**Limbaj masina** = totlitatea instructiuniilor masina puse la dispozitie de processor

**Limbaj de asamblare** = Limbaj symbolic (instructiuniile coincide cu instructiuniile masina)

**Eticheta** = Nume de variabila sau zona de memorie

* 2 categorii de etichete: **eticheta cod**: apar in cadrul secventelor de instructiuni pt a define destinatia de dransfer de date

**eticheta de date**: unele locatii de memorie

* **Valoarea** unei etichete in limbaj de asamblare este un nr intreg reprezentand adresa (FAR) a instructiunii, directive sau datelor ce urmeaza etichetei
* Daca numele etichetei este pusa intre paranteze drepte: [] inseamna ca ne referim la valoarea din memorie a ei

**Instructiuni** = Mnemonice ce sugereaza actiunea

* Asamblorul genereaza octetii care genereaza respectiva instructiune
* Mnemonice de instructiuni si nume de directive

**Directive** = Indicatii date asamblorului in scopul generarii corecte de octeti

* Tipuri de directive:
* **DIRECTIVA SEGMENT** nume [tip] [ALIGN=aliniere] [combinare] [utilizare] [CLASS=clasa]
  + Numele segmentului = adresa de segment
  + TIP: permite selectarea unui model de folosire al segmentului si poate fi:
    - code – segmentul va contine cod: poate fi citi si executat, dar nu scris
    - data – segmentul va contine date: poate fi citit si scris, dar nu executat
    - rdata – segmentul contine doar constante: poate fi doar citit
  + ALINIERE: specifica multiplul numarului de octeti la care trebuie sa inceapa segmentul respectiv: putere a lui 2 intre 1 si 4096: este 1 implicit
  + COMBINARE: modul in care se combina cu alte module:
    - PUBLIC: e vizibil de orice alt modul, poate fi combinat cu orice alt modul
    - COMMON: acest modul se suprapune peste inceputul oricarui alt modul cu acelasi nume => un segment cu dimensiunea egala a celui mai mare segment cu acelasi nume
    - PRIVATE: nu e vizibil, nu poate fi combinat cu nici un alt modul
    - STACK: segmentele cu acelasi nume vor fi concatenate
  + CLASS: are rolul de a permite stabilirea ordinii in care editorul de legaturi plaseaza segmentele in memorie. Toate segmentele cu aceeasi clasa vor plasate intr-un singur bloc
  + segment code use32 class=CODE
  + segment data use32 class=DATA
* **DIRECTIVE PENTRU DEFINIRE A DATELOR**:
  + Definire date = declarare (specificare atribute) + alocare (rezervare spatiu de memorie)
  + Tip de date (TD) = byte/word etc (Dimensiune de reprezentare)
  + Variabila = nume, set de atribute, adresa, valoare
  + Set de atribute = TD, Scope (domeniu de vizibilitate), durata de viata, clasa de memorie, etc (in C)
  + Directive de definire: DB, DW, DD, DQ, DT (10 octeti)
    - A db 1,2,3,4; Se intializeaza patru variabile in memorie (4 octeti) sau un tablou de 4 elemente
* **DIRECTIVE DE DATE NEINITIALIZATE** (valoarea initializarii e 0):
  + RESB, RESW, RESD, RESQ, REST
* **DIRECTIVA TIMES**:
  + Permite asamblarea repetata a unei instructiuni sau definitii de date:
  + TIMES factor expresie; factor e un numar, si expresie se va copia de factor (in asamblare)
* **DIRECTIVA EQU**:
  + Permite atribuirea unei valori numerice constante sau sir de caractere unei etichete fara alocarea de spatiu de memorie sau generare de octeti
  + Nume EQU expresie

**Contor de locatii** = Numar intreg generat de asamblor

* În fiecare moment, valoarea contorului coincide cu numărul de octeţi generaţi corespunzător instrucţiunilor şi directivelor deja întâlnite în cadrul segmentului respectiv

**Operanzii** = parametrii care definesc valoriile ce vor fi prelucrate de instructiuni sau directive

* Pot fi: registrii, constante, etichete, expresii, cuvinte cheie sau alte simboluri
* 3 tipuri:
  + Operanzi imediati: 1, 2, 3 etc
    - Constante numerice determinabile la momentul asamblarii
    - 0x 0h pentru baza 16, 0d 0t pentru zecimal, 0o 0q pentru octal, ob oy pentru binar
    - Aceste litere poti fi puse si la final: 101010b, 1234AFh
    - Deplasamentele etichetelor sunt variabile determinabiile la momentul asamblarii
  + Operanzi registru: eax, ebx, esi etc
    - Mov eax,ebx -> accesare directa
    - Mov eax,[ebx] -> accesare indirecta
  + Operanzi in memorie: byte[a], word[b] etc
    - Operandul cu adresare directa: o constanta care indica adresa (segment si deplasament) pot fi: etichete sau valoarea contorului de locatii (b db $-a)
      * Deplasamentul e calculat in timpul asamblarii, adresa fiecarui operand este calculata in momentul editarii de legaturi (LINKING TIME), iar adresa fizica efectiva in momentul incarcarii
      * Un deplasament este intotdeauna raportat la un segment (CS, SS, DS etc)
      * Asocieri implicite de asociere:
        + CS daca pentru etichete de cod destimatie (jmp repeta, call [printf], ret, jz etc)
        + SS daca in SIB se foloseste ESP sau EBP ca baza
        + DS pentru restul
    - Operandul cu adresare indirecta: Utilizeaza registrii pentru a indica zone de memorie:
      * [baza] + [index\*scala] + [constanta]
      * Constanta este expresie a carei valoare este determinabila la momentul asamblarii
      * Pot fi rescrise si formula sa fie tot valida: mov al,[ebx\*3]; ⬄ mov al,[ebx+ebx\*2]
      * Scala poate fi 1 2 sau 4

**Operatori** = pentru combinarea, compararea, modificarea si analiza operatoriilor.

TIPURI DE OPERATORI:

* - (scadere), + (adunare), ~(complement fata de 1, negare bit cu bit), !(negare logica !0=1 si !a=0 daca a!=0), \*(inmultire), / (catul impartirii fara semn), // (catul impartirii cu semn), % (restul impartirii cu semn), %% (restul impartirii fara semn), << (deplasare la stanga pe biti), >> (deplasare la dreapta pe biti) & („si” bit cu bit), | („sau” bit cu bit), ^ („sau exclusiv” bit cu bit)
* Operator de **indexare**: „[]” cand e folosit cu formula de calcul al offset-ului
* Operator de **specificare al segmentului**: „:” ex: mov eax,[SS:EBX+4] (se refera exact la segmentul de stiva si suprascrie regulile implicite de setare al segmentului)
* Operator de **tip**: byte, word, dword, qword -> forteaza o expresie sa fie tratata ca avand o anumita dimensiune (se foloseste adesea cand ne raportam la memorie)
* Valoarea operanziilor este determinabila la momentul asamblarii pentru operanzii imediati si operanzii cu adresare directa (offset-ul doar), in momentul incarcarii (LOADING TIME) pentru operanzii cu adresare directa (offset + segment -> adresa FAR) (acest pas implica un proces numit RELOCARE a adreselor adica determinarea adreselor de segment pe baza selectorului de segment) si in momentul executiei (RUN-TIME) pentru operanzii registru si cei adresati indirect (cu formula de calcul al offset-ului in care intra o formula bazata si/sau indexat-scalata)

**Expresie** = Operanzi + Operatori

* Operatorii sunt determinabili la momentul asamblarii ( cu exceptia formulii de la 2 noaptea )

**Depasire** = O conditie/situatie matematica care exprima faptul ca rezultatul unei operatii nu a incaput in spatiul rezervat acestuia

* Prin setarea CF=1 sau OF=1, procesorul ne transmite mesajul ca ambele interpretari in baza 10 (cu semn sau fara semn) sunt INCORECTE (CF=1 => interpretarea fara semn are rezultatul gresit; OF=1 => interpretarea cu semn are rezultatul gresit)

**Adresa unei locatii** = Nr de octeti consecutivi dintre inceputul memoriei RAM si inceputul locatiei respective

**Segment** = O succesiune continua de locatii de memorie, menita sa deserveasca scopuri similare

* O diviziune logica a memoriei unui program
* Contine o adresa de baza (inceput), limita (dimensiune), si tipul acesteia (toate reprezentate pe 32 de biti)
  + Segment de cod: contine instructiuni masina (CS – selector de segment)
  + Segment de date: contine definiri de date (DS-selector de segment)
  + Segment de stiva: SS-selector de segment
  + Extra segment: ES
  + FS, GS -> selectori care indica spre segmente suplimentare neavand atribuiri implicite
  + EIP -> offsetul instructiunii curente in cadrul segmentului de cod curent

**Specificare de adresa** = O pereche: Selector de segment si offset ([DS:EBX])

* Selectorul de segment este definit de catre sistemul de operare (SO)
* Acea pereche se mai numeste ADRESA FAR

**Adresa NEAR** = atunci cand o adresa se precizeaza doar prin offset

* Daca se depaseste limita => MEMORY VIOLATION ERROR
* Calculul pentru validare este facuta de componenta ADR din BIU
* Este intotdeauna in interiorul din cele 4 segmente active
* Daca segmentele incep de la 0 => Model de memorie flat

**Paginare** = Implica impartirea memoriei virtuale in pagini care sunt asociate memoriei fizice disponibile

* Sarcina Sistemului de operare (SO)

**Instructiune masina** = O secventa de la 1 pana la 15 octeti

* Reprezinta o operatie de efectuat
* Maxim 2 operanzi (sursa si destinatie)
* Maxim un operand din memoria RAM
* Forma generala:
  + [prefixe] + cod + [Mod R/M] + [SIB] + [deplasament] +[imediat]
  + Prefixe: controleaza modul in care o instructiune se executa
  + Mod R/M: Natura si locul operanziilor (registru/memorie) si se poate combina cu octetul SIB
  + Deplasament (displacement): forma de adresare particulara
  + Imediat: consanta numerica

**Mod adresare** = Modul in care ne adresam la un operand

* Mod adresare la memorie:
  + Adresare directa: daca apare doar o constanta (imediat si o eticheta): mov **byte[a]**,1
  + Adresare bazata: daca apare baza: mov **byte[ebx]**,1
  + Adresare scalat-indexata: daca apare index (implicit si scala): mov **byte[ebx\*2]**,1
  + Adresa relativa: pozitia urmatoarei instructiuni de executat in raport cu pozitia curenta
  + O adresa care nu este directa se numeste indirecta
* Mod registru: mov **EAX**,17
* Mod imediat: mov EAX,**17**

**Arhitectura microprocesoarelor**:

* **EU** = Executive unit
  + Executa instructiuniile masina prin intermediul componentei ALU (aritmetic and logic unit)
* **BIU** = Bus interface unit
  + Pregateste executia fiecarei instructiuni masina
* Cele doua lucreaza in paralel

**Flag** = un indicator reprezentat pe un bit (0 sau 1)

* 2 tipuri de flaguri: Cele cu efect de dupa o operantie, si cele ce au efect dupa setarea lor
* CF, OF, ZF, SF, PF, AF, PF -> dupa operatie
* DF, TF IF -> efect dupa setare

**Conversie distructive** = Schimba efectiv numarul

* CBW, CWD etc
* Movzx, movsx, mov

**Conversie nedistructiva =** Nu schimba efectiv numarul

* BYTE, WORD, DWORD, QWORD

**Resurse la subrutine** (programare Multimodul)

* **Resurse volatile** = Acei registrii pe care conventia de apel ii defineste ca apartinand subrutinei apelate, ca atare apelantul avand datoria sa le salveze daca are nevoie de ele, si sa le restaureze dupa (adica registrii folositi in alt modul ca parametrii)
* **Resurse nevolatile** = Adrese de memorie/registrii care nu ii apartin explicit subrutinei chemate, iar ca atare, daca acesta doreste sa efectueze modificari este necesar sa le salveze inainte de iesire
* Diferenta apare in privinta cui are obligatia sa faca salvarea si returnarea
* Volatile -> Apelantul (unde apelam o functie)
* Nevolatile -> Codul apelant (in subprogram)