MODUL V

STACK dan PENGENALAN PROCEDURE

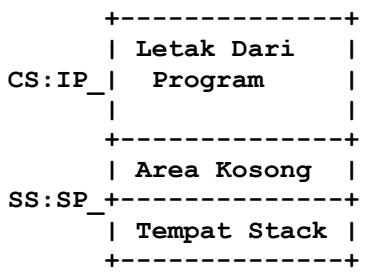
Stack

Stack merupakan bagian memori yang digunakan untuk menyimpan nilai dari suatu register secara sementara. Operasi stack dinamakan juga LIFO (Last In First Out).

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:

```
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
```

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 di bawah ini



PUSH DAN POP

Pengoperasian Stack :

1. PUSH

Untuk menambahkan sebuah *word* yang baru pada stack kita *PUSH* ke stack. Sintaksnya adalah:

PUSH *sumber*

Eksekusi PUSH menyebabkan terjadinya hal berikut:

a. SP (*stack pointer*) dikurangi 2
b. Salinan isi ***sumber*** disalin ke alamat yang ditetapkan oleh SS:SP. ***Sumber*** tidak berubah

2. POP

Untuk mengambil (remove) item pada puncak stack, kita lakukan **POP.** Sintaksnya adalah:

POP *tujuan*

Eksekusi POP menyebabkan terjadinya hal berikut:

1. Isi dari SS:SP (puncak stack) disalin ke ***tujuan***
2. SP ditambah 2.

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. 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
```

Sebagai contoh lainnya pada perintah berikut:

```
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
```

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
```

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.

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.

Procedure

Prosedure adalah bagian perangkat lunak yang dapat dipergunakan berulang kali, mengingat bahwa prosedur disimpan pada memori sekali, namun digunakan seseringkali mungkin. Hal ini tentu saja menghemat ruang memori dan memudahkan pengembangan perangkat lunak. Satu-satunya kerugian dari prosedur ialah bahwa prosedur sedikit menghabiskan waktu sari komputer untuk terhubung dengan prosedur tersebut hingga kemudian selesai menjalankan prosedur tersebut. Instruksi **CALL** ialah instruksi yang melakukan hubungan dengan prosedur dan **RET** (Return) untuk kembali dari prosedur. 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. 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.

Dalam Assembler ada peraturan khusus mengenai penyimpanan prosedur, sebuah prosedur di awali dengan directive PROC dan diakhiri dengan directive ENDP. Dengan menggunakan procedure suatu program yang besar dapat diselesaikan dengan lebih mudah. Proses pencarian kesalahan pun akan lebih mudah bil;a digunakan procedure.

Sintaks deklarasi prosedur adalah:

```
Nama       PROC       tipe

; badan/bagian dari prosedur
RET
Nama       ENDP
```

Tipe operand yang dapat dipilih adalah **NEAR** atau **FAR**. Bentuk NEAR digunakan jika procedure tersebut dipanggil oleh program yang letaknya masih satu segmen dari procedure tersebut. Far digunakan apabila procedure dapat dipanggil dari segmen lain.

MACRO

Macro adalah lebih mudah dibuat daripada procedure. Untuk membuat Macro bisa anda gunakan bentuk seperti ini

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

"P1" dan "P2" adalah parameter yang bisa anda gunakan pada macro. Parameter ini berbentuk optional, artinya bisa digunakan ataupun tidak.

```
Cetak_Kar MACRO Kar
MOV CX,3
MOV AH,02
MOV DL,Kar
Ulang:
INT 21h ; Cetak Karakter
LOOP Ulang
ENDM ; End Macro
ORG 100h
Proses:
Cetak_Kar 'S' ; Cetak Huruf S
INT 20h
RET
END
```

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.

Tugas Pendahuluan

1. buatlah dengan menggunakan procedure, program di bawah ini :

```
(1). Konversi  8 bit heksadesimal – desimal

(2)  Konversi  8 bit desimal – heksadesimal

(3)  Keluar Program

masukkan pilihan (1/2/3) : 1

input bilangan heksadesimal 8 bit : AA

dalam bentuk desimal              : 170

Lagi (y / t) = y

masukkan pilihan (1/2/3) : 2

input bilangan heksadesimal 8 bit : 66

dalam bentuk desimal              : 42
```

2. buatlah program untuk membaca maksimal 6 digit password, dengan password = **"TEKKOM".** dengan memanfaatkan instruksi PUSH dan POP Berikut ialah tampilan programnya

```
input Password : ABCDE

     SALAH PASSWORD

Lagi (y / t) = _
```

Apabila lebih dari 3 kali salah menginputkan, maka program berakhir.

Latihan

```
1.          ORG 100H
     MULAI:
             MOV DX, 'Z'
             PUSH DX
             MOV DX, 'Y'
             PUSH DX
```

```
            MOV AH, 02H
            MOV DL, 'A'
            INT 21H
            POP DX
            INT 21H
            POP DX
            INT 21H
    RET
```

2.
```
ORG 100H
    MOV BX, OFFSET TAMPIL
    MOV DL,'O'
    CALL BX
    MOV DL,'K'
    CALL BX

    TAMPIL PROC NEAR
         MOV AH,2
         INT 21H
         RET
    TAMPIL ENDP
    END
```

2.
```
ORG 100H
    PROSES: CALL CETAK_KAR
            MOV AH, 02H
            MOV DL, DATA
            INT 21H
            RET

    DATA DB 'R'

    CETAK_KAR PROC NEAR
              MOV AH, 02H
              MOV DL, 'S'
              INT 21H
              RET
    CETAK_KAR ENDP
```

3.
```
ORG 100h

   Proses:
   LEA DX,Kal
   PUSH DX
   MOV AH,09
   INT 21h
   LEA DX,Ganti
```

```
    INT 21h
    POP DX
    INT 21h
    Exit :
    INT 20h
    RET

    Kal DB 'UNIKOM UNIKOM $'
    Ganti DB 13,10,'$'
    Stacks DW ?
```

5.
```
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
ORG 100h
Proses:
Cetak_Kar 'P' ; Cetak Huruf P
Cetak_Kar 'C' ; Cetak Huruf C
INT 20h
RET
END
```

Tugas Praktikum

1. Buatlah program untuk menginput karakter ke stack, kemudian ambil dan cetak karakter tersebut apabila merupakan angka
2. Buatlah program untuk menginputkan beberapa nilai di program utama, kemudian menjumlahkan nilai tersebut dan menampilkan hasilnya dalam bentuk heksadesimal menggunakan procedure.

   Contoh Output

   ```
   Input bilangan      : 1320567
   Jumlah              : 18H
   ```