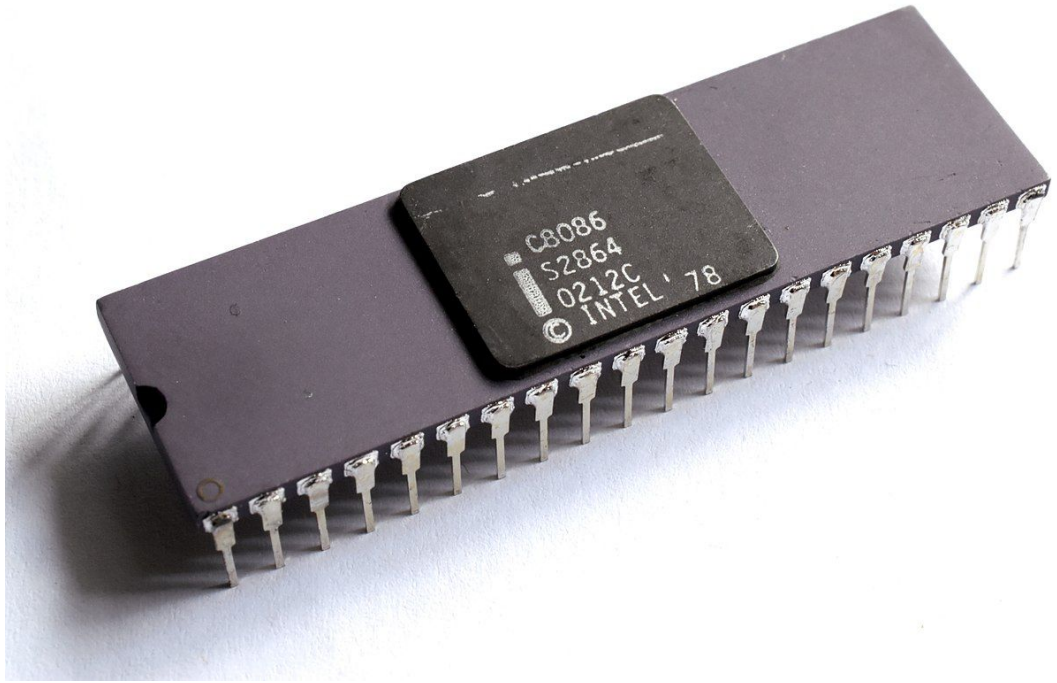
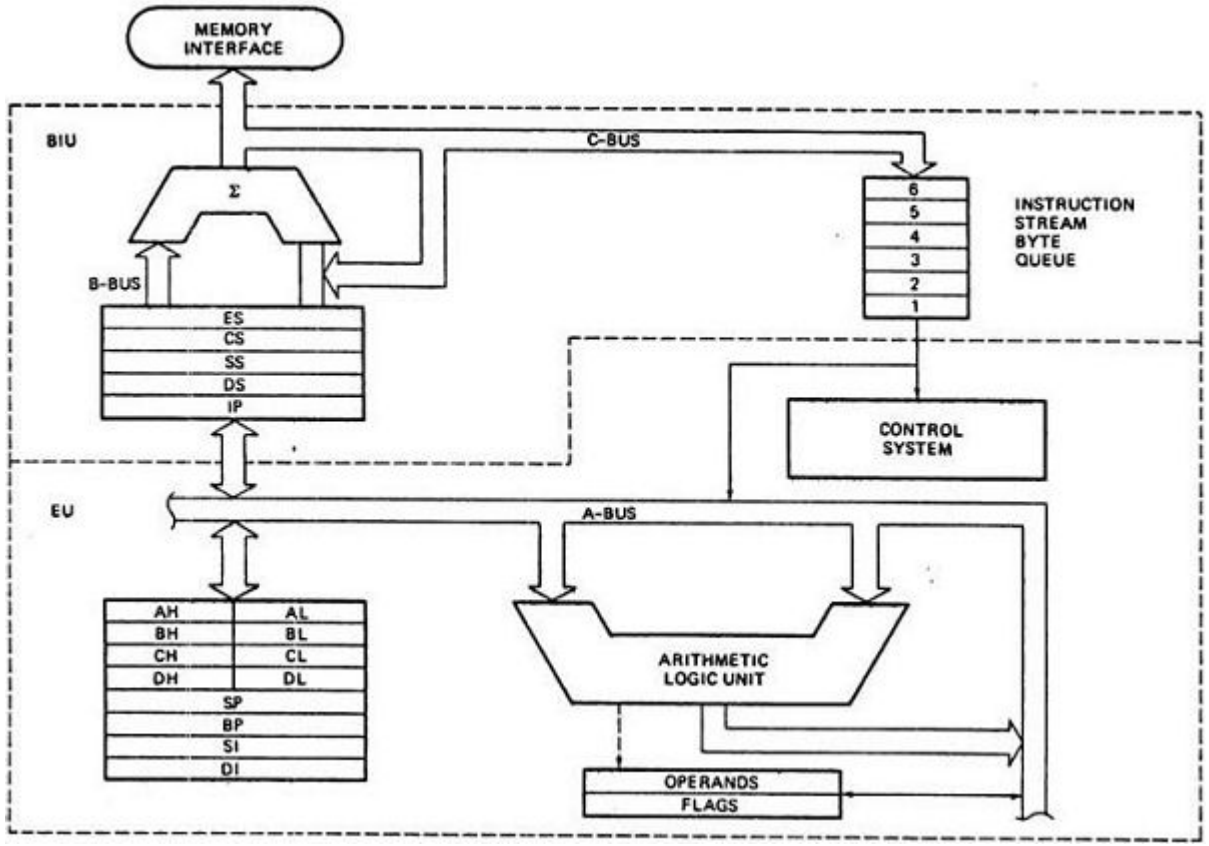
Asebmblersko programiranje i procesor i8086

Kratak opis karakteristika procesora

- Dizajn trajao od 1976. do 1978. godine, kada je lansiran
- 16 bitova
- frekvencija od 5 do 10 MHz
- 29000 tranzistora
- Širina memorijske magistrale 20 bitova



Arhitektura procesora i8086



BIU - bus interface unit
EU - execution unit

Arhitektura procesora podeljena je na dva dela. Uloga BIU dela jeste da ubrza rad sistema procesom **pajplajninga** (dok jedinica izvršavanja izvršava instrukciju, BIU donosi sledeću instrukciju.

Arhitektura procesora i8086 - registri opšte namene

- AX - accumulator register (AH / AL).
- BX - base address register (BH / BL).
- CX - count register (CH / CL).
- DX - data register (DH / DL).

Arhitektura procesora i8086 - ofset registri

- SI - source index register.
- DI - destination index register.
- BP - base pointer.
- SP - stack pointer.

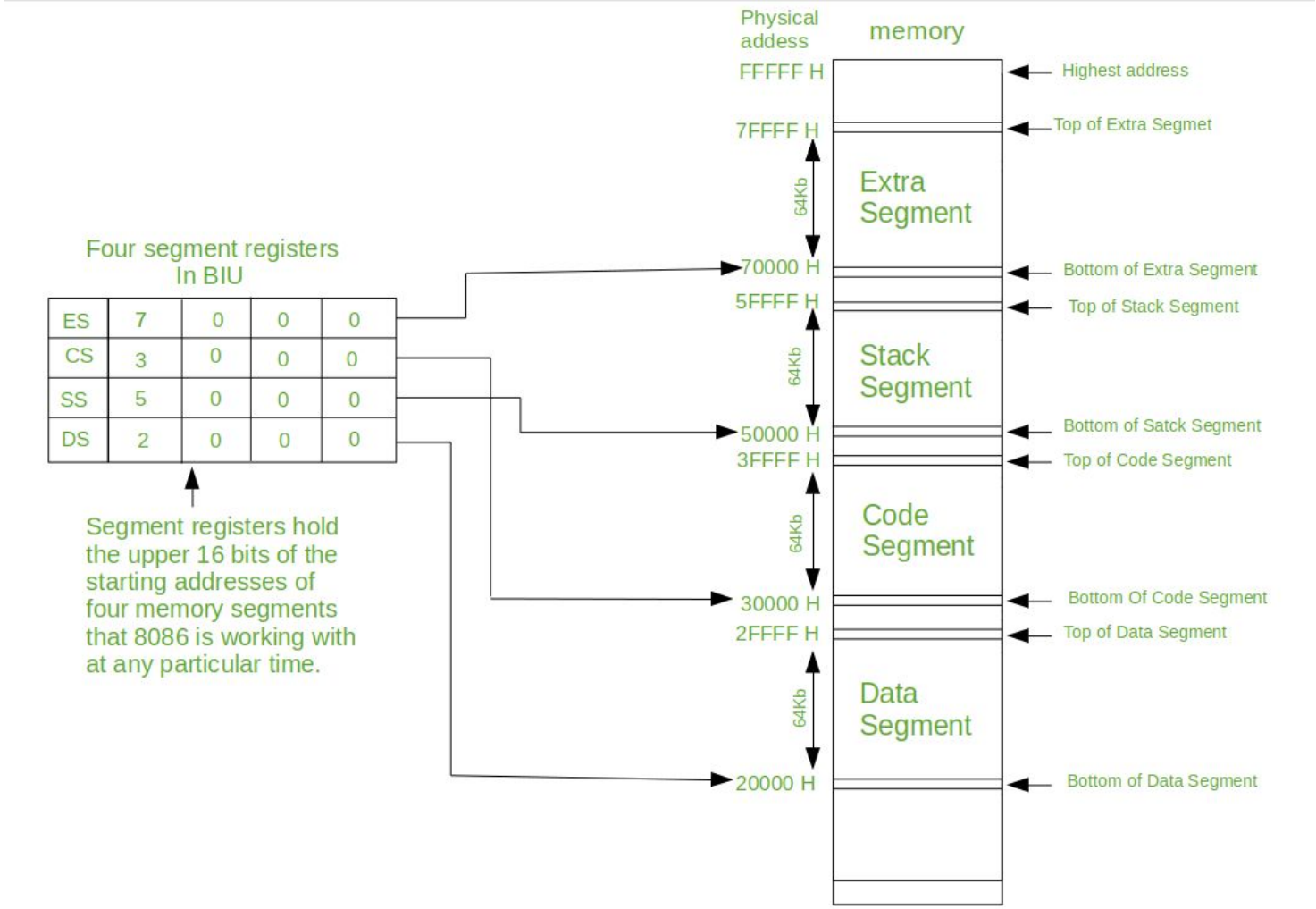
Arhitektura procesora i8086 - segmentni registri

- CS - pokazuje na segment koji sadrži kod programa (code segment)
- DS - pokazuje na segment gde su promenljive definisane (data segment)
- ES - segment opšte namene, za proizvoljno korišćenje, (extra segment)
- SS - pokazuje na stek segment (stack segment)

Arhitektura i8086

- Instruction System Byte Queue (Prefetch Queue) jeste bafer za nadolazeće instrukcije koje BIU donosi. Ona je veličine 6 **bajtova** (ne 6 instrukcija, instrukcije mogu biti različiti veličine)
- Procesor čeka da se isprazne 2 bajta iz ovog reda pre nego što donese nove (procesor je 16-obitni)
- Red se invalidira ukoliko dođe do skoka

Segmenti u memoriji i njihovi registri



Segmenti u memoriji i njihovi registri

- Efektivna edresa se dobija po formuli

  EA = Segmentni registar * 10h + ofset registar

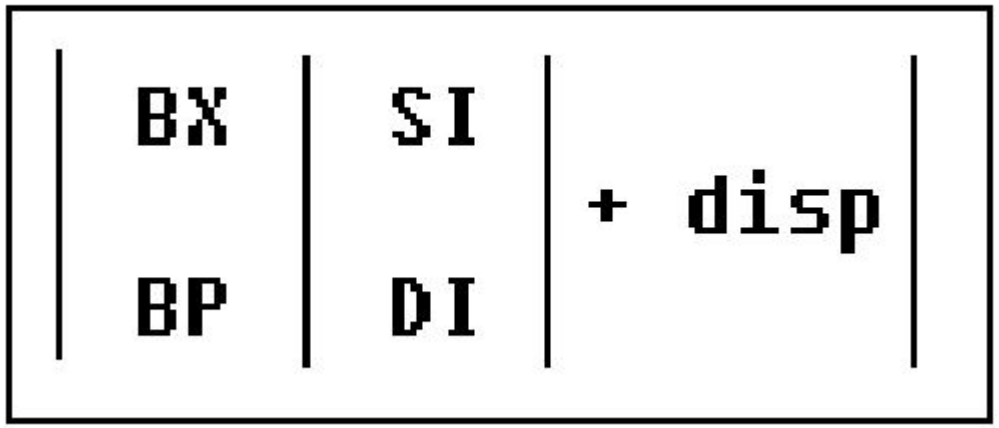
  Za segment podataka (**data segment**), možemo koristiti registre **BX**, **SI** i **DI**. Za **stek segment**, možemo koristiti registre **BP** i **SP**. Napomena, BL i BH registri ne mogu se koristiti za formiranje fizičke adrese. **IP** registar je uvek updaren sa **CS** segmentom.

Segmenti u memoriji i njihovi registri

- Generalno, u asembleru možemo da pristupamo različitim lokacijama u memoriji. Realnu fizičku adresu kojoj pristupamo računa sam procesor, a mi dajemo ofset

- [BX + SI + 5h] --> DS * 10h + BX + SI + 5h
- [DI + 16h] --> DS * 10h + DI + 16h
- ....

Segmenti u memoriji i njihovi registri



Registri koji se nalaze u istoj koloni ne mogu se koristiti da bi se formirao ofset.
Možemo birati registar iz svake kolone, a možemo i da poreskočimo kolonu.

Varijable

- Kompajler emu8086 podržava 2 tipa promenljivih, osmobitne i šesnaestobitne promenljive.

```
ime DB vrednost     ;za osmobitnu
ime DW vrednost     ;za šesnaestobitnu
```

- DB - define byte
- DW - define word

Varijable

- Promenljivoj se ne mora dati ime. U tom slučaju, ona će dobiti adresu, i to je jedini način na koji može da joj se pristupi. Vrednost promeljive može biti zapisana u proizvoljnom sistemu, ili može biti ? ukoliko nije inicijalizovana.

Nizovi

- Nizovi se mogu posmatrati kao ulančane varijable. Asembler emu8086 dozvoljava prilikom definicije varijabli da, pored brojeva, koristimo i slova (koja se treturaju kao njihove ASCII vrednosti)

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

Gore su definisana dva ista niza (asemblerautomatski pretrava string u navodnicima u skup bajtova)

Nizovi

- Elementima niza može se pristupati na standardan način, preko uglastih zagrada
- a[5]

- Pored toga, moguće je koristiti i registre BX, SI, DI i BP da pristupimo elementima

```
MOV SI, 5
a[SI]
```

Nizovi

- Ukoliko je potrebno da napravimo kolekciju elemenata koji se ponavljaju, koristimo operator DUP

ime DB broj_ponavljanja DUP(šta_se_ponavlja)

Primer c DB 5 DUP(1) daje isti rezultat kao
c DB 1, 1, 1,1,1

Primer d DB 2 DUP(1,2) daje isti rezultat kao
d DB 1,2,1,2

Dobijanje adrese varijable

- Kako bismo dobili adresu neke promenljive, koristimo instrukcije **LEA (load effective address)** ili **OFFSET.**

- **Primer: LEA AX, varijabla**

Prekidi

- Prekide možemo posmatrati kao određene procedure koje se izvršavaju u nekom trenutku. Možemo govoriti o **hardverskim** i o **softverskim prekidima.**
- Hardverski prekidi nastaju nada neki hardverski uređaj pošalje signal procesoru da obradi neki rezultat
- Softverski prekidi definisani su u kodu, i njihova uloga je da pozovu specijalne procedure.

Prekidi

- Softverski prekidi se u emu8086 emulatoru pozivaju preko komande INT

```
INT vrednost    ;vrednost je neki jednobajtni broj
```

- Na ovaj način moguće je definisati, na prvi pogled, 256 prekida. Međutim, svaki prekid može imati podfunkcije. Ove podfunkcije biraju se podešavanjem registra AH.

Instrukcija MOV

- Kopira vrednosti iz drugog parametra u prvi

MOV REG, memory
MOV memory, REG
MOV REG, REG
MOV memory, immediate
MOV REG, immediate

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.

memory: [BX], [BX+SI+7], variable, etc...

immediate: 5, -24, 3Fh, 10001101b, etc...

Instrukcija MOV

- Ukoliko želimo da koristimo MOV instrukciju da upisujemo nešto u segmentne registre, možemo, ali su pravila nešto drugačija.

MOV SREG, memory
MOV memory, SREG
MOV REG, SREG
MOV SREG, REG

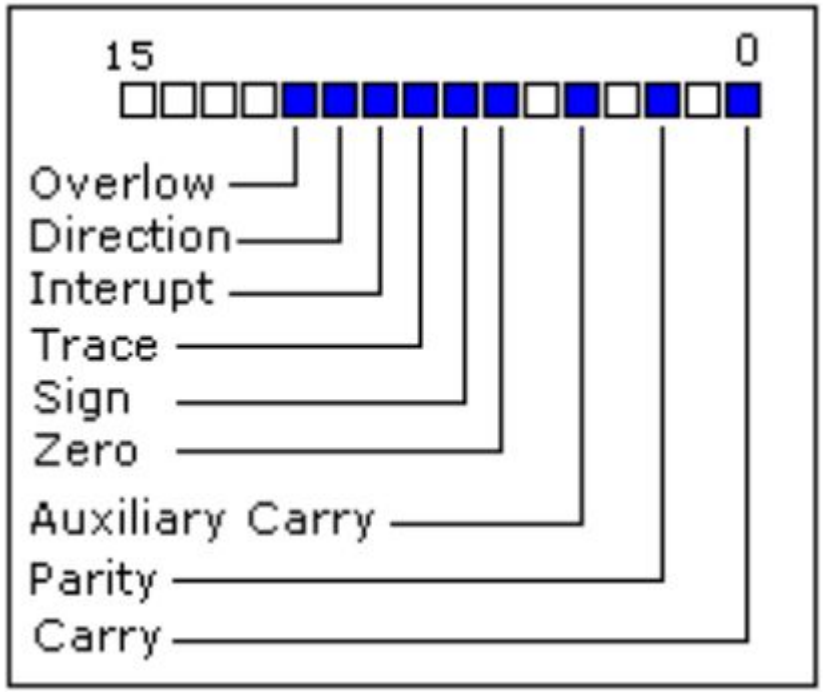
SREG: DS, ES, SS, and only as second operand: CS.

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.

memory: [BX], [BX+SI+7], variable, etc...

Aritmetičke i logičke funkcije

- Većina aritmetičkih operacija, pored svog očiglednog posla, takođe postavljau određene vrednosti u procesorski status registar (flag registar)



Aritmetičke i logičke funkcije

- **Carry Flag (CF)** - postavljen na 1 kada postoji neoznačeno prekoračenje
- **Zero Flag (ZF)** - postaljen na 1 kada je rezultat operacije 0
- **Sign Flag (SF)** -postavlje na 1 kada je rezultat negativan
- **Overflow Flag (OF)** - postavljen na 1 kada postoji označeno prekoračenje

Aritmetičke i logičke funkcije

- **Parity Flag (PF)** - postavljen na 1 kada postoji paran broj bitova jedinice u rezultatu (postamtraju se samo najnižih 8 bitova)
- **Auxiliary Flag (AF)** - postavljen na 1 ukoliko postoji neoznačeno prekoračenje na najnižoj nibli
- **Interrupt enable Flag (IF)** - postavljen na 1 ukoliko CPU reaguje na prekide od eksternih uređaja
- **Direction Flag (DF)** - za procesuiranje lanaca podataka, 0 ili jedan definišu smer u kojem se procesuiranje izvršava

Prva grupa funkcija

ADD, SUB,CMP, AND, TEST, OR, XOR

REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate

CF, ZF, SF, OF, PF, AF.

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.

memory: [BX], [BX+SI+7], variable, etc...

immediate: 5, -24, 3Fh, 10001101b, etc...

Druga grupa funkcija

Second group: **MUL**, **IMUL**, **DIV**, **IDIV**

These types of operands are supported:

REG
memory

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.

memory: [BX], [BX+SI+7], variable, etc...

Jedino operacije MUL i IMUL postavljaju flegove, i to CF i OF. Ovi flegovi su postavljeni na 1 ukoliko je rezultat veći od veličine registra, odosno 0 u suprotnom slučaju. Sve operacije podrazumevaju da se množenik nalazi u registru AX (AL ukoliko je brojilac jednobajtan)

Treća grupa funkcija

Third group: **INC**, **DEC**, **NOT**, **NEG**

These types of operands are supported:

REG
memory

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.

memory: [BX], [BX+SI+7], variable, etc...

INC, **DEC**
ZF, **SF**, **OF**, **PF**, **AF**.

NOT

ne utiče ni na koji fleg

NEG
CF, **ZF**, **SF**, **OF**, **PF**, **AF**.

Kontrola toka

- Flegovi imaju ulogu u kontroli toka. U zavisnosti od vrednosti flegova, funkcije za skakanje mogu da se izvrše ili ne. Osnovna instrukcija za skok je JMP instrukcija. Kada procesor naiđe na nju, on odmah prelazi na adresu na koju JMP pokazuje.

Kontrola toka

JZ , JE	Jump if Zero (Equal).	ZF = 1	JNZ, JNE
JC , JB, JNAE	Jump if Carry (Below, Not Above Equal).	CF = 1	JNC, JNB, JAE
JS	Jump if Sign.	SF = 1	JNS
JO	Jump if Overflow.	OF = 1	JNO
JPE, JP	Jump if Parity Even.	PF = 1	JPO
JNZ , JNE	Jump if Not Zero (Not Equal).	ZF = 0	JZ, JE
JNC , JNB, JAE	Jump if Not Carry (Not Below, Above Equal).	CF = 0	JC, JB, JNAE
JNS	Jump if Not Sign.	SF = 0	JS
JNO	Jump if Not Overflow.	OF = 0	JO
JPO, JNP	Jump if Parity Odd (No Parity).	PF = 0	JPE, JP

Procedure

- Procedure su deo koda koji se može pozvati u okviru programa. Procedure čine da se program može lakše čitati i da je bolje struktuiran. Sintaksa za definisanje neke procedue je:

```
ime PROC
 ;kod
 ;kod
 RET
ime ENDP
```

Procedure

- Kako bi se procedura pozvala, koristi se ključna reč CALL.

Stek

- Uloga steka jeste u čuvanju privremenih podataka, kada treba da im se pristupa po principu LIFO. Implicntno, stek se koristi kada se pozivaju instrukcije CALL i INT, kako bi se na njega postavila adresa povratka.
- Povratak na tu adresu izvršava se instrukcijama RET (u slučaju CALL instrukcije), ondnosno IRET (u slučaju softverskog prekida)

Stek

- Steku takođe možemo da pristupimo i manuelno, koristeći instrukcije PUSH i POP.

PUSH REG
PUSH SREG
PUSH memory
PUSH immediate

REG: AX, BX, CX, DX, DI, SI, BP, SP.

SREG: DS, ES, SS, CS.

memory: [BX], [BX+SI+7], 16 bit variable, etc...

immediate: 5, -24, 3Fh, 10001101b, etc...

Stek

POP REG
POP SREG
POP memory

REG: AX, BX, CX, DX, DI, SI, BP, SP.

SREG: DS, ES, SS, (except CS).

memory: [BX], [BX+SI+7], 16 bit variable, etc...