit. Address-size prefix (67H) digunakan dalam bentuk yang sama, dan hal ini akan dijelaskan pada akhir bab ini. Prefiks mengubah ukuran register dan alamat operand dari 16-bit hingga 32-bit atau dari 32-bit ke 16-bit untuk instruksi prefiks.

(a) Mode Instruksi 16-bit

OPCODE 1-2 Byte	**MOD-REG-R/M 0-1 Byte**	**Displacement 0-1 Byte**	**Immediate 0-2 Byte**

(b) Mode Instruksi 32-bit (80386 – Pentium 4)

	Ukuran Alamat 0-1 Byte	**Ukuran operand 0-1 Byte**	
OPCODE 1-2 Byte	**MOD-REG-R/M 0-1 Byte**	**Displacement 0-4 Byte**	**Immediate 0-4 Byte**

Gambar 3.1 Format instruksi pada 8086-Pentium

Opcode : memilih operasi.

D W

Keterangan :

D=1	R/M ke Register
D=0	Register ke R/M

W=1	Data berukuran word atau doubleword
W=0	Data berukuran byte

MOD-REG-R/M

MOD	**REG**	**R/M**

Keterangan :

MOD : memilih tipe pengalamatan, dan menentukan kehadiran dan ketidakhadiran displacement. Tabel berikut berlaku untuk operasi 16 bit.

00	Tidak displacement
01	Displacement extended-sign 8-bit
10	Displacement 16-bit
11	R/M adalah register

Untuk 32 bit, berlaku tabel MOD berikut :

00	Tidak displacement
01	Displacement extended-sign 8-bit
10	Displacement 32-bit
11	R/M adalah register

REG dan **R/M**

Kode	**W=0 (byte)**	**W=1 (Word)**	**W=1 (DoubleWord)**
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

Kasus : tentukan bahasa mesin untuk instruksi MOV BP,SP ?

Solusi :

Opcode : MOV = 100010

1 0 0 0 1 0 **D** **W**

Pemindahan ke register BP, D=1

W=1, BP dan SP adalah word.

1 0 0 0 1 0 **1** **1**

R/M = SP, register, maka MOD=11

MOD **REG** **R/M**

1 1

REG BP=101, dan R/M SP=100, sehingga

MOD **REG** **R/M**

1 1 1 0 1 1 0 0

Maka bahasa mesin MOV BP, SP adalah :

1 0 0 0 1 0 **1** **1**

MOD **REG** **R/M**

1 1 1 0 1 1 0 0

Mulai dari atas : 1000 = 8, 1011 =B, 1110 =E, dan 1100=C, atau 8BECH.

Apabila field MOD berisi 00, 01, 10, maka field R/M akan mempunyai arti yang baru. Tabel berikut memuat daftar mode pengalamatan memori untuk field R/M ketika MOD berisi 00, 01, 10 untuk instruksi 16 bit.

Kode R/M	Mode Pengalamatan
000	DS:[BX+SI]
001	DS:[BX+DI]
010	SS:[BP+SI]
011	SS:[BP+DI]
100	DS:[SI]
101	DS:[DI]
110	SS:[BP]
111	DS:[BX]

Kasus : MOV DL, [DI]
Solusi :
Opcode = MOV
D = Pemindahan ke REG
W = Byte
MOD = tidak ada displacement
REG = DL
R/M = DS:[DI]

1	0	0	0	1	0	**1**	**0**
MOD			**REG**			**R/M**	
0	0	0	1	0	1	0	1

Mulai dari atas : 1000 = 8, 1010 =A, 0001 = 1, dan 0101 = 5, atau 8A15H.

Jika instruksi kita rubah sedikit menjadi MOV DL,[DI+1], maka bahasa mesinnya,

1	0	0	0	1	0	**1**	**0**
MOD			**REG**			**R/M**	
0	1	0	1	0	1	0	1

Displacement

0	0	0	0	0	0	0	1

Atau setara dengan 8A5501H

Untuk MOV DL,[DI+1000], maka bahasa mesinnya adalah :

1	0	0	0	1	0	**1**	**0**
MOD			**REG**			**R/M**	
1	0	0	1	0	1	0	1

Displacement rendah

0 0 0 0 0 0 0 0

Displacement tinggi

0 0 0 1 0 0 0 0

1000 = 8, 1010 = A, 1001 = 9, 0101 = 5, 0010 atau 8A950010H.

Mode Pengalamatan Khusus. Ada mode pengalamatan yang tidak muncul dalam berbagai tabel sebelumnya. Hal ini terjadi ketika data memori direferensikan hanya dengan mode displacement dari pengalamatan untuk instruksi 16-bit. Contohnya adalah instruksi MOV [1000H],DL dan MOV NUMB,DL. Instruksi pertama memindahkan isi register DL ke lokasi memori segmen data 1000H. Instruksi kedua memindahkan register DL ke lokasi memori segmen data simbolis NUMB.

Ketika instruksi hanya mempunyai satu displacement, maka field MOD selalu 00 dan field R/M selalu 110. Seperti diperlihatkan dalam tabel, instruksi tidak berisi displacement dan menggunakan mode pengalamatan [BP]. Anda sebenarnya tidak dapat menggunakan mode pengalamatan [BP] tanpa displacement dalam bahasa mesin. Assembler sangat menjaga hal ini dengan menggunakan displacement 8-bit (MOD = 01) dari 00H ketika [BP] mode pengalamatan digunakan dalam suatu instruksi. Ini artinya mode pengalamatan [BP] terakit sebagai [BP+0], meskipun [BP] digunakan dalam instruksi. Mode pengalamatan yang khusus juga dapat digunakan dalam mode 32-bit.

Mode pengalamatan 32-bit dalam 80386 sampai versi di atasnya diperoleh dengan menjaiankan mesin ini dalam mode instruksi 32-bit atau dalam mode insiruksi 16-bit dengan mempergunakan prefiks ukuran-alamat 67H. Tabel berikut memperlihatkan pengkodean untuk R/M digunakan untuk mode pengalamatan 32 bit secara khusus. Perhatikan bahwa

pada saat R/M = 100, maka byte tambahan akan muncul dalam instruksi dan disebut byte indeks-berskala. Byte indeks-berskala menandakan bahwa bentuk pengalamatan indeks-berskala tidak ada dalam Tabel berikut :

Mode pengalamatan 32 bit yang dipilih R/M

Kode R/M	Mode Pengalamatan
000	DS:[EAX]
001	DS:[ECX]
010	DS:[EDX]
011	DS:[EBX]
100	Memakai byte indeks-berskala
101	SS:[EBP]
110	DS:[ESI]
111	DS:[EDI]

Jika kode R/M = 100, maka Byte indeks-skala adalah :

s	**s**	**Indeks**	**Base**

Keterangan :

s	**s**	
0	0	$\times 1$
0	1	$\times 2$
1	0	$\times 4$
1	1	$\times 8$

Byte indeks-berskala terutama digunakan ketika dua register ditambahkan ke dalam alamat memori dalam suatu instruksi. Karena byte indeks-berskala ditambahkan dalam instruksi, maka ada tujuh bit dalam opcode yang dibuat dan delapan bit dalam byte indeks-berskala. Ini artinya instruksi indeks-berskala mempunyai 2^{15} (32K) kemungkinan kombinasi. Ada lebih dari 32.000 variasi yang berbeda dari

instruksi MOV sendiri dalam mikroprosesor 80386 sampai Pentium 4.

Untuk field indeks dan basis, keduanya berisi nomor register seperti terlihat pada tabel berikut : (32 bit)

Kode	W=0 (byte)	W=1 (Word)	W=1 (DoubleWord)
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

Kasus : konversikan instruksi MOV [1000H], DL ke bahasa mesin ?

Solusi :

1	0	0	0	1	0	**0**	**0**
MOD			**REG**			**R/M**	
0	0	0	1	0	1	1	0

Displacement rendah

0	0	0	0	0	0	0	0

Displacement tinggi

0	0	0	1	0	0	0	0

Keterangan :
D = 0, transfer dari register
W = 0, byte
R/M = SS : [BP]
MOD : karena R/M sama dengan BP (pengalamatan khusus)
Displacement = 1000H
Reg = DL

Kasus : konversikan instruksi MOV [BP], DL ke bahasa mesin ?
Solusi :

1	0	0	0	1	0	**0**	**0**
MOD			**REG**			**R/M**	
0	1	0	1	0	1	1	0

Displacement 8 bit

0	0	0	0	0	0	0	0

Keterangan :
D = transfer dari register
W = byte
R/M = SS : [BP]
MOD : karena R/M sama dengan BP (pengalamatan khusus)
Displacement = 00H
Reg = DL

Instruksi MOV EAX,[EBX+4*ECXJ dikodekan sebagai 67668B048BH. Perhatikan bahwa ukuran alamat (67H) dan ukuran register (66H) muncul pada instruksi. Kode ini (67668B048BH) digunakan ketika mikroprosesor 80386 hingga versi di atasnya beroperasi dalam mode instruksi 16-bit untuk instruksi ini. Apabila mikroprosesor beroperasi dalam mode instruksi 32-bit, kedua prefiks akan hilang dan instruksi akan menjadi instruksi 8B048BH. Penggunaan prefiks tergantung pada mode operasi dari mikroprosesor. Pengalamatan indeks-berskala dapat juga menggunakan register tunggal dikalikan dengan faktor skala. Sebagai contoh adalah instruksi MOV AL, [2*ECXJ. Isi dari lokasi segmen data dialamatkan oleh dua kali ECX yang disalin ke dalam AL.

Instruksi Segera. Diasumsikan bahwa instruksi MOV WORD PTR [BX+1000H], 1234H dipilih sebagai contoh instruksi 16-bit menggunakan pengalamatan *segera.* Instruksi ini memindahkan 1234H ke lokasi memori berukuran-word yang dialamatkan oleh penjumlahan dari 1000H, BX dan DS x

I0H. Instruksi enam-byte menggunakan 2 byte untuk field opcode, W, MOD dan R/M. Dua dari enam byte adalah data 1234H. Dua dari enam byte adalah displacement dari I000H.

Instruksi ⇒ MOV WORD PTR [BX+1000H],1234H

1	1	0	0	0	1	**1**	**1**
MOD			**REG**			**R/M**	
1	0	0	0	0	1	1	1

Displacement rendah

0	0	0	0	0	0	0	0

Displacement tinggi

0	0	0	1	0	0	0	0

Data rendah

0	0	1	1	0	1	0	0

Data tinggi

0	0	0	1	0	0	1	0

Instruksi MOV segmen.

Kode REG segmen yang digunakan adalah :

Kode	**Register Segmen**
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS

Kasus : MOV BX, CS

Solusi :

1	0	0	0	1	1	**0**	**0**
MOD			**REG**			**R/M**	
1	1	0	0	1	0	1	1

BAB IV
MULAI DENGAN BAHASA RAKITAN (ASSEMBLY)

Format struktur untuk pemrograman satu segmen (program COM) menggunakan bahasa Assembly adalah :

```
                .Model Small
                .Code
                Org 100h
Label :         Listing Program

                Int 20h
End             Label
```

Keterangan :

.Model Small	Data dan code yang digunakan oleh program kurang dari ukuran 1 segment atau 64 KB.
.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.
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.

Label:	**Listing Program**
	Int 20h Interupsi 20h berfungsi untuk mengakhiri program dan menyerahkan kendali sepenuhnya kepada DOS.
End Label	

A. Program Pertama

Program pertama kita adalah program untuk menampilkan karakter. Buka notepad yang telah tersedia pada sistem operasi Windows, tuliskan listing berikut :

```
        .MODEL SMALL
        .CODE
        ORG 100h
mulai :
        MOV AH,02h;
        MOV DL,'A';
        INT 21h;
        INT 20h;
END mulai
```

Lalu simpan listing di atas dengan nama file program1.asm, dan pastikan file tersimpan pada direktori di mana file Tasm.EXE dan Tlink.Exe berada.

Untuk kompilasi dan eksekusi, dapat kita lakukan beberapa langkah berikut :

```
C:\Tasm program1.asm
Turbo Assembler Version 4.1 Copyright (c) 1988,
1996 Borland International

Assembling file: program1.asm
Error messages: None
Warning messages: None
Passes: 1
```

Remaining memory: 453k

C:\Tlink program1.obj (menghasilkan file berekstensi EXE) atau
C:\Tlink/t program1.obj (menghasilkan file berekstensi COM)
Turbo Link Version 7.1.30.1 Copyright (c) 1987,
1996 Borland International
Warning : No stack

C:\program1
Pada layar akan muncul karakter “A”.
Keterangan :

Untuk memunculkan karakter, kita gunakan interupsi 21H dengan nilai AH=2, dan DL berisi karakter yang akan ditampilkan. Interupsi 20H berguna untuk berhenti dari program dan kembali ke mode DOS.

B. Berbagai Kasus

Program selanjutnya adalah program menampilkan karakter dengan atributnya. Untuk itu kita gunakan interupsi 10H, dengan aturan :
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
Tulis listing berikut pada notepad :

```
        .MODEL SMALL
        .CODE
        ORG 100h
mulai :
        MOV AH,09h
        MOV AL,'A'
        MOV BH,00h
```

```
        MOV BL,93h
        MOV CX,03h
        INT 10h
        INT 20h
        END mulai
```

Setelah dikompilasi dan dieksekusi, maka pada layar akan muncul tiga buah karakter “AAA” dengan background berwarna biru.

Program selanjutnya adalah program menampilkan beberapa karakter terurut ABCDEFGHIJKLMNOPQRSTUVWXYZ. Untuk itu kita perlukan proses looping. Proses looping memiliki bentuk berikut :

```
        MOV CX, banyak pengulangan
Ulang : INT nh; instruksi interupsi yang diulang
        Loop Ulang
```

Untuk program menampilkan beberapa karakter, dapat kita ikuti listing berikut :

```
        .MODEL SMALL
        .CODE
        ORG 100h
mulai :
        MOV AH,02h
        MOV DL,'A'
        MOV CX,1Ah
Ulang :
        INT 21h
        INC DL
        LOOP Ulang

        INT 20h
END mulai
```

Bila program di atas kita modifikasikan agar menghasilkan urutan karakter terbalik dari Z ke A, maka listing yang kita tuliskan adalah :

```
        .MODEL SMALL
        .CODE
        ORG 100h
mulai :
        MOV AH,02h
        MOV DL,'Z'
        MOV CX,1Ah
Ulang :
        INT 21h
        DEC DL
        LOOP Ulang

        INT 20h
END mulai
```

C. Operasi Dasar Aritmatika

1. Penambahan

   Untuk operasi penambahan dapat kita gunakan mnemonic berikut :

   ADD destination, source

   Instruksi ini setara dengan destination=destination +source.

   Contoh program yang dapat kita buat adalah :

```
        .MODEL SMALL
        .CODE
        ORG 100h
mulai :
        MOV AL,30
        MOV BL,20
        ADD AL,BL
        INT 20h
```

```
END mulai
```

Nilai AL setelah penambahan adalah 50h(AL+BL). Jika kita menggunakan Debug untuk fasilitas tempat kita menuliskan program di atas, maka nilai 30 secara otomatis akan dianggap 30h, dan begitu juga nilai 20 akan diasumsikan 20h.

Instruksi penambahan lainnya adalah :

ADC destination, source

ADC adalah penambahan dengan carry, perhatikan kasus berikut :

```
MOV AL,FF
MOV BL,09
MOV DL,09
ADD BL,AL
ADC DL,AL
INT 20h
```

Nilai BX setelah ADD BL,AL, adalah 08, dan nilai DX setelah ADC DL,AL adalah 09 (DL=DL+AL+carry).

AX	BX	DX	Carry	Keterangan
00FF	0000	0000		MOV AL,FF
00FF	0009	0000		MOV BL,09
00FF	0009	0000		MOV DL,09
00FF	0008	0009	1	ADD BL,AL
00FF	0008	0009		ADC DL,AL

Tabel di atas hasil eksperimen penulis menggunakan Debug.

2. Pengurangan

Instruksi yang digunakan untuk operasi pengurangan adalah :

SUB destination, source

Instruksi ini setara dengan destination=destination - source.

Contoh program yang dapat kita buat adalah :

```
        .MODEL SMALL
        .CODE
        ORG 100h
mulai :
        MOV AL,30
        MOV BL,20
        SUB AL,BL
        INT 20h
        END mulai
```

Nilai AL setelah penambahan adalah 10h(AL-BL).

Instruksi lainnya yang digunakan untuk pengurangan adalah :

SBB destination, source

Instruksi ini setara dengan destination=destination – source - borrow.

Contoh :

```
MOV AL,10
MOV BL,20
MOV DL,30
SUB AL,BL
SBB BL,DL
INT 20h
```

Hasil yang diperoleh adalah :

AX	BX	DX	Borrow	Keterangan
0010	0000	0000		MOV AL,10
0010	0020	0000		MOV BL,20
0010	0020	0030		MOV DL,30
00F0	0020	0030	1	SUB AL,BL
00F0	00EF	0030		SBB BL,DL

3. Perkalian

   Bentuk instruksi,

   MUL source

   Contoh program (diambil dari bukunya Susanto, Belajar Sendiri Pemrograman Dengan Bahasa Assembly):

```
        .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 1EF*2FE
        MOV HslLo,AX
        MOV HslHi,DX
        INT 20h ; Kembali ke DOS
END TData
```

4. Pembagian

   Bentuk instruksi

   DIV source

   Contoh program,

```
MOV BX,2
MOV AX,1EF
DIV BX
INT 20h
```

   Setelah program ini dieksekusikan maka,

AX	BX	DX	Keterangan
0000	0002	0000	MOV BX,2
01EF	0002	0000	MOV AX,1EF
00F7	0002	0001	DIV BX

BAB V
MANIPULASI BIT DAN OPERASI LOGIKA

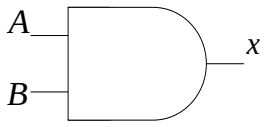
A. Gerbang Logika

Informasi biner direpresentasikan di dalam komputer digital oleh beberapa kuantitas yang disebut juga sebagai sinyal. Sinyal-sinyal elektris yang hadir di dalam komputer seperti voltase, direpresentasikan ke dalam salah satu bentuk satu atau dua state yang dikenal. Untuk representasi ke dalam dua state, variabel biner dapat berupa nilai 1 atau 0. Pada beberapa komputer digital, sinyal 3 volt merepresentasikan nilai biner 1 dan sinyal 0,5 volt merepresentasikan nilai biner 0.

Logika biner bertransaksi dengan berbagai variabel biner dan operasi-operasi yang mengandung berbagai logika, yang selanjutnya digunakan untuk memaparkan, dalam bentuk tabular atau aljabar, pemanipulasian dan pengolahan informasi biner. Pemanipulasian informasi biner dilakukan oleh berbagai sirkuit logika yang disebut juga sebagai gate. Gate (gerbang) merupakan sebuah blok hardware yang memproduksi sinyal-sinyal biner 1 dan 0 ketika input logikal yang diperlukan telah terpenuhi. Masing-masing gate memiliki simbol grafis yang berbeda dan berbagai operasi yang dilakukan berdasarkan maksud-maksud dari ekspresi aljabar. Hubungan input-output dari berbagai variabel biner untuk masing-masing gate direpresentasikan dalam bentuk tabular, sebagai berikut :

1. Gerbang AND

   Simbol :



   Fungsi aljabar :

   $x = A \cdot B$ atau $x = AB$

Tabel kebenaran :

A	B	x
0	0	0
0	1	0
1	0	0
1	1	1

Bentuk instruksi :

AND destination, source

Contoh :

```
MOV AL,FF
MOV BL,AB
AND AL,BL
INT 20h
```

Keadaan register setelah program dieksekusi :

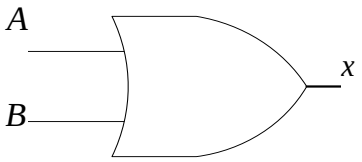
AX	**BX**	**DX**	**Keterangan**
00FF	0000	0000	MOV AL,FF
00FF	00AB	0000	MOV BL,AB
00AB	00AB	0000	AND AL,BL

Penjelasan :

FF = 1111 1111
AB = 1010 1011
FF AND AB = 1010 1011 = AB

2. Gerbang OR

Simbol :



Fungsi aljabar :

$x = A + B$

Tabel kebenaran :

A	B	x
0	0	0
0	1	1
1	0	1
1	1	1

Bentuk instruksi :

OR destination, source

Contoh :

```
MOV AL,FF
MOV BL,AB
OR AL,BL
INT 20h
```

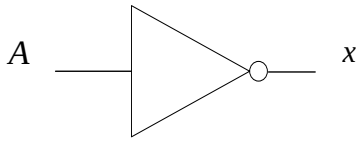
Keadaan register setelah program dieksekusi :

AX	BX	DX	Keterangan
00FF	0000	0000	MOV AL,FF
00FF	00AB	0000	MOV BL,AB
00FF	00AB	0000	OR AL,BL

Penjelasan :

FF = 1111 1111
AB = 1010 1011
FF OR AB = 1111 1111 = FF

3. INVERTER
   Simbol :



Fungsi aljabar :

$x = A'$

Tabel kebenaran :

A	x
0	1
1	0

Bentuk instruksi :

NOT operand

Contoh :

```
MOV BL,FA
NOT BL
INT 20h
```

Keadaan register setelah program dieksekusi :

AX	**BX**	**DX**	**Keterangan**
0000	00FA	0000	MOV BL,FA
0000	0005	0000	NOT BL

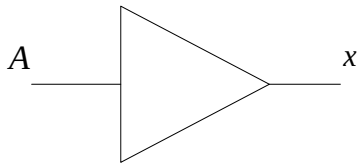
Penjelasan :

FA = 1111 1010

NOT FA = 0000 0101 = 5

4. BUFFER

   Simbol :



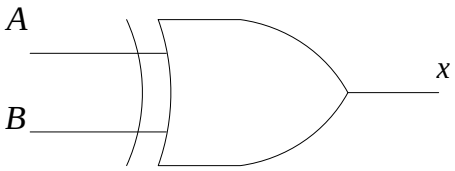
Fungsi aljabar :

$x = A$

Tabel kebenaran :

A	x
0	0
1	1

5. Gerbang XOR
   Simbol :



   Fungsi aljabar :
   $x = A \oplus B$ atau $x = A'B + AB'$

Tabel kebenaran :

A	B	x
0	0	0
0	1	1
1	0	1
1	1	0

   Bentuk instruksi :

XOR destination, source

   Contoh :

```
MOV AL,FF
MOV BL,AB
XOR AL,BL
INT 20h
```

   Keadaan register setelah program dieksekusi :

AX	BX	DX	Keterangan
00FF	0000	0000	MOV AL,FF
00FF	00AB	0000	MOV BL,AB
0056	00AB	0000	XOR AL,BL

   Penjelasan :

   FF = 1111 1111
   AB = 1010 1011
   FF OR AB = 0101 0100 = 54

B. Pergeseran Bit

1. SHL (Shift Left)

   Bentuk instruksi :

   SHL operand1, operand2

   Tujuan : untuk menggeser operand1 sebanyak operand2 ke kiri secara per bit.

   Contoh :

```
MOV AX,FAD
MOV CL,3
SHL AX,CL
```

   Keadaan register setelah program dieksekusi adalah :

AX	BX	DX	Keterangan
0FAD	0000	0000	MOV AX,FAD
0FAD	0000	0000	MOV CL,3
7D68	0000	0000	SHL AX,CL

   Keterangan :

0				F				A				D			
0	0	0	0	1	1	1	1	1	0	1	0	1	1	0	1

   Geser 3 bit ke kiri akan menghasilkan :

7				D				6				8			
0	1	1	1	1	1	0	1	0	1	1	0	1	0	0	0

2. SHR (Shift Right)

   Bentuk instruksi :

   SHR operand1, operand2

   Tujuan : untuk menggeser operand1 sebanyak operand2 ke kanan secara per bit.

   Contoh :

```
MOV AX,FAD
MOV CL,3
SHR AX,CL
```

Keadaan register setelah program dieksekusi adalah :

AX	BX	DX	Keterangan
0FAD	0000	0000	MOV AX,FAD
0FAD	0000	0000	MOV CL,3
	0000	0000	SHR AX,CL

Keterangan :

0				F				A				D			
0	0	0	0	1	1	1	1	1	0	1	0	1	1	0	1

Geser 3 bit ke kanan akan menghasilkan :

0				1				F				5			
0	0	0	0	0	0	0	1	1	1	1	1	0	1	0	1

BAB VI
MODE PENGALAMATAN DATA MENGGUNAKAN DEBUG & TURBO ASSEMBLER

A. Debug

Kata bug yang diucapkannya sebagai BAG-an, dapat mempunyai beberapa arti, yang umum adalah berati kutu, dalam bahasa Slang Amerika berarti mengganggu. Dalam dunia komputer dan elektronika dimaksudkan suatu kesalahan di dalam hardware maupun software, tetapi yang lebih lazim adalah dalam software.

Konon kata bug telah digunakan pada zaman permulaan perusahaan telepon dan elektronik serara senda-gurau bahwa dalam instalasi terjadi kerusakan karena kabel-kabelnya dimakan kutu, dan pada masa itu, belum ditemukan isolusi kabel yang tidai disukai oleh kutu. Tetapi menurut catatan buku yang ditulis oleh Grace Hopper yang terdapat di Naval Museum di kota Dahlgren negara bagian Virginia. Pada waktu Hopper bekerja dengan komputer Mark II, komputernya berhenti tidak jalan dan ditemukan sebuah kutu terhimpit dalam salah satu relay (semacam kontak). Diambil bangkai kutu tersebut dan ditempelkan pada buku catatan dengan pita tape. kemudian dibubuhkan catatan :

"First actual case of bug being found."

Lepas dari cerita tersebut, suatu kesalahan di dalam program dinamakan *bug*, sedangkan mencari dan menghilangkan kesalahan di dalam program dinamakan *debugging*.

Debug di sini, adalah utilitas bantu yang kita manfaatkan untuk penulisan bahasa Assembly dan pemrosesan penelusuran berbagai keadaan program yang telah kita buat.

Berbagai instruksi yang hadir di dalam operasional menggunakan Debug :

1.	H (Hexa)	Melaksanakan penjumlahan dan pengurangan dua buah bilangan hexa sekaligus. - H 1976 0ABC 2432 0EBA
2.	R (Register)	Mengetahui isi dari register pada saat dieksekusi perintah bersangkutan.
3.	A (Assembler)	Untuk melakukan penulisan program Assembler pada debug. - A100 0B0B:0100
4.	N (Name)	Instruksi penamaan program yang telah dibuat. Bentuk perintah : - N Fadli.com
5.	RCX	Perintah untuk memperbaharui isi dari register CX. CX merupakan tempat penampungan panjang program yang sedang aktif sebelum dijalankan.
6.	RIP	Perintah pemberitahuan kepada komputer untuk mulai memproses program dari nilai alamat tertentu.
7.	W (Write)	Perintah untuk menulis program ke dalam file. Sebelum ini dilakukan, RCX dan RIP telah diatur terlebih dahulu. - W
8.	G (Go)	Eksekusi program.
9.	T (Trace)	Perintah penjelajahan program baris per baris.
10.	U (Unassemble)	Melihat listing program yang sedang aktif.
11.	Q (Quit)	Keluar dari Debug
12	D (Data)	Melihat bagaimana data disimpan
13	E (Edit)	Ex : E 0B62:0100

Langkah-langkah pembuatan program menggunakan fasilitas Debug.
Pada prompt,
C:\debug

```
-       A100
0B0B : 0100   MOV AH,02
0B0B : 0102   MOV DL,41
0B0B : 0104   INT 21
0B0B : 0106   INT 20
0B0B : 0108   ↵ (enter)
-       RCX
CX 0000
: 8
-       RIP
IP 0100
:100
-       NHURUFA.COM
-       W
Writing 00008 bytes
-       G
A
Program terminated normally
```

B. Mode Pengalamatan Data

1. Pengalamatan register

Contoh :

Instruksi	Size	Operasi
MOV AL, BL	8-bit	Pindahkan isi BL ke AL
MOV CH, CL	8-bit	Pindahkan isi CL ke CH
MOV AX, CX	16-bit	Pindahkan isi CX ke AX
MOV ECX, EBX	32-bit	Pindahkan isi EBX ke ECX
MOV ES, DS	-	Tidak diijinkan (segmen ke segmen)
MOV BL, DX	-	Tidak diijinkan (beda ukuran)
MOV CS, AX	-	Tidak diijinkan (register segmen tidak boleh menjadi register tujuan)

Program :

```
MOV AL,BL
MOV DL,BL
INT 20h
```

Lakukan tracing menggunakan debug, maka nilai akhir yang diperoleh : AX=0000, BX=0000, DX=0000.

Andaikan instruksi di atas kita ubah sedikit menjadi :

```
MOV AL,CL
MOV BL,CL
INT 20h
```

Maka nilai akhirnya akan berupa : AX=BX=CX≠0, kenapa ? (kasus ditinggalkan untuk pembaca).

2. Pengalamatan segera

Contoh :

Instruksi	Size	Operasi
MOV AL, 44	8-bit	AL = 44 desimal (dalam utility Debug dianggap 100 hexadesimal)
MOV CH, 44H	8-bit	CH = 44 hexadesimal
MOV SI, 0	16-bit	SI = 0 hexadesimal
MOV CH, 100	32-bit	CH = 100 desimal (dalam utility Debug dianggap 100 hexadesimal, jika instruksi tersebut berbentuk MOV CX, 100, tetapi jika kita memaksa menulis instruksi MOV CH, 100 tersebut juga ke dalam Debug, maka pesan ^error akan muncul)
MOV AL, 'A'	8-bit	AL = 'A' (tidak berlaku dalam utility Debug)
MOV AX, 'AB'	16-bit	AX = 'BA' (tidak berlaku dalam utility Debug)
MOV CL, 11001110B	8-bit	CL = 11001110 biner (tidak berlaku dalam utility Debug)
MOV EAX, 100Y	32-bit	EAX = 100 biner (tidak berlaku dalam utility Debug)

Program :

```
MOV AL, 10
MOV DL, AL
INT 20h
```

Lakukan tracing menggunakan debug, maka nilai akhir yang diperoleh : AX=0010, DX=0010.

3. Pengalamatan data langsung

Contoh :

Instruksi	Size	Operasi
MOV AL, NUMBER	8-bit	Menyalin isi byte lokasi memori segmen data NUMBER ke dalam AL
MOV NUMBER, AL	8-bit	Menyalin AL ke dalam lokasi memori segmen data NUMBER
MOV ES:[2000 h], AL	8-bit	Menyalin AL ke dalam lokasi memori segmen data tambahan 2000h
MOV AL, DS:[1234h]	8-bit	Menyalin isi byte pada lokasi 1234h ke dalam AL

Program : (dibuat menggunakan Turbo Assembler)

```
        .MODEL SMALL
        .CODE
        ORG 100h
DATA :
        JMP proses
        F DB 10h
proses :
        MOV AL, F
        MOV DS:[1234h], AL
        MOV BL, DS:[1234h]
        INT 20h
END DATA
```

Lakukan tracing menggunakan Debug, maka nilai akhir yang diperoleh : AX =0010, BX=0010.

Tugas : pada program di atas berapa nilai yang dikandung oleh register CX ?.

Keterangan :

Baris F DB 10h bermaksud bahwa tempatkan nilai 10 hexa ke dalam variabel F.

Tipe-tipe data yang lain yang digunakan di dalam Assembler adalah :

→ **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

Contoh pemakaian :

```
        .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
```

Tugas :

Bandingkan kedua program di bawah ini :

```
        .MODEL SMALL
        .CODE
        ORG 100h
  DATA :
        JMP proses
        F DB 10h
  proses :
        MOV AL, F
        MOV DS:[1234h], AL
        MOV AL, DS:[1234h]
        INT 20h
  END DATA
```

dan

```
        .MODEL SMALL
        .CODE
        ORG 100h
  DATA :
        JMP proses
        F DB 10h
  proses :
        MOV AL, F
        MOV DS:[1234h], AL
        MOV CL, DS:[1234h]
        INT 20h
  END DATA
```

Lakukan analisis langkah per langkah kedua program di atas.

4. Pengalamatan register tidak langsung

Contoh :

Instruksi	Size	Operasi
MOV CX, [BX]	16-bit	Menyalin isi word lokasi memori segmen data yang dialamatkan BX ke dalam CX
MOV [BP], DL	8-bit	Menyalin DL ke dalam lokasi segmen stack yang dialamatikan BP
MOV [DI], BH	8-bit	Menyalin BH ke dalam lokasi segmen memori yang dialamatkan DI
MOV [DI], [BX]	-	Tidak diijinkan, penyalinan memori ke memori, kecuali dengan instruksi string.
MOV ECX,[EBX]	32 bit	Menyalin is doubleword dari lokasi memori segmen data yang dialamatkan oleh EBX ke dalam ECX

Dalam beberapa kasus, pengalamatan tidak langsung memerlukan penspesifikasian ukuran data yang diatur dengan direktif assembler khusus BYTE PTR, WORD PTR, atau DWORD PTR.

Urgensinya :

MOV AL, [DI] secara jelas merupakan instruksi perpindahan data berukuran byte.

MOV [DI], 10h, instruksi ini sangat membingungkan, apakah data yang dialamatkan berupa byte, word, doubleword.

Kita ubah MOV [DI], 10h menjadi berbagai alternatif berikut :

MOV WORD PTR [DI], 10h, atau MOV BYTE PTR [DI], 10h, atau juga MOV DWORD PTR [DI], 10h.

Program pendukung materi :

```
        .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
        ADD BX,2 ;
        LOOP Ulang ;

        INT 20h
END TData
```

Keterangan :

Bentuk LEA Reg, Data difungsikan untuk mengambil suatu alamat efektif (Load Effectif Address). Untuk mengakses data yang ditunjukkan oleh register Reg, setelah didapatkannya alamat efektive harus digunakan tanda kurung siku ('[ ]'), contoh : MOV DL, [BX]. 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.

5. Pengalamatan base-plus-index

Contoh :

Instruksi	Size	Operasi
MOV CX, [BX+DI]	16-bit	Menyalin isi word lokasi memori segmen data yang dialamatkan BX plus DI ke dalam CX
MOV CH, [BP+SI]	8-bit	Menyalin isi byte lokasi memori segmen data yang dialamatkan BP plus SI ke dalam CH
MOV [BX+SI], SP	16-bit	Menyalin SP ke dalam lokasi memori segmen data yang dialamatkan oleh BX plus SI
MOV CL, [EDX+EDI]	8-bit	Menyalin isi byte lokasi memori segmen data yang dialamatkan EDX plus EDI ke dalam CL
MOV [EAX+EBX],ECX	32-bit	Menyalin ECX ke dalam lokasi memori segmen data yang dialamatkan oleh EAX plus EBX

Program pendukung materi :

```
        .MODEL SMALL
        .CODE
        ORG 100h
TData : JMP Proses
        ARRAY DB 16 DUP (?)

Proses:
        MOV BX, OFFSET ARRAY
        MOV DI, 10h
        MOV AL, [BX+DI]
        MOV DI, 20h
        MOV [BX+DI], AL
        INT 20h
END TData
```

Setelah program dieksekusi, maka nilai untuk berbagai register adalah sebagai berikut :

AX	BX	CX	DX	SP	BP	SI	DI	DS	ES	SS	CS	IP
00	010	002		FF			002	0B	0B6	0B	0B6	01

BB	3	2		FE			0	62	2	62	2	20
NV	**UP**	**EI**	**PL**	**NZ**	**NA**	**PO**	**NC**					

6. Pengalamatan register relatif
   Contoh :

Instruksi	Size	Operasi
MOV AX, [DI+100h]	16-bit	Menyalin isi word lokasi memori segmen data yang dialamatkan DI plus 100h ke dalam AX
MOV ARRAY[SI], BL	8-bit	Menyalin BL ke dalam lokasi memori segmen data yang dialamatkan oleh ARRAY plus SI
MOV LIST[SI+2], CL	8-bit	Menyalin CL ke dalam lokasi memori segmen data yang dialamatkan oleh LIST +SI+2
MOV DI,SET_TI[BX]	16-bit	Menyalin isi word lokasi memori segmen data yang dialamatkan SET_TI plus BX ke dalam DI

Program pendukung materi :

```
        .MODEL SMALL
        .CODE
        ORG 100h
TData : JMP Proses
        ARRAY DB 16 DUP (?)
              DB 29h
              DB 30 DUP (?)
Proses:
        MOV DI, 10h
        MOV AL, ARRAY[DI]
        MOV DI,20H
        MOV ARRAY[DI], AL
        INT 20h
END TData
```

Hasil eksekusi (keadaan dari nilai berbagai register) program ditinggal untuk tugas pembaca.

7. Pengalamatan base relatif-plus-index
   Contoh :

Instruksi	Size	Operasi
MOV DH, [BX+DI+20h]	8-bit	Menyalin isi byte lokasi memori segmen data yang dialamatkan oleh jumlah BX, DI dan 20h ke dalam DH
MOV EAX, FILE[EBX+ECX+2]	32-bit	Menyalin isi doubleword lokasi memori segmen data yang dialamatkan oleh jumlah FILE, EBX, ECX, dan 2 ke dalam EAX

Program pendukung materi :

```
        .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 ;
END TData
```

Analisis hasil :

```
C:\TASM\BIN>DEBUG MP.COM
-d
0B62:0100 EB 47 90 00 00 00 00 00-00 00 00 00 00 00
00 00
0B62:0110 00 00 00 00 00 00 00 00-00 00 00 00 00 00
00 00
0B62:0120 00 00 00 00 00 00 00 00-00 00 00 00 00 00
00 00
0B62:0130 00 00 00 00 00 00 00 00-00 00 00 00 00 00
00 00
0B62:0140 00 00 00 00 00 00 00 00-00 BB 03 01 83 C3
15 33
0B62:0150 F6 C7 00 99 00 C6 40 02-10 C6 40 04 78 C6
40 05
0B62:0160 99 C6 40 06 50 C6 40 07-83 CD 20 8B 46 FE
89 46
0B62:0170 F8 C7 46 F6 00 00 C4 5E-F6 26 C7 47 02 00
00 26
-u
0B62:0100 EB47      JMP 0149
0B62:0102 90        NOP
0B62:0103 0000      ADD [BX+SI],AL
0B62:0105 0000      ADD [BX+SI],AL
0B62:0107 0000      ADD [BX+SI],AL
0B62:0109 0000      ADD [BX+SI],AL
0B62:010B 0000      ADD [BX+SI],AL
0B62:010D 0000      ADD [BX+SI],AL
0B62:010F 0000      ADD [BX+SI],AL
0B62:0111 0000      ADD [BX+SI],AL
0B62:0113 0000      ADD [BX+SI],AL
```

```
0B62:0115 0000      ADD [BX+SI],AL
0B62:0117 0000      ADD [BX+SI],AL
0B62:0119 0000      ADD [BX+SI],AL
0B62:011B 0000      ADD [BX+SI],AL
0B62:011D 0000      ADD [BX+SI],AL
0B62:011F 0000      ADD [BX+SI],AL
-u
0B62:0141 0000      ADD [BX+SI],AL
0B62:0143 0000      ADD [BX+SI],AL
0B62:0145 0000      ADD [BX+SI],AL
0B62:0147 0000      ADD [BX+SI],AL
0B62:0149 BB0301   MOV BX,0103
0B62:014C 83C315   ADD BX,+15
0B62:014F 33F6      XOR SI,SI
0B62:0151 C7009900 MOV WORD PTR [BX+SI],0099
0B62:0155 C6400210 MOV BYTE PTR [BX+SI+02],10
0B62:0159 C6400478 MOV BYTE PTR [BX+SI+04],78
0B62:015D C6400599 MOV BYTE PTR [BX+SI+05],99
-u
0B62:0161 C6400650        MOV        BYTE        PTR
[BX+SI+06],50
0B62:0165 C6400783        MOV        BYTE        PTR
[BX+SI+07],83
0B62:0169 CD20 INT 20
0B62:016B 8B46FE MOV AX,[BP-02]
0B62:016E 8946F8   MOV [BP-08],AX
0B62:0171 C746F60000 MOV WORD PTR [BP-0A],0000
0B62:0176 C45EF6   LES BX,[BP-0A]
0B62:0179 26 ES:
0B62:017A C747020000 MOV WORD PTR [BX+02],0000
0B62:017F 26        ES:
0B62:0180 C7070000 MOV WORD PTR [BX],0000
-q
```

Hasil tracing

AX=0000 BX=0118 CX=006B DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0B62 ES=0B62 SS=0B62 CS=0B62 IP=0151 NV UP EI PL ZR NA PE NC
0B62:0151 C7009900 MOV WORD PTR [BX+SI],0099 DS:0118=0000
-t
AX=0000 BX=0118 CX=006B DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0B62 ES=0B62 SS=0B62 CS=0B62 IP=0155 NV UP EI PL ZR NA PE NC
0B62:0155 C6400210 MOV BYTE PTR [BX+SI+02],10 DS:011A=00
-t
AX=0000 BX=0118 CX=006B DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0B62 ES=0B62 SS=0B62 CS=0B62 IP=0159 NV UP EI PL ZR NA PE NC
0B62:0159 C6400478 MOV BYTE PTR [BX+SI+04],78 DS:011C=00
-t
AX=0000 BX=0118 CX=006B DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0B62 ES=0B62 SS=0B62 CS=0B62 IP=015D NV UP EI PL ZR NA PE NC
0B62:015D C6400599 MOV BYTE PTR [BX+SI+05],99 DS:011D=00
-t
AX=0000 BX=0118 CX=006B DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0B62 ES=0B62 SS=0B62 CS=0B62 IP=0161 NV UP EI PL ZR NA PE NC

```
0B62:0161 C6400650 MOV BYTE PTR [BX+SI+06],50
DS:011E=00
-t
AX=0000 BX=0118 CX=006B DX=0000 SP=FFFE
BP=0000 SI=0000 DI=0000
DS=0B62 ES=0B62 SS=0B62 CS=0B62 IP=0165 NV UP EI
PL ZR NA PE NC
0B62:0165 C6400783 MOV BYTE PTR [BX+SI+07],83
DS:011F=00
-t
AX=0000 BX=0118 CX=006B DX=0000 SP=FFFE
BP=0000 SI=0000 DI=0000
DS=0B62 ES=0B62 SS=0B62 CS=0B62 IP=0169 NV UP EI
PL ZR NA PE NC
0B62:0169 CD20 INT 20
-q
```

8. Pengalamatan index-berskala

   Contoh :

Instruksi	Size	Operasi
MOV EAX, [EBX+4*ECX]	32-bit	Menyalin isi doubleword lokasi memori segmen data yang dialamatkan oleh jumlah dari 4 kali ECX, ditambah EBX, ke dalam EAX
MOV [EAX+2*EDI+100h], CX	16-bit	Menyalin CX ke dalam lokasi memori segmen data yang dialamatkan oleh jumlah dari EAX, 100h dan 2 kali EDI

   Program pendukung materi :

```
        .MODEL SMALL
        .386
        .CODE
        ORG 100h
TData : JMP Proses
```

```
        LIST DW 0,1,2,3,4
        DW 5,6,7,8,9
Proses:
        MOV EBX, OFFSET LIST
        MOV ECX,2
        MOV AX, [EBX+2*ECX]
        MOV ECX,4
        MOV [EBX+2*ECX], AX
        MOV ECX,7
        MOV [EBX+2*ECX], AX
        INT 20h
END TData
```

Kasus :

Tulis program di bawah ini menggunakan notepad, eksekusikan dan analisis, dan kategorikan program tersebut ke salah satu mode pengalamatan di atas,

```
        .MODEL SMALL
        .CODE
        ORG 100h

TData : JMP Proses
        Kalimat DB 'FADLISYAH' ; 9 karakter
Proses:
        XOR BX,BX ; BX=0 Untuk penunjuk Offset
        MOV CX,9 ; 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
        INT 20h
END TData
```

BAB VII
POP/PUSH

A. POP/PUSH

Instruksi POP dan PUSH adalah sangat penting, karena kedua instruksi ini berfungsi untuk menyimpan dan mengambil data dari memori stack LIFO. Mikroprosesor memiliki enam bentuk instruksi dari instruksi PUSH dan POP yaitu : register, immediate, register segmen, flag, dan semua register. Bentuk POP dan PUSH immediate dan POPA dan PUSHA hanya tersedia pada mikroprosesor 80286 ke atas, hingga Pentium 4.

1. PUSH

   Instruksi PUSH pada 8086-80286 selalu memindahkan dua byte dari data ke stack; pada 80386 ke atas dapat memindahkan hingga 4 byte, tergantung dari register atau ukuran memori yang digunakan.

   Format instruksi :

Simbolik	Example	Ketereangan
PUSH reg16	PUSH BX	Register 16-bit
PUSH reg32	PUSH EDX	Register 32-bit
PUSH mem16	PUSH WORD PTR [BX]	Pointer 16-bit
PUSH mem32	PUSH DWORD PTR [EBX]	Pointer 32-bit
PUSH segmen	PUSH DS	Register segmen
PUSH imm8	PUSH ‘,’	Segera 8-bit
PUSHW imm16	PUSHW 1000h	Segera 16-bit
PUSHD imm32	PUSHD 20	Segera 32-bit
PUSHA	PUSHA	Simpan semua register 16-bit
PUSHAD	PUSHAD	Simpan semua register 32-bit
PUSHF	PUSHF	Simpan flag
PUSHFD	PUSHFD	Simpan EFLAG

2. POP

Instruksi POP adalah kebalikan dari instruksi PUSH. Instruksi POP berfungsi untuk menghapus data dari stack dan menempatkannya pada register 16-bit, register segmen, atau lokasi memori 16-bit.

Format instruksi :

Simbolik	Example	Ketereangan
POP reg16	POP BX	Register 16-bit
POP reg32	POP EDX	Register 32-bit
POP mem16	POP WORD PTR [BX]	Pointer 16-bit
POP mem32	POP DWORD PTR [EBX]	Pointer 32-bit
POP segmen	POP DS	Register segmen
POPA	POPA	POP semua register 16-bit
POPAD	POPAD	POP semua register 32-bit
POPF	POPF	POP flag
POPFD	POPFD	POP EFLAG

Kasus : mencetak kalimat dengan operasi stack

```
        .MODEL SMALL
        .CODE
        ORG 100h

TData : JMP Proses
        Kal DB 'FADLISYAH 132321540 $'
        Ganti DB 13,10,'$'
        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
```

Hasil eksekusi :
FADLISYAH 132321540
FADLISYAH 132321540

BAB VIII
PEMROGRAMAN MODULAR

A. Procedure

Procedure atau subrutin adalah sekelompok instruksi yang biasanya memiliki satu fungsi tertentu.

Format procedure :

Nama_procedure PROC NEAR/FAR

Program

RET

Nama_procedure ENDP

Di belakang kata PROC terdapat 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 instruksi pemanggil 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.

Contoh program : Program menampilkan kalimat pada layar komputer. [1]

```
        .MODEL SMALL
        .CODE
        ORG 100h

TData : JMP Proses
        Kar DB ?
        Klm DB 'UNIVERSITAS GATOT KACA '

Proses : MOV CX,22 ; 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
```

[1] Program adalah modifikasi dari program prosedur yang terdapat pada buku Susanto. 1995. *Belajar Sendiri Pemrograman Dengan Bahasa Assembly*, Elex Media Komputindo.

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

END TData
```

B. Macro

Macro adalah sekumpulan instruksi yang dikemas untuk melakukan satu fungsi atau pekerjaan. Pada dasarnya baik procedure maupun macro adalah sama-sama melaksanakan satu tugas dan dikemas dengan format tersendiri, tetapi dalam hal pemakaian, procedure dilakukan setelah dipanggil oleh instruksi CALL, sedangkan macro disisip dalam tubuh program, seakan-akan berprilaku seperti opcode.

Format macro :

```
NamaMacro   Macro[parameter1, parameter2,.....]

            program

          ENDM
```

Parameter yang mengikuti syntax Macro bersifat optional, artinya bisa hadir dan bisa tidak.

Program :

```
Cetak_Kar MACRO Kar
        MOV CX,3
        MOV AH,02
        MOV DL,Kar
Ulang :
        INT 21h
        LOOP Ulang
        ENDM

        .MODEL SMALL
```

```
        .CODE
        ORG 100h
Proses:
        Cetak_Kar 'F'
        INT 20h
        END Proses
```

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 direktif LOCAL assembler akan membedakan label tersebut setiap kali terjadi pemanggilan terhadapnya.

Program menggunakan direktif lokal

```
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

        .MODEL SMALL
        .CODE
        ORG 100h
Proses:
        Cetak_Kar 'F'
        Cetak_Kar 'A'
```

```
        INT 20h
        END Proses
```

C. Berbagai Program Untuk Latihan

Program menghapus layar dengan merontokkan berbagai karakter pada layar satu per satu.

```
Delay MACRO
        PUSH CX
        XOR CX,CX
Loop1:
        LOOP Loop1
        POP CX
        ENDM
Geser MACRO PosY
        PUSH AX
        PUSH BX
        PUSH CX
        XOR CX,CX
        MOV AL,26
        SUB AL,PosY
        MOV CL,AL
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
        POP CX
        POP BX
        POP AX
ENDM

        .MODEL SMALL
```

```
        .CODE
        ORG 100h

TData : JMP Proses
        PosY DB ?
Proses:
        MOV AX,0B800h
        MOV ES,AX
        MOV BX,3998
        MOV CX,25
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
```

Program untuk menampilkan bilangan prima dari 1 hingga 1000.

```
Cetak_Klm MACRO Klm
        MOV AH,09
        LEA DX,Klm
        INT 21h
        ENDM

CDesimal MACRO Angka
```

```
        LOCAL Ulang, Cetak
        MOV AX,Angka
        MOV BX,10
        XOR CX,CX
Ulang :
        XOR DX,DX
        DIV BX
        PUSH DX
        INC CX
        CMP AX,0
        JNE Ulang
Cetak :
        POP DX
        ADD DL,'0'
        MOV AH,02
        INT 21h
        LOOP Cetak
        ENDM

        .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
```

Hasil eksekusi adalah :
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

Program menu sorot.

```
  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
```

```
        MOVAL,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

        .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
```

DAFTAR PUSTAKA

Brey, Barry B. 2000. *Mikroprosesor Intel 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, dan Pentium-Pro : Arsitektur, Pemrograman, Antarmuka (Edisi kelima jilid 1)*, Penerbit Erlangga.

Brey, Barry B. 2003. *The Intel Mikroprosesor 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium-Pro Prosessor, Pentium II, Pentium III, Pentium 4 : Architecture, Programming, and Interfacing (6thEdition)*, Prentice Hall.

I. Siauw, Soen. 1987. *Belajar Sendiri Personal Computer*, Penerbit PT. Elex Media Komputindo – Kelompok Gramedia, Jakarta.

Lukito, Ediman. 1990. *Dasar-Dasar Pemrograman dengan Assembler 8088*, Elex Media Komputindo.

Mano, M. Morris. 1993. *Computer System Architecture*, Prentice-Hall International.

Stalling, William. 1998. *Organisasi dan Arsitektur Komputer : Perancangan Kinerja (jilid 1)*, Prentice Hall yang diterjemahkan oleh PT Prenhallindo, Jakarta.

Susanto. 1995. *Belajar Sendiri Pemrograman Dengan Bahasa Assembly*, Elex Media Komputindo.

BAHASA RAKITAN

Dahlan Abdullah Lahir di Lhokseumawe Provinsi Aceh pada tanggal 28 Februari 1976, SD (Sekolah Dasar) pada tahun 1982 dan selesai pada tahun 1988, melanjutkan pendidikan ke Pasentren Bustanul Ulum yang berada di Desa Alue Pineng – Langsa pada tahun 1988 hingga selesai pada tahun 1991 dengan pendidikan MTSN No. 16 Langsa, kembali ke Lhokseumawe untuk melanjutkan pendidikan pada SMA Negeri Nomor 2 pada tahun 1991 dan selesai pada tahun 1994, kemudian berangkat menuju Kota Yogyakarta untuk melanjutkan Program Pendidikan Strata Satu (S1) di Jurusan Teknik Informatika Universitas Islam Indonesia pada tahun 1994 dan selesai pada tahun 1999 dengan menyandang gelar Sarjana Teknik (S.T) sambil menunggu pekerjaan yang tetap maka saya juga ikut mengajar di Universitas Ahmad Dahlan untuk waktu 1 tahun dan pada tahun 2001 kembali ke Kota Lhokseumawe untuk masuk menjadi Pegawai Negeri Sipil (PNS) sebagai Tenaga Pendidik (Dosen) di Universitas Malikussaleh yang baru saja di negerikan, jabatan pertama yang saya terima sebagai sekretaris LPPM, Ketua PSIK, Kepala UPT Pusat Komputer dan selanjutnya berangkat kuliah pada Program Strata Dua (S2) di Jurusan Teknik Informatika STMIK Eresha pada tahun 2011 dan selesai pada tahun 2014 dengan gelar Magister Komputer (M.Kom), pada saat itu di Universitas Malikussaleh menjabat sebagai Kepala UPT Perpustakaan dan melanjutkan pendidikan ke Program Doktor di Jurusan Ilmu Komputer Universitas Sumatera Utara pada tahun 2014 dan selesai pada tahun 2018 dengan menyandang gelar Doktor (Dr.), aktif dibeberapa organisasi baik yang berskala Nasional atau Internasional, aktif menulis Artikel diberbagai Seminar Nasional atau Internasional dan di Jurnal bereputasi (Scopus/WOS) dan sering memberikan Materi di berbagai Workshop atau Seminar.

SEFA BUMI PERSADA
Jl. Malikussaleh No. 3 Bayu - Aceh Utara
email: sefabumipersada@gmail.com
Telp. 085260363550

ISBN 978-623-6983-04-1