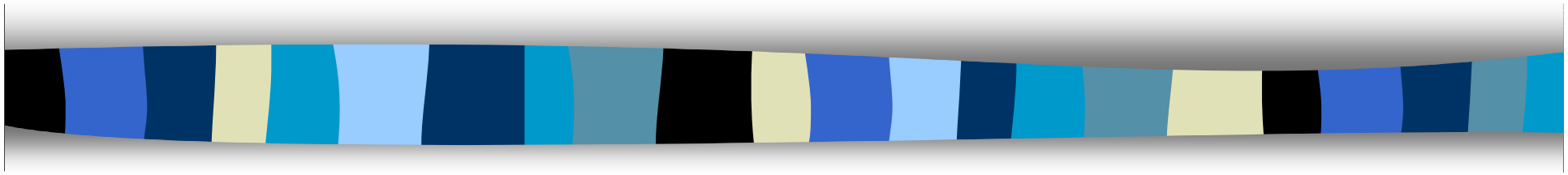
Technika mikroprocesorowa



Mikroprocesor 8086 – architektura

Materiały z wykładu na stronie:

www.m.szmajda.po.opole.pl

Technika mikroprocesorowa (8086)

Przygotował: dr inż. Krzysztof Górecki

Literatura

1. Z. Mroziński: „Mikroprocesor 8086”, WNT 1992, Warszawa
2. A. Andrusz, M. Sokołowski: „Mapa pamięci IBM/PC w przykładach”, Lynx-Soft, Gdańsk 1995
3. S. Kruk: „Programowanie w języku asembler”, PLJ, Warszawa 1993
4. A. Dudek: „Jak pisać wirusy”. Sztuka programowania. RM, 1994
5. G. Syck: „Turbo Asembler. Biblia użytkownika”, LT&P, 2002

Architektura – definicja

Architektura komputerów – jest to opis komputera z punktu widzenia programisty w języku niskiego poziomu (asemblera), łącznie z semantyką i składnią rozkazów oraz opisem typów danych.

Oznacza to że komputer (mikroprocesor) przedstawia się jako zespół dostępnych programowo rejestrów, miejsc w pamięci i układów we/wy, na których operuje określony zbiór rozkazów, reprezentowanych symbolicznie w języku asemblera.

Architektury komputerów - podział

- Architektura von Neumanna (komputer jednoadresowy) – zawiera pojedynczą pamięć, która jest wspólna dla słów operacyjnych i argumentów lub ich adresów
- Architektura harvardzka (typu Harvard) – zawiera odrębne magistrale dla rozkazów i danych

Architektura von Neumanna

System komputerowy zbudowany w oparciu o architekturę Von Neumanna powinien:

- mieć skończoną i funkcjonalnie pełną listę rozkazów
- mieć możliwość wprowadzenia programu do systemu komputerowego poprzez urządzenia zewnętrzne i jego przechowywanie w pamięci w sposób identyczny jak danych
- dane i instrukcje w takim systemie powinny być jednakowo dostępne dla procesora
- informacja jest tam przetwarzana dzięki sekwencyjnemu odczytywaniu instrukcji z pamięci komputera i wykonywaniu tych instrukcji w procesorze.

Architektura harvardzka

W stosunku do architektury

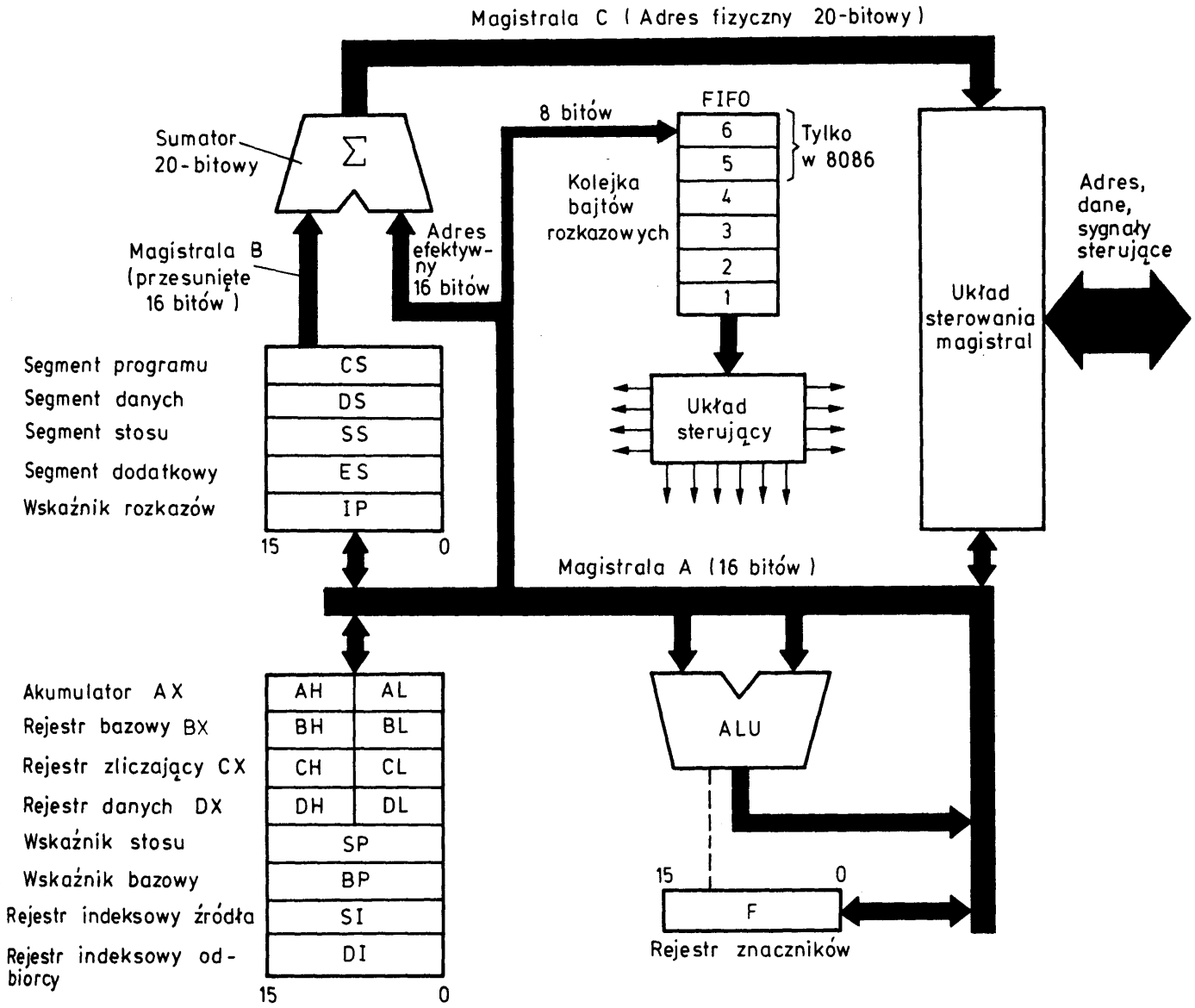
Von Neumanna, pamięć przydzielana danym programu jest oddzielona od pamięci rozkazów.

Architektura – podział ze względu na sposób adresowania

- Komputer jednoadresowy – np. zawsze akumulator i adres danej np. w pamięci
- Komputer dwuadresowy – wykonują rozkazy dwuadresowe (np. 8086) – pamięć – pamięć, pamięć – rejestr, rejestr – pamięć.
- Komputer trójadresowy – (np. CRAY-1)

Architektura – podział ze względu na rodzaj wykonywanych rozkazów

- Architektura RISC – komputer o zredukowanym zbiorze rozkazów
- Architektura CISC – komputer o zbiorze rozkazów złożonych (np. 8086) – obecnie np. jądro procesora Pentium IV posiada architekturę RISC



Funkcjonalny schemat blokowy mikroprocesora 8086/8088 firmy Intel

Magistrala adresowa i danych i przestrzeń adresowa

Magistrala adresowa i magistrala danych mikroprocesorów 8086 jest 16 bitowa – jest to komputer 16-bitowy.

Przestrzeń adresowa jest

20 bitowa–1 MB (adres od 0h – 0FFFFFh)

Adres fizyczny a adres logiczny

Adres fizyczny mikroprocesorów 8086 jest 20-bitowy.

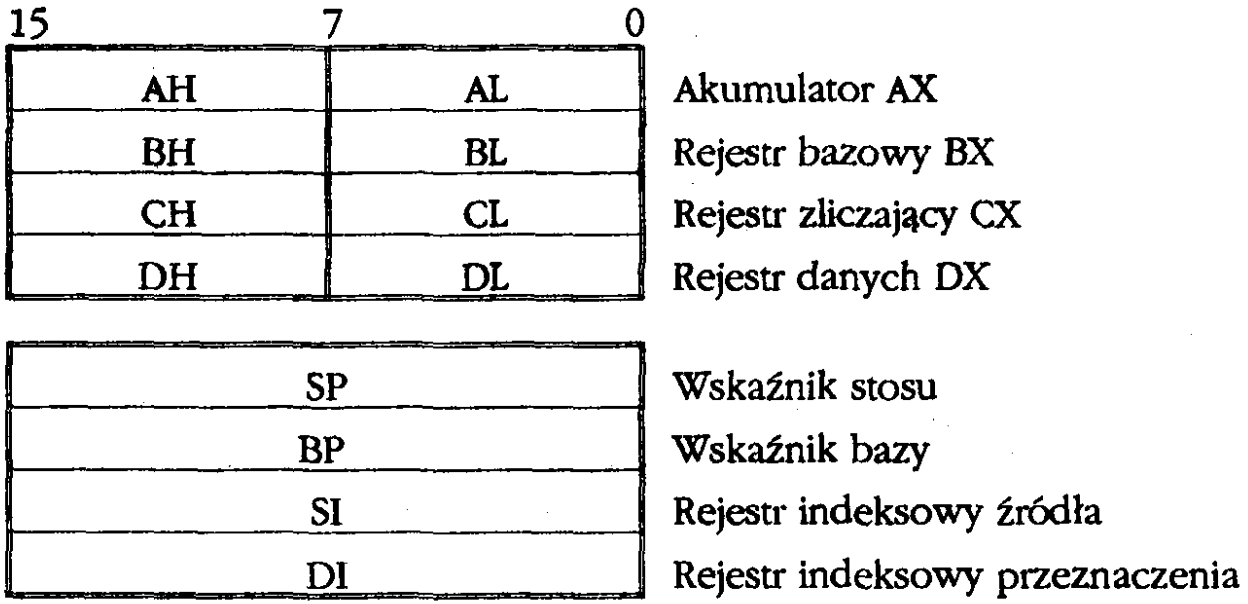
Adres logiczny składa się z dwóch liczb 16-bitowych, z tzw. segmentu i offsetu.

Jako segment wykorzystuje się jeden z rejestrów segmentowych (CS, DS, SS, ES), a jako offset najczęściej (SI, DI, BP, SP)

Adres fizyczny = segment *16 +offset

Rejestry w mikroprocesorze 8086

- osiem rejestrów ogólnego przeznaczenia



Opis rejestrów ogólnego przeznaczenia

AX – akumulator. Niektóre instrukcje mogą wykorzystywać tylko akumulator (mnożenie, dzielenie)

BX – rejestr bazowy. Zawiera przemieszczenie względem początku segmentu danych

CX – rejestr licznikowy. Określa liczbę wykonań grupy instrukcji (licznik pętli)

DX – Rozszerzenie akumulatora w 32 bitowych operacjach

Opis rejestrów ogólnego przeznaczenia c.d.

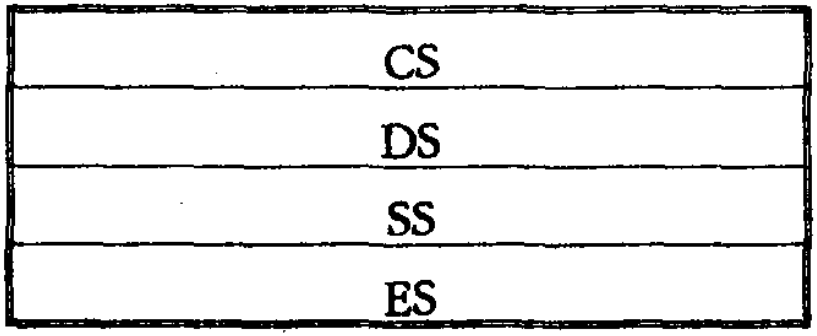
SP – wskaźnik stosu

BP – rejestr bazowy. Zawiera przemieszczenie względem początku segmentu stosu

SI – rejestr indeksowy źródła. Zawiera adres źródła w operacjach łańcuchowych

DI – rejestr indeksowy przeznaczenia. Zawiera adres przeznaczenia w operacjach łańcuchowych

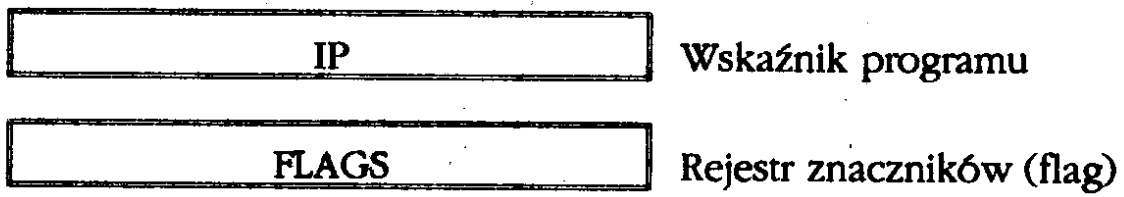
Rejestry segmentowe



Rejestr segmentowy programu
Rejestr segmentowy danych
Rejestr segmentowy stosu
Rejestr segmentowy dodatkowy

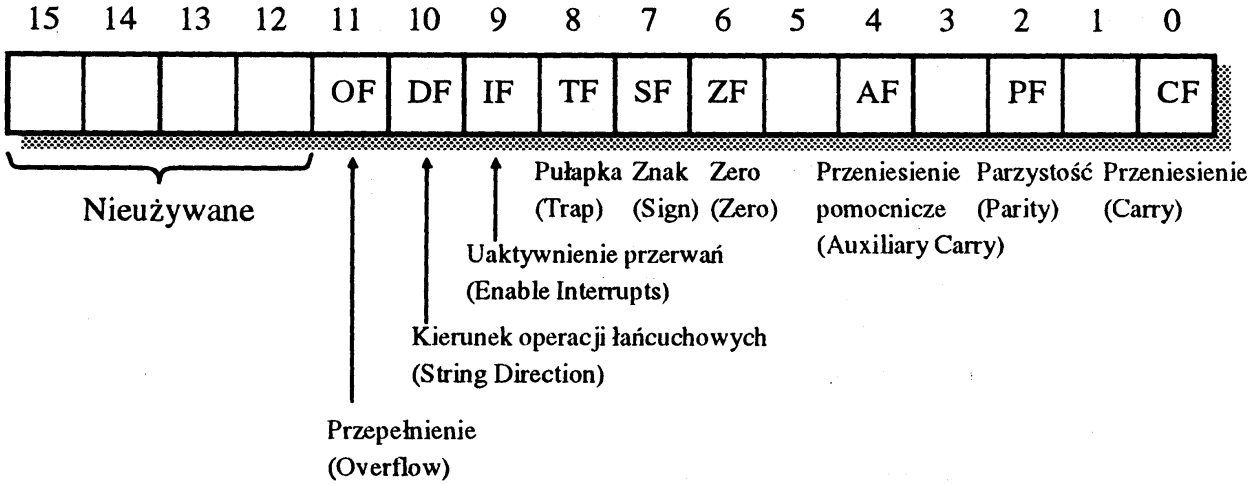
CS – numer segmentu kodu programu
DS – numer segmentu danych programu
ES – numer dodatkowego segmentu danych programu
SS – numer segmentu stosu programu

Wskaźnik rozkazów i rejestr znaczników (flagowy)



IP – (16-bitowy) wskaźnik rozkazów

FLAGS – (16-bitowy) rejestr znaczników

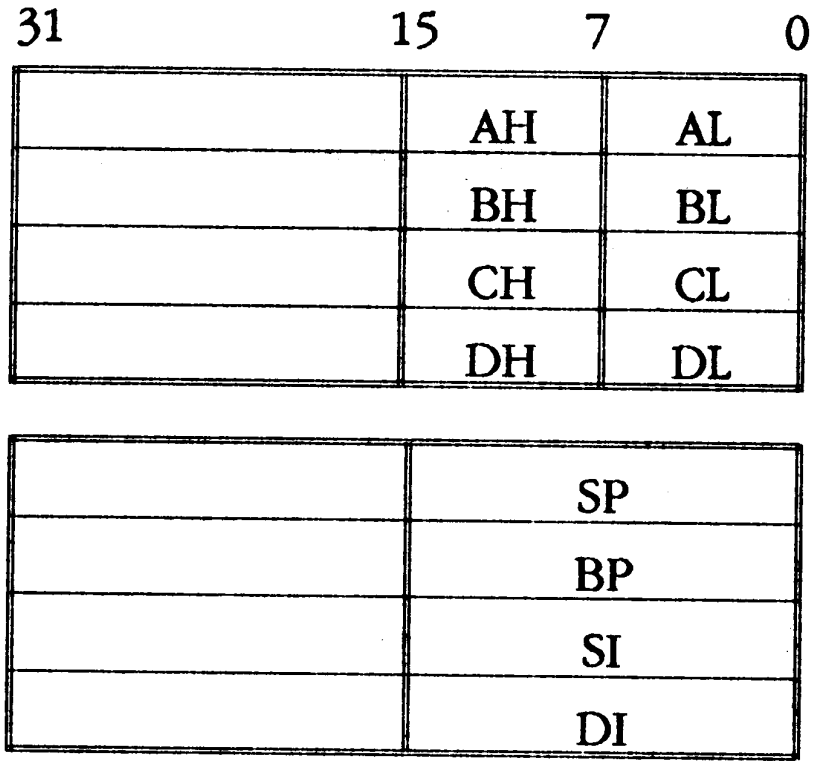


Mikroprocesory rodziny 8086

Tabela 3.1. *Wybrane parametry procesorów 80x86*

Procesor	Data wprowadzenia	Mikro-architektura	Pierwotna częstotliwość zegara	Rozmiar dostępnych rejestrów (w bitach)	Przestrzeń adresowa pamięci	Pamięć podręczna
8086	1978		8 MHz	16	1 MB	—
Intel 286	1982		12,5 MHz	16	16 MB	—
Intel 386 DX	1985		20 MHz	32	4 GB	—
Intel 486 DX	1989		25 MHz	GP: 32 FPU: 80	4 GB	8 KB L1
Pentium	1993	P5	60 MHz	GP: 32 FPU: 80	4 GB	16 KB L1
Pentium Pro	1995	P6	200 MHz	GP: 32 FPU: 80	64 GB	16 KB L1 512 KB L2
Pentium II	1997	P6	266 MHz	GP: 32 FPU: 80 MMX: 64	64 GB	16 KB L1 512 KB L2
Pentium III	1999	P6	700 MHz	GP: 32 FPU: 80 MMX: 64 XMM: 128	64 GB	16 KB L1 256 lub 512 KB L2
Pentium 4	2000	Intel NetBurst micro-architecture	1,4 GHz	GP: 32 FPU: 80 MMX: 64 XMM: 128	64 GB	12 KB µop Execution Trace Cache 8 KB L1 256 KB L2

Rejestry 32-bitowe ogólnego przeznaczenia mikroprocesorów 80386 i nowszych



Akumulator EAX (AH i AL dają AX)

Rejestr bazowy EBX (BH i BL dają BX)

Rejestr zliczający EAX (CH i CL CX)

Rejestr danych EDX (DH i DL dają DX)

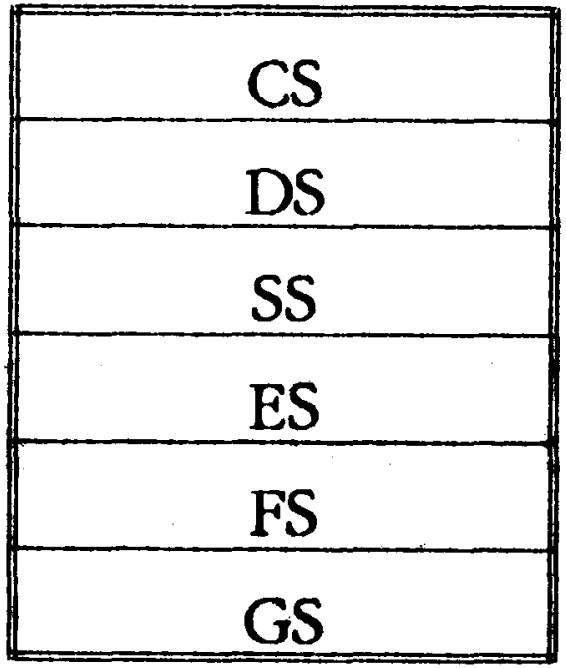
Wskaźnik stosu ESP

Wskaźnik bazy EBP

Rejestr indeksowy źródła ESI

Rejestr indeksowy przeznaczenia EDI

Rejestry segmentowe 32-bitowe



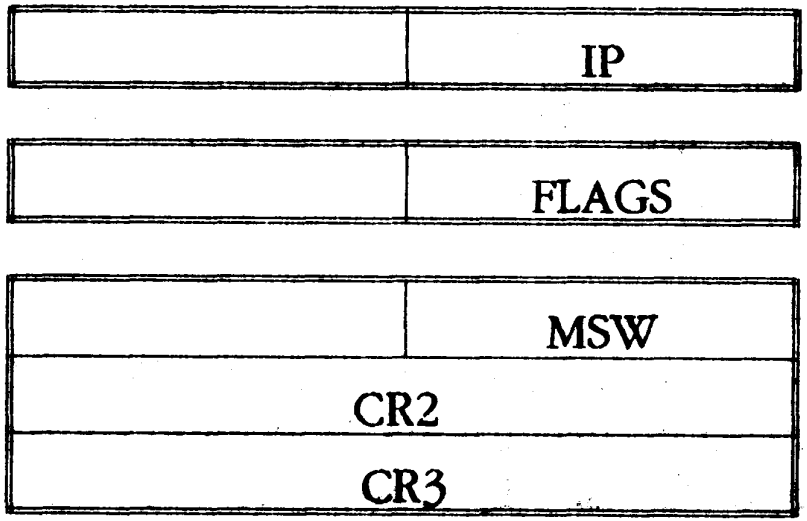
Rejestr segmentowy programu

Rejestr segmentowy danych

Rejestr segmentowy stosu

Rejestr segmentowy dodatkowy

Rejestry 32-bitowe c.d.

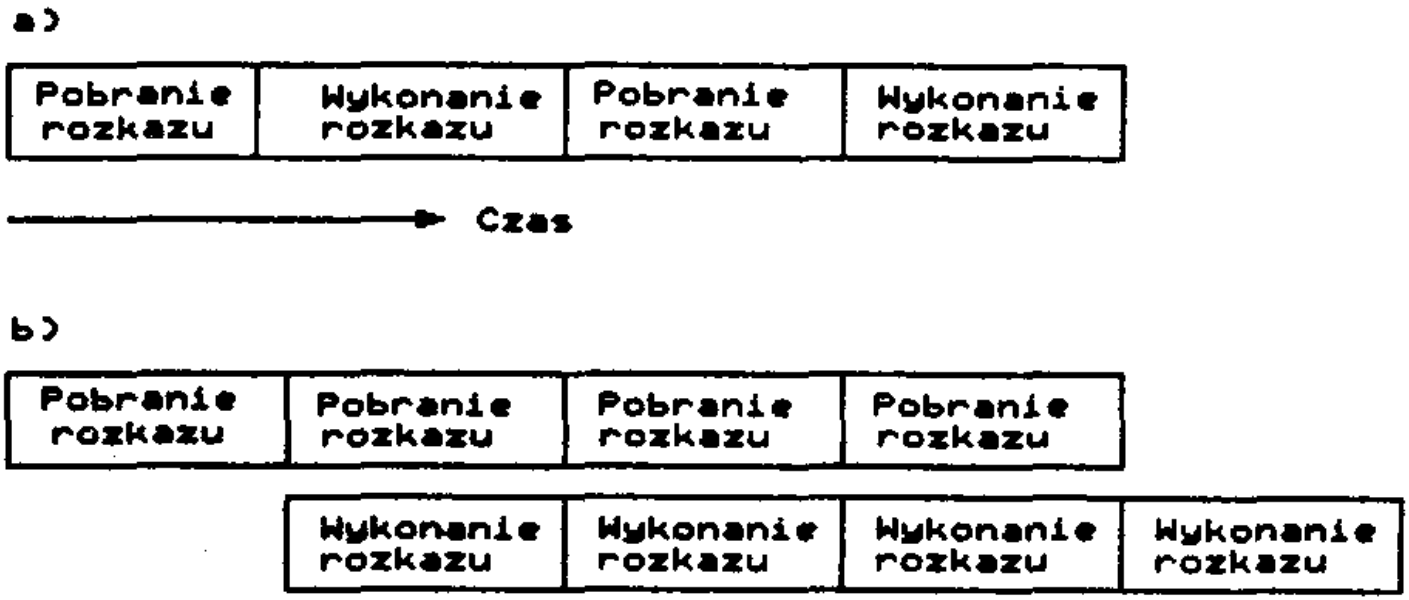


Wskaźnik programu EIP

Rejestr znaczników (flag) EFLAGS

CR0
blok rejestrów sterujących
pracą procesora

Wykonywanie rozkazów w mikroprocesorach 8086



Wykonywane operacje przez mikroprocesor 8086

- przesłania typu rejestr-rejestr, stos-rejestr, rejestr-stos, rejestr-we/wy, we/wy-rejestr
- operacje arytmetyczne: dodawanie, odejmowanie, porównanie, mnożenie i dzielenie dwójkowych liczb całkowitych bez znaku (w kodzie NB) lub ze znakiem (w kodzie U2), jak również liczb dziesiętnych (w kodzie BCD)
- operacje logiczne AND, OR, XOR i NOT, oraz różne rodzaje przesunięć logicznych

Wykonywane operacje przez mikroprocesor 8086 c.d.

- operacje tekstowe (gdy słowa traktowane są jak łańcuchy znakowe): przepisywanie ciągu słów z jednego obszaru do drugiego, wyszukiwanie i porównywanie słów
- różne rodzaje skoków
- sterowanie stanem mikroprocesora

ALU – Arithmetic Logic Unit
Jednostka arytmetyczno-logiczna

Operacje arytmetyczno-logiczne wykonywane są przez jednostkę arytmetyczno-logiczną – ALU.

Pamięć danych, programu i stos

Pamięć danych i programu jest 8-bitowa, aczkolwiek jest możliwe adresowanie słów wielobajtowych. Stos jest 16-bitowy.

Te trzy rodzaje pamięci w mikroprocesorach 8086 są adresowane za pomocą wspólnej magistrali adresowej.

Stos jest to obszar pamięci, do którego wpisujemy dane, bądź są ładowane adresy w wyniku wykorzystania podprogramów lub przerwań. Ze stosu można pobrać najpierw ostatnio wpisaną daną, czyli odwrotnie do kolejności wpisywania.

Stos

- Stos - liniowa struktura danych, znaczeniowo odpowiadająca nazwie: dane dokładane są na wierzch stosu, również z wierzchołka są ściągane (stosuje się też określenie LIFO (ang. Last In First Out), oddające tę samą zasadę).

Język maszynowy - asembler

Asembler jest językiem zorientowanym maszynowo tzn. :

- program dla określonego procesora nie może być wykonany przez inne procesory
- aby programować w tym języku należy znać architekturę procesora

Asembler – program tłumaczący mnemoniki na postać binarną

Aby łatwiej programować w języku maszynowym programy pisane są w postaci mnemoników (np. MOV DS,AX).

Następnie za pomocą asemblera (Masm.exe, Tasm.exe, itp.) pliki tekstowe *.asm tłumaczone są na kod binarny – tworzony jest plik *.obj.

Żeby uzyskać plik wykonywalny typu *.exe należy plik *.obj skonsolidować linkerem (Tlink.exe). Aby uzyskać plik wykonywalny typu *.com należy skonsolidować plik następująco tlink /t *.obj.

Pliki typu *.com

Poza tym program musi spełniać dwa warunki:

- dane, program źródłowy i stos nie mogą wykraczać poza segment tzn. 64 kB.
- program musi zaczynać się od offsetu 100h (w programie należy użyć dyrektywę org 100h)

Ogólna postać rozkazów asemblera 8086

pole_etykiety pole_operacji pole_argumentów pole_komentarza

START: MOV AX,DATA ; początek prog.

;wielkość liter nie ma znaczenia

Dyrektywy asemblera 8086

```
.MODEL .....  ; TINY, SMALL, MEDIUM, COMPACT,
              ; LARGE, HUGE – modele pamięci
.STACK ..... ; określenie wielkości stosu
.DATA         ; segment danych
        dana1 DB 8 ; deklaracje adresów w pamięci
                   ; DB – adresowanie bajt po bajcie
                   ; DW – adresowanie słowo po
                   ; słowie (16 bit)
                   ; DD – słowa 4 bajtowe
```

Dyrektywy asemblera 8086 c.d.

```
.CODE              ;segment kodu (programu)
Start:  ...........
        ...........
        ...........
        ........... ; komentarze
End
```

Dyrektywy asemblera 8086 c.d

```
.386            ;lista rozkazów procesora 386
.387            ;oraz koprocesora 387
CODE SEGMENT NAZWA_PROGRAMU
ASSUME CS:CODE, DS:CODE
ORG 100H                    ; offset 100h
START:
PUSH CS             ; na stos CS
POP DS              ;pobranie ze stosu do DS.
.....;
.....;
CODE ENDS
END START
```

Procedury i podprogramy

Nazwa_procedury **PROC NEAR** ; zakres adresowania
; **FAR**

.....

.....

RET ;rozkaz wyjścia z procedury

Nazwa_procedury ENDP ; koniec procedury

; wywołanie procedury

CALL Nazwa_procedury

Dyrektywy segmentowe

.ALPHA, ASSUME, .CODE, CODESEG, .CONST, CONST, .DATA, DATASEG, .DATA?, DOSSEG, .FARDATA, FARDATA, .FARDATA?, GROUP, .MODEL, MODEL, SEGMENT, .STACK, STACK, UDATASEG, UFARDATA

Symbole i wyrażenia asemblera

$, @code, @CodeSize, @CPU, @curseg, @data, @DataSize, ??date, @fardata, @fardata?, @FileName, ??filename, @Model, @Startup, ??time, ??version, @WordSize, (), *, +, -, ., /, :, [], AND, ALIGN, BYTE, BYTE PTR, CATSTR, .CONST, CONST, CODEPTR, .CREF, DATAPTR, DB, DD, DF, DP, DQ, DT, DUP, DWORD, DWORD PTR, EQ, EQU, EVEN, EVENDATA, EXTRN, FAR, FAR PTR, FWORD, FWORD PTR, GE, GLOBAL, GT, HIGH, INSTR, LARGE, LE, LENGTH, LOW, LT, MASK, MOD, NE, NEAR, NEAR PTR, NOT, OFFSET, OR, ORG, PTR, PWORD, PWORD PTR, QWORD, QWORD PTR, RECORD, SEG, SHL, SHORT, SHR, SIZE, SIZESTR, SMALL, SUBSTR, SYMTYPE, TBYTE, TBYTE PTR, THIS, .TYPE, TYPE, UNION, UNKNOWN, WIDTH, WORD, WORD PTR, XOR

Asemblacja warunkowa i sterowanie procesorem

.186, .286, .286C, .286P, .287, .386, .386C, .386P, .387, .8086, .8087, ELSE, ELSEIF, EMUL, END, ENDIF, .ERR, ERR, .ERR1, .ERR2, .ERRDEF, .ERRDIF, .ERRDIFI, .ERRE, .ERRIDN, .ERRIDNI, ERRIF, ERRIF1, ERRIF2, ERRIFB, ERRIFDEF, ERRIFDIF, ERRDIFI, ERRIFE, ERRIFIDN, ERRIFIDNI, ERRIFNB, ERRIFNDEF, .ERRNB, .ERRNDEF, .ERRNZ, JUMPS, IF, IF1, IF2, IFB, IFDEF, IFDIF, IFDIFI, IFE, IFDIN, IFDINI, IFNB, IFNDEF, INCLUDE, INCLUDELIB, MASM, MASM51, MULTERRS, NOEMUL, NOJUMPS, NOLOCALS, NOMASM51, NOSMART, ORG, P186, P286, P286N, P286P, P287, P386, P386N, P386P, P387, P8086, P8087, PNO87, QUIRKS, SMART

Rozkazy mikroprocesora 8086

Rozkazy przesłania danych:

MOV reg,reg ;np. MOV AX,BX
MOV mem,reg ;np. MOV [1234],AL
MOV reg,mem
MOV reg,stala ;np. MOV AH,#12h
MOV reg,seg ;np. MOV AX,DS
MOV mem,seg ;np. MOV [1234],DS
MOV seg,reg ;np. MOV DS,AX
MOV seg,mem

Rozkazy przesłania

MOVSB ; przesłanie bajtu z
; DS:SI do ES:DI

MOVSW ; przesłanie słowa

PUSH reg ; ładowanie na stos
; np. PUSH AX

PUSH mem

PUSH stala ; np. PUSH 12

PUSHF ; przesłanie znaczników na stos

LODSB ; ładowanie z DS:SI do AL

LODSW ; ładowanie z DS:SI do AX

POP reg ; pobranie ze stosu do rejestru

Rozkazy przesłania c.d.

POP mem

POPF ; pobranie ze stosu znaczników

STOSB ; prześlij bajt z AL do ES:DI

STOSW ; prześlij słowo z AX do ES:DI

IN acc,stala ; pobranie danej z portu do
; akumulatora, np. IN AX,3f8h

IN ACC,DX ;

OUT port,acc ; wyprowadzenie danej na
; port np. OUT 20h,AL

OUT DX,AX ;

LEA reg,mem ; ładowanie adresu efektywnego
; np. LEA DX,tablica ;

Rozkazy przesłania c.d.

LAHF ; pobranie do AH rejestru znaczników

SAHF ; przesłanie AH do rejestru znaczników

XCHG reg,reg ; zamiana danych
; miejscami

XCHG reg,mem

XCHG mem,reg

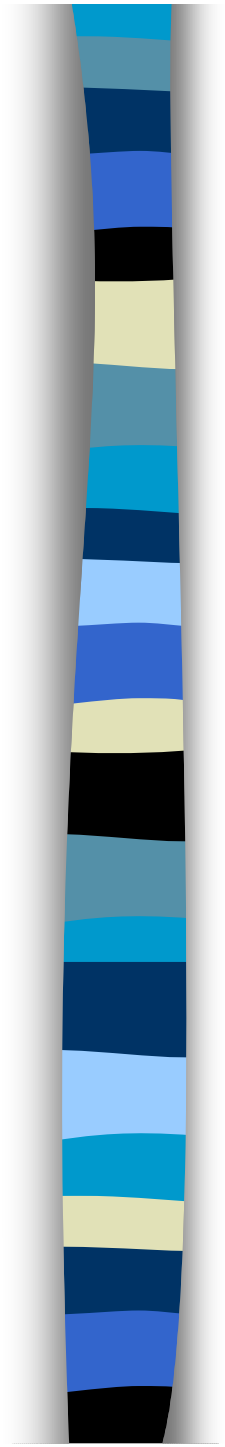
XLAT ; pobranie elementu z tablicy o
; adresie DS:BX do AL

Rozkazy przesłania c.d.

```
LDS   reg,mem   ; pobranie adresu z
                ; jednoczesnym
                ; załadowaniem rejestru DS
                ; i podanego rejestru
LES   reg,mem   ; pobranie adresu z
                ; jednoczesnym
                ; załadowaniem rejestru DS
                ; i podanego rejestru
```

Rozkazy arytmetyczne

```
ADD     reg,reg     ; dodanie arytmetyczne
                    ; zawartości dwóch
                    ; rejestrów
                    ; (8 lub 16 bitowo)
ADD     reg,mem     ;
ADD     mem,reg     ;
ADD     reg,stała   ;
ADD     mem,stała   ;
```



ADC reg,reg ; dodanie arytmetyczne
; zawartości dwóch
; rejestrów i znacznika
; przeniesienia (8 lub 16 bitowo)

ADC reg,mem ;

ADC mem,reg ;

ADC reg,stała ;

ADC mem,stała ;

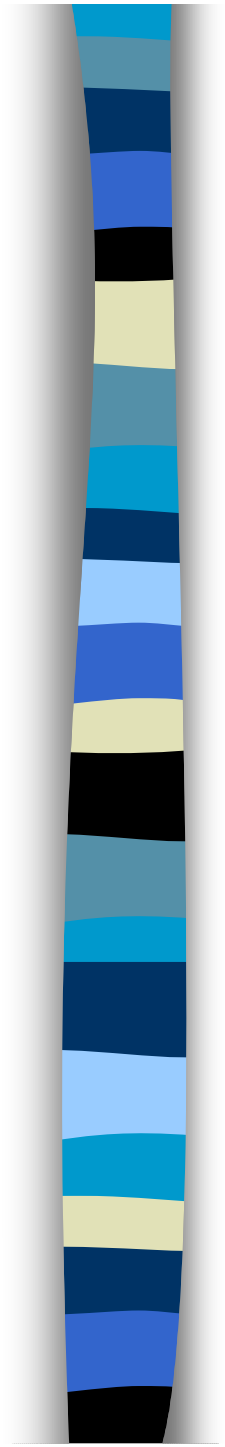
SUB reg,reg ; odejmowanie arytmetyczne
; zawartości dwóch rejestrów
; (8 lub 16 bitowo)

SUB reg,mem

SUB mem,reg

SUB reg,stała

SUB mem,stała



SBB reg,reg ; odejmowanie arytmetyczne
; zawartości dwóch rejestrów z
; pożyczką (bit Carry)
; (8 lub 16 bitowo)

SBB reg,mem

SBB mem,reg

SBB reg,stała

SBB mem,stała

INC reg ; inkrementacja rejestru (reg+1)

INC mem ; inkrementacja komórki w pamięci

NEG reg ; zmiana znaku liczby w rejestrze

NEG mem ; w pamięci

DEC reg ; dekrementacja rejestru (reg-1)

DEC mem ; dekrementacja komórki w pamięci

AAA ; korekta wyniku po dodaniu dwóch
; liczb w rozpakowanym kodzie
; BCD (generalnie 16-bitowych).

Przykład:

MOV BX,0207H

MOV AX,0405H

ADD AX,BX ← TERAZ AX=060CH

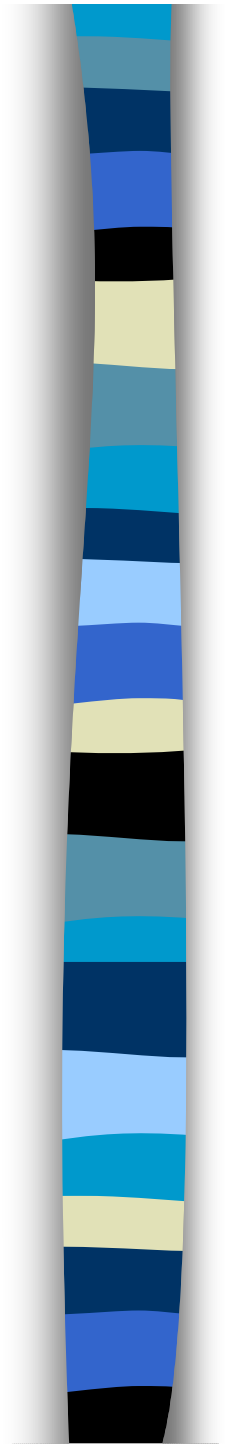
AAA ← TERAZ AX=0702H

AAS ; korekta wyniku po odjęciu dwóch
; liczb w rozpakowanym kodzie BCD
; (generalnie 16-bitowych)

Przykład:

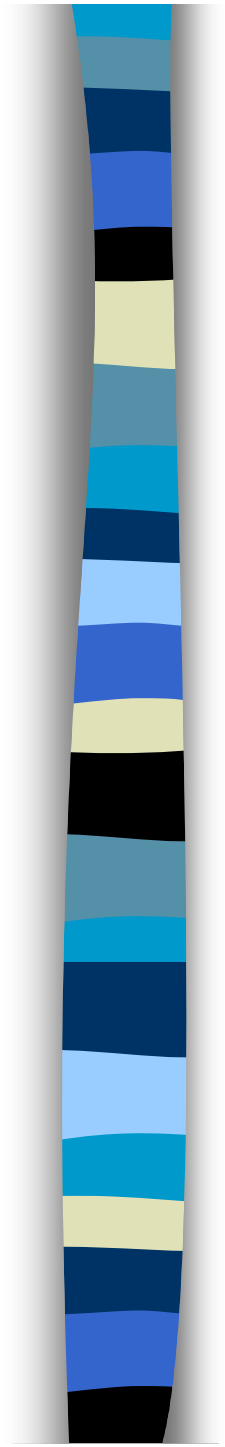
```
MOV     AX,0505H
MOV     BX,0207H
SUB     AX,BX       ← TERAZ AX=02FEH
AAS                 ← TERAZ AX=0108H
```

DAA, DAS - działają tak jak instrukcje AAA i AAS, tylko na argumentach w upakowanym kodzie BCD (generalnie 8-bitowych)



; **MUL** - mnożenie liczb bez znaku. Operacja
; 8-bitowa pobiera pierwszy argument z AL.,
; natomiast operacja 16-bitowa pobiera
; argument z AX. Wynik jest przekazywany w
; pierwszym przypadku do AX, natomiast w
; drugim przypadku do pary rejestrów AX i DX.

```
MUL reg         ; mnożenie przez rejestr
MUL mem         ; mnożenie przez zawartość
                ; komórki pamięci
```

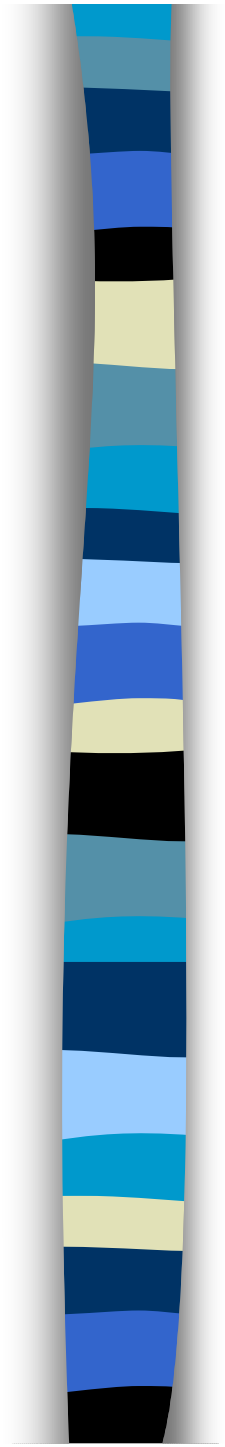


IMUL reg ; mnożenie ze znakiem przez
; rejestr – argumenty i wynik
; mnożenia (umiejscowienie)–
; tak samo jak w przypadku
; rozkazu **MUL**

IMUL mem ; mnożenie przez
; zawartość komórki pamięci

; DIV - dzielenie liczb bez znaku. Operacja dzielenia
; przez dzielnik 8-bitowy pobiera pierwszy argument z
; AX, natomiast operacja dzielenia przez dzielnik
; 16-bitowy pobiera argument z pary rejestrów AX i DX
; (DX jest starszym słowem wyniku).

DIV reg ; dzielenie przez daną z rejestru
DIV mem ; dzielenie przez daną z komórki pamięci

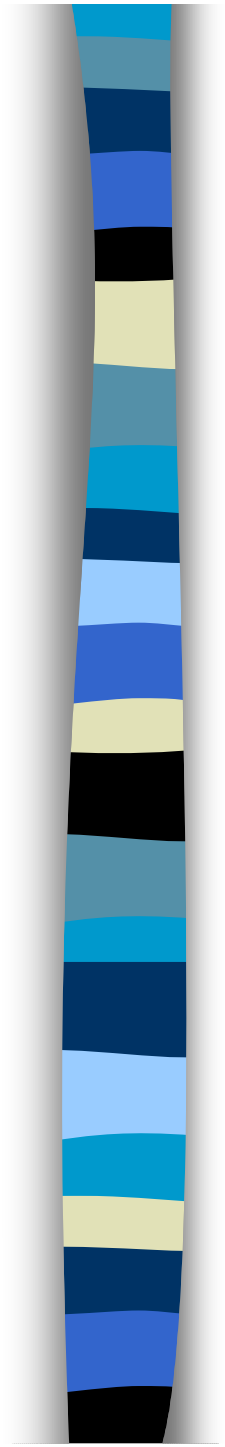


IDIV reg ; dzielenie liczb ze znakiem przez
; rejestr – argumenty i wynik
; mnożenia (umiejscowienie)–
; tak samo jak w przypadku
; rozkazu **DIV**

IDIV mem ; dzielenie przez
; zawartość komórki pamięci

AAD ; instrukcja konwersji liczby w rozpakowanym
; kodzie BCD, znajdującej się w rejestrach AH i
; AL., na liczbę binarną.

AAM ; instrukcja konwersji liczby w binarnej z rejestru
; AX na rozpakowany kod BCD, do rejestrów AH
; i AL.



Przykład:

```
MOV AX,52H   ;← DZIESIĘTNIE JEST TO 82
AAM                ;← AL=2, AH=8

CBW          ; przekształcenie bajtu w AL na słowo w
             ; AX z uwzględnieniem znaku liczby
```

Przykład:

```
MOV  AL,80H      ;← DZIESIĘTNIE JEST TO -128
CBW              ;← AX=FF80H

CWD          ; przekształcenie słowa w AX na
             ; podwójne słowo (zapisane w parze
             ; rejestrów AX i DX).
```

Przykład:

```
MOV  AX,8000H   ;← DZIESIĘTNIE JEST TO -32768
CWD                ;← AX=8000H, DX=FFFFH,
          ;DX JEST STARSZYM SŁOWEM WYNIKU
```

Rozkazy logiczne, przesunięcia i obroty

; **OR** - suma logiczna argumentów

OR reg,reg ; suma logiczna zawartości
; dwóch rejestrów

OR reg,mem

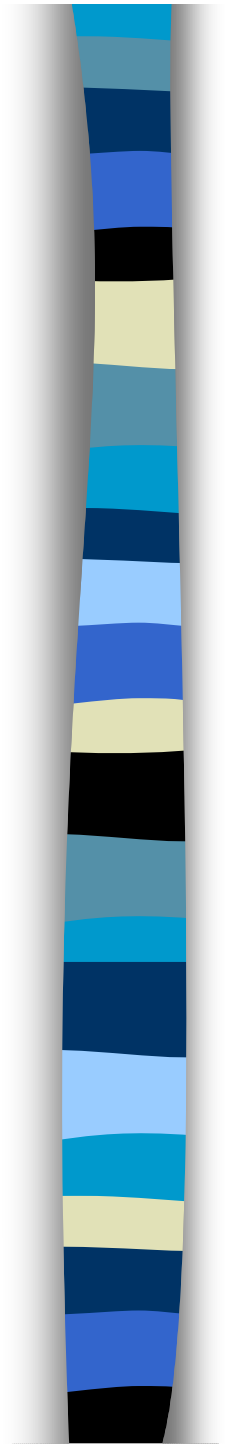
OR mem,reg

OR reg,stała

OR mem,stała

; **XOR** różnica symetryczna argumentów.

; Składnia jak dla instrukcji OR.



; **AND** - iloczyn logiczny argumentów.

AND reg,reg ; iloczyn logiczny zawartości
; dwóch rejestrów

AND reg,mem

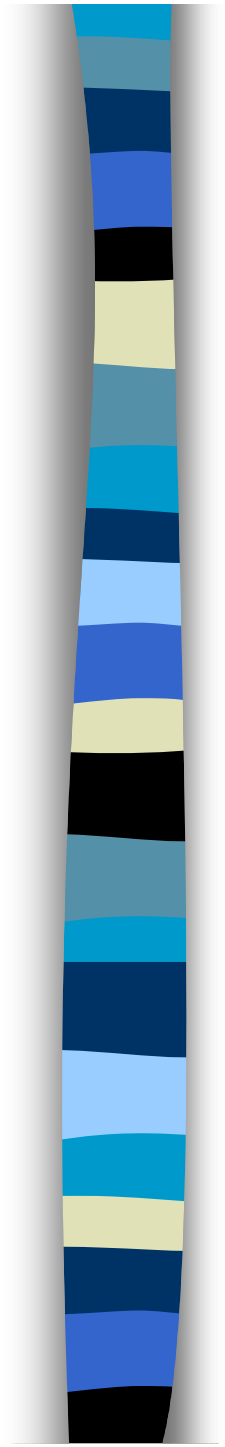
AND mem,reg

AND reg,stala

AND mem,stała

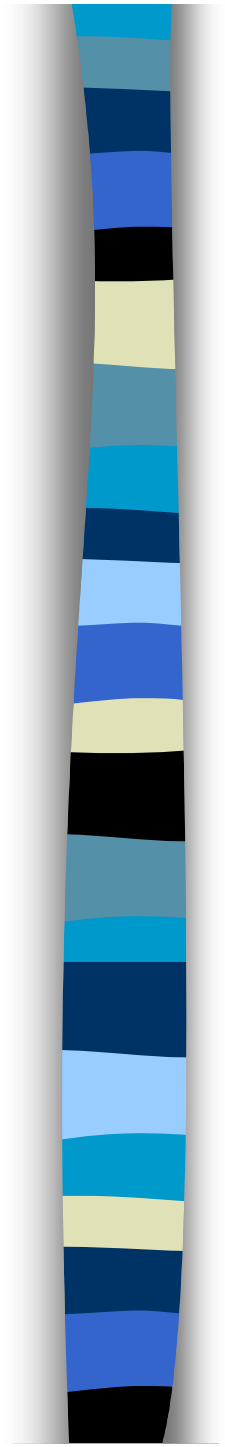
; **TEST** - testowanie wybranych bitów

; argumentów. Instrukcja ta wykonuje logiczną operację AND na swoich argumentach. Wynik nie jest nigdzie zapamiętywany, ale na jego podstawie ustawiane są znaczniki.

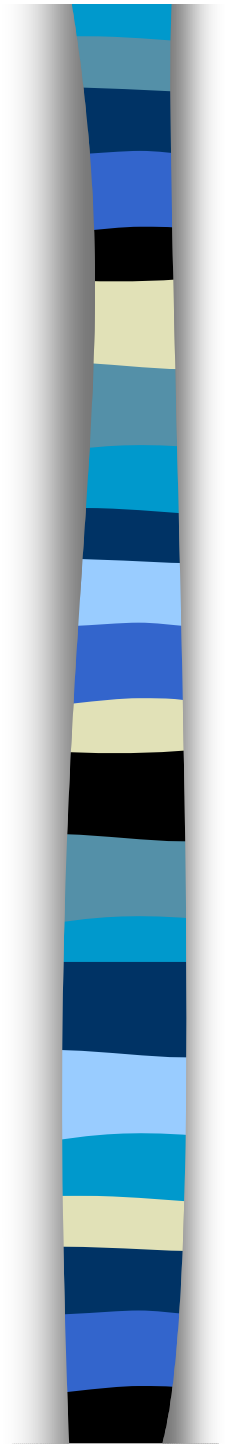


TEST reg,reg ; testowanie zawartości
; dwóch rejestrów
TEST reg,mem
TEST mem,reg
TEST reg,stała
TEST mem,stała

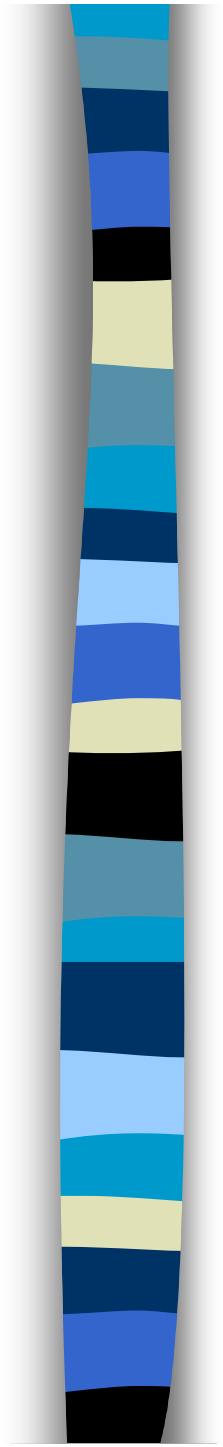
; **NOT** - negacja każdego bitu liczby.
NOT reg ; negacja danej w rejestrze
NOT mem ; negacja danej w pamięci



; **SHL** - przesunięcie logiczne w lewo,
; najstarszy bit przechodzi do CF,
; a najmłodszy bit jest uzupełniany zerem
SHL mem,1 ; przesunięcie logiczne komórki
; pamięci o 1 w lewo
SHL reg,1
SHL mem,cl ; od 80286
SHL reg,cl ; przesunięcie logiczne
; komórki rejestru o zawartość
; CL w lewo
; **SHR** - przesunięcie logiczne w prawo,
; najmłodszy bit przechodzi do CF,
; a najstarszy bit jest uzupełniany zerem
; Składnia i argumenty jak dla SHL.



```
; SAL - przesunięcie arytmetyczne w lewo = SHL.
; Operacja równoważna z mnożeniem przez 2.
SAL   mem,1
SAL   reg,1
SAL   mem,CL        ; przesunięcie arytm. komórki
                    ; pamięci o zawartość CL w lewo
SAL   reg,CL        ; od 80286

; SAR - przesunięcie arytmetyczne w prawo. Operacja
; równoważna z dzieleniem przez 2. Najbardziej
; znaczący bit posiada taka wartość jak przed
; przesunięciem – znak liczby
SAR   mem,1
SAR   reg,1
SAR   mem,Cl        ; od 80286
SAR   reg,cl
```



; **ROL** - obrót w lewo. Składnia jak dla rozkazów
; SHL, SHR, SAL, SHR, przy czym bit
; najbardziej znaczący jest przesuwany zarówno do CF
; jak i do bitu najmłodszego

; **ROR** - obrót w prawo. Składnia jak dla rozkazów SHL,
; SHR, SAL, SHR, przy czym bit najmłodszy jest
; przesuwany zarówno do CF jak i do bitu najstarszego

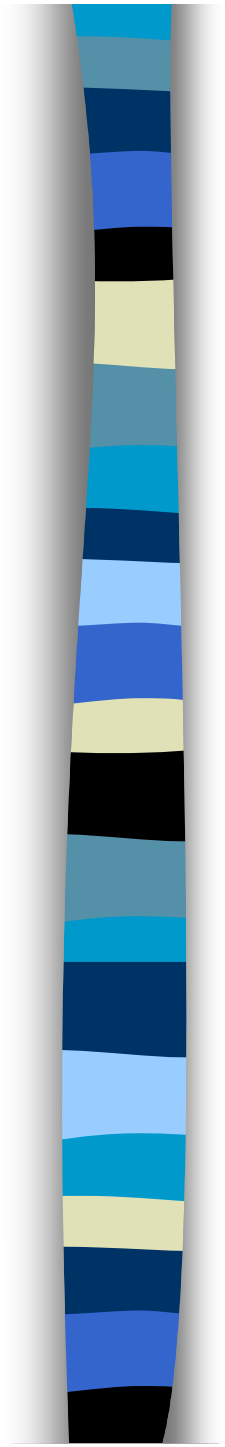
; **RCL** - obrót w lewo z uwzględnieniem bitu Carry (CF).
; Najbardziej znaczący bit jest przesuwany do CF, a CF
; do bitu najmłodszego
; Składnia jak dla rozkazów SHL, SHR, SAL, SHR.

; **RCR** - obrót w prawo z uwzględnieniem bitu Carry.
; Najmniej znaczący bit jest przesuwany do CF, a CF
; do bitu najstarszego
; Składnia jak dla rozkazów SHL, SHR, SAL, SHR.

Rozkazy skoków warunkowych i bezwarunkowych (pętle)

; **JE/JZ** - skok, gdy równy/równy zero
; **JL/JNGE** - skok, gdy mniejszy/nie równy
; lub nie większy
; **JLE/JNG** - skok, gdy mniejszy lub równy/nie
; większy
; **JB/JNAE** - skok, gdy mniejszy/nie większy lub
; nie równy
; **JBE/JNA** - skok, gdy mniejszy lub równy/nie
; większy
; **JP/JPE** - skok, gdy parzystość/parzystość
; parzysta

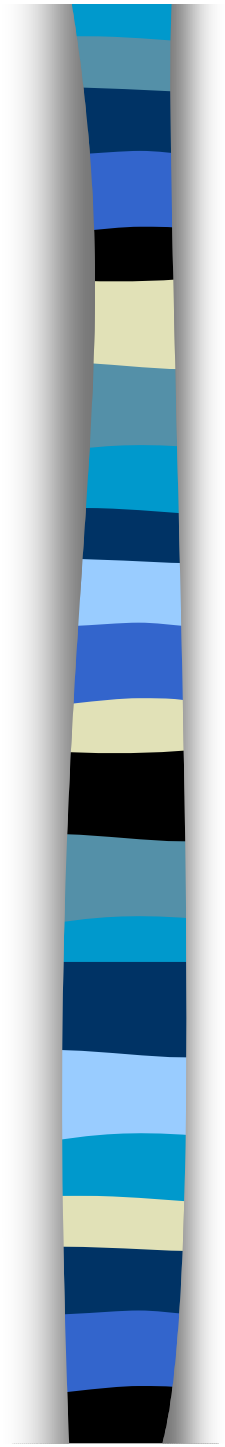
; **JO** - skok, gdy nadmiar (przepełnienie)
; **JS** - skok, gdy znak ujemny
; **JNE/JNZ** - skok, gdy nie równy/nie zero
; **JNL/JGE** - skok, gdy nie
; mniejszy/większy lub równy
; **JNLE/JG** - skok, gdy nie mniejszy lub równy/większy
; **JNB/JAE** - skok, gdy nie
; mniejszy/większy lub równy
; **JNBE/JA** - skok, gdy nie mniejszy lub
; równy/większy
; **JNP/JPO** - skok, gdy
; nieparzystość/parzystość nieparzysta



; **JNO** - skok, gdy nie ma nadmiaru
; **JNS** - skok, gdy znak dodatni
; **JC** - skok, gdy CF=1
; **JNC** - skok, gdy CF=0
CALL nazwa_podprogramu ; skok do
;podprogramu
; (wywołanie procedury)
; **JMP** - skok bezwarunkowy
RET ; powrót z podprogramu
; (procedury)

; **LOOP** - zmniejsza CX o 1 i wykonuje
; skok, gdy CX ≠ 0.

LOOP nazwa_etykiety ; skok do
; etykiety, gdy CX ≠ 0



; **LOOPZ/ LOOPE** - jak LOOP, tylko może wykonać
; dodatkowo skok, gdy bit zera (ZF) jest ustawiony.

; **LOOPNZ/LOOPNE** - jak LOOP, tylko może wykonać
; dodatkowo skok, gdy bit zera nie jest
; ustawiony.

; **JCXZ** - rozkaz skoku warunkowego, który jest
; wykonywany, gdy CX jest równy zero.

; Należy pamiętać, aby w pętlach uważać z modyfikacją
; rejestru CX, gdyż jest on licznikiem pętli. Poza tym
; zasięg tych skoków to -128¸+127 bajtów. Etykieta, do
; której wykonywany jest skok powinna się znajdować
; na ogół powyżej instrukcji kontroli iteracji.

; **Przykład:**

MOV CX,10

Tu:

.....

..... ; ciąg instrukcji w pętli

.....

LOOP Tu

Powyższa pętla zostanie wykonana 10 razy.

Rozkazy przerwań programowych

; **INT** - przerwanie programowe

; **INT** numer_przerwania ; wywołanie
;przerwania o podanym numerze

; **INTO** - wygenerowanie przerwania nr 4 w

; przypadku, gdy znacznik nadmiaru jest

; ustawiony.

; **RET** - powrót z podprogramu.

; **IRET** - powrót z podprogramu obsługi

; przerwania

Rozkazy łańcuchowe

; Rozkazy łańcuchowe operują na ciągach
; danych bajtowych lub słowowych. Ciągi te
; mogą mieć długość do 128 KB. Po wykonaniu
; rozkazu łańcuchowego rejestry SI i DI są
; automatycznie zmieniane, aby przygotować je
; do zaadresowania następnego elementu
; łańcuchowego. Gdy flaga DF (direct flag) = 1,
; zawartość tych rejestrów jest zwiększana, gdy
; zero jest zmniejszana.

Rozkazy łańcuchowe c.d.

; Gdy elementy łańcucha są bajtami wartość SI
; i DI jest zmniejszana o 1, a gdy słowami - o 2.
; Po zrealizowaniu rozkazu łańcuchowego
; wartość CX jest zmniejszana o 1. Przed
; rozkazem łańcuchowym należy wpisać
; odpowiednią liczbę do CX określającą ilość
; powtórzeń rozkazu łańcuchowego.
; Powtórzenia wykonuje się za pomocą
; rozkazów **REP...** . Gdy CX=0 ; to zostaje
; przerwane powtarzanie rozkazów
; łańcuchowych i program przechodzi do
; wykonania kolejnego rozkazu.

Powtórzenia rozkazów łańcuchowych

; **REP** – powtórz

; **REPE** – powtórz dopóki równe

; **REPZ** – powtórz dopóki zero

; **REPNZ** – powtórz dopóki nie zero

; **REPNE** – powtórz dopóki nie równe

; **Przykład:**

MOV CX,10 ;

REP STOSW ; 10 razy z AX do ES:DI

Rozkazy łańcuchowe

MOVSB ; przesłanie bajtu z
; DS:SI do ES:DI

MOVSW ; przesłanie słowa

LODSB ; ładowanie z DS:SI do AL

LODSW ; ładowanie z DS:SI do AX

STOSB ; prześlij bajt z AL do ES:DI

STOSW ; prześlij słowo z AX do ES:DI

CMPS [seg:]argument,argument ; porównanie
; bajtów

CMPSB ; bajtów z DS:SI i ES:DI

CMPSW ; słów

Rozkazy łańcuchowe

SCAS [seg:]argument ; porównanie bajtu
; lub słowa wskazywanego przez ES:DI
; z daną, która znajduje się w
; akumulatorze (AL lub AX) – po
; odejmowaniu ustawiane są znaczniki

SCASB ; porównanie bajtów
SCASW ; porównanie słów

Rozkazy sterowania procesem

CLC ; zerowanie znacznika CF
CMC ; zmiana stanu CF na przeciwny
STC ; ustawienie znacznika CF
CLD ; zerowanie znacznika DF (direct flag)
STD ; ustawienie znacznika DF (direct flag)
CLI ; zerowanie znacznika IF (interrrupt flag)
STI ; ustawienie znacznika IF (interrupt flag)
HLT ; zatrzymanie pracy procesora
WAIT ; wprowadzenie procesora w stan
; oczekiwania
NOP ; nic nie rób
LOCK ; blokada dostępu do magistrali

Rodzaje adresowania 8086

- Natychmiastowe

  MOV AX,1234H
- Bezpośrednie

  MOV AX,BX

  MOV AH,[1234]
- Pośrednie
  - przez rejestr bazowy

  MOV CX,1234H

  MOV BX,8

  MOV BP,2

  MOV [BP],CX ; SS:BP

  MOV AX,[BX] ; DS:BX

Rodzaje adresowania 8086 c.d.

- przez rejestr bazowy i przemieszczenie

```
MOV CX,1234H
MOV BX,8
MOV BP,2
MOV [BP+1],CX    ; SS:BP+1
MOV AX,[BX+2]    ; DS:BX+2
```

- przez rejestr indeksowy

```
MOV DI,3
MOV SI,7
MOV AX,[SI]      ; DS:SI
MOV [DI],BX      ; ES:DI
```

Rodzaje adresowania 8086 c.d.

- przez rejestr indeksowy i przemieszczenie

```
MOV DI,3
MOV SI,7
MOV AX,[SI+5]      ; DS:SI+5
MOV [DI+3],BX      ; ES:DI+3
```

- przez rejestr bazowy, indeksowy i przemieszczenie

```
MOV SI,3
MOV BX,2
MOV BP,1
MOV DI,4
MOV DX,45
MOV [BP+DI+4],DX     ; SS:BP+DI+4
MOV AX,[BX+SI+1]     ; DS:BX+SI+1
MOV CX,[DI+BP+1]     ; SS:DI+BP+1
```

Rodzaje adresowania 8086 c.d.

- Łańcuchowe

```
MOV AX,0H
MOV DS,AX
MOV SI,0H
MOV AX,9000H
MOV ES,AX
MOV DI,0H
MOV CX,1024D
CLD                 ; jeśli DF = 0 to inkrementacja
PETLA: MOVSB ; przesłanie z DS:SI do ES:DI
LOOP PETLA     ;
```

Adresowanie układów wejścia, wyjścia

IN acc,stala ; pobranie danej z portu do
; akumulatora, np. IN AX,3f8h

IN acc,DX ;

OUT port,acc ; wyprowadzenie danej na
; port np. OUT 20h,AL

OUT DX,AX ;

Makroinstrukcje a podprogramy

Podprogram to sekwencja rozkazów, która raz umieszczona w pamięci może być wykonywana wielokrotnie w różnych miejscach programu głównego.

Dla skrócenia zapisów programów w języku asembler wprowadzono makroinstrukcje, czyli symboliczne oznaczenia odpowiadające nie pojedynczym rozkazom, lecz sekwencjom rozkazów. Podczas tłumaczenia programu, makroinstrukcje są tłumaczone na odpowiadające im ciągi rozkazów.

Nazwa_macra **MACRO**

.....

..... ; tekst programu

ENDM

Makroinstrukcje - przykłady

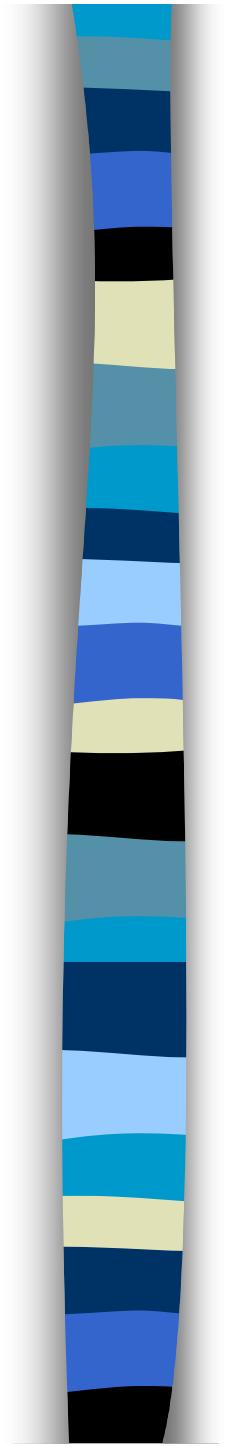
```
WYSWIETL_ZNAK MACRO
      MOV AH,02H
      INT 21H
   ENDM

   GOTO_XY MACRO x,y
    MOV AH,2
    MOV DL,X
    MOV BH,0
    MOV DH,Y
    INT 10H
ENDM
```

Makroinstrukcje - przykłady

```
; PRZYKŁAD
START:
        MOV AX,DATA
        MOV DS,AX
        GOTO_XY 2,3
        MOV DL,'0'
        WYSWIETL_ZNAK
KONIEC:
        MOV AH,4CH
        INT 10H
END START
```

PRZYKŁAD PROGRAMU

```
.MODEL SMALL
.DATA
    pusta_linia DB 13,10,'$'
    iloraz DB ?
    reszta DB ?
    dzielnik DB 10
.CODE
Start:
        MOV AX,@DATA
        MOV DS,AX
        XOR AX,AX                ; zerowanie AX
        XOR DX,DX
        XOR CX,CX
        MOV BX,12d     ; ilość liczb do zsumowania
Petla:
        INC CX
        ADD AX,CX
        PUSH AX
```

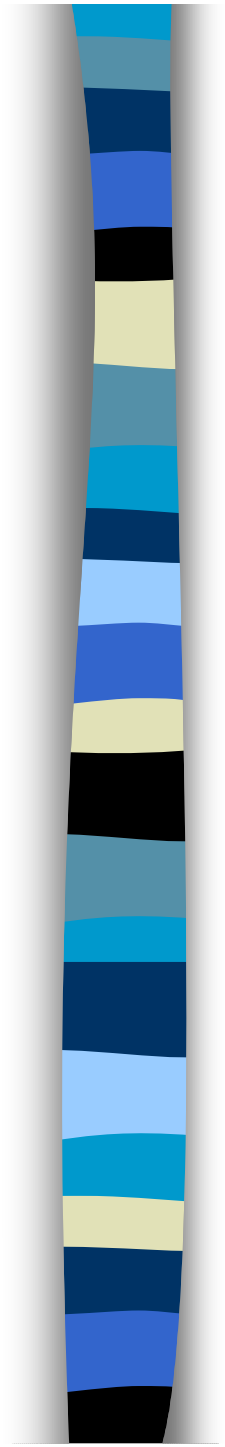


```
        DIV dzielnik       ; AX / dzielnik-> AL, reszta ->AH
        MOV [iloraz],AL
        MOV [reszta],AH
        CALL ILORAZY
        CALL RESZTY
        CALL LINIA
        POP AX
        CMP CX,BX
        JNE Petla
        JMP Koniec
;************************************PROCEDCURY**************
ILORAZY PROC NEAR
        MOV AH,2
        MOV DL,[Iloraz]
        ADD DL,48
        INT 21H
        RET
        ILORAZY ENDP
```

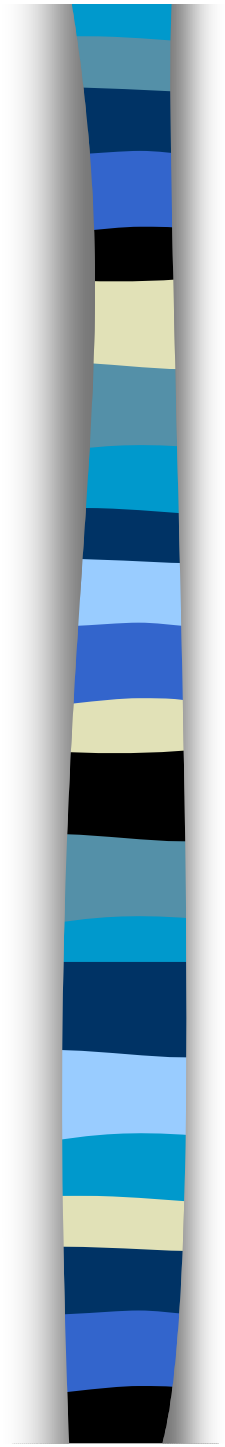
```
RESZTY PROC NEAR
        MOV AH,2
        MOV DL,[reszta]
        ADD DL,48
        INT 21H
        RET
        RESZTY ENDP
;*************************************************************
LINIA PROC NEAR
        MOV AH,09
        MOV DX,OFFSET pusta_linia
        INT 21H
        RET
        LINIA ENDP
;*************************************************************
Koniec:
        MOV AH,4CH
        INT 21H
END START
```

Program zamiany Dec -> Bin

```
.model small
.stack 512
.data
      RESZTY DB 16 DUP(?) ; rezerwowanie 16 bajtów
.code
start:

;ustawienie początku danych
    mov ax,@data
    mov ds,ax
;kursor na początku ekranu
    mov dh,0
    mov dl,0
    mov bh,0  ;strona zero
    mov ah,02h
    int 10h
```



```
        mov ax,57291        ; liczba do zamiany
        xor bx,bx           ;zerowanie BX
        xor dx,dx
        xor cx,cx
petla1:
        inc cl
        push ax
        shr ax,1
        pop dx
        sub dx,ax
        sub dx,ax
        mov byte ptr[reszty+bx],dl ;prześlij z DL pod
        mov bl,cl       ; adres symboliczny określonego
        cmp ax,0        ; etykietą reszty + BX
        jne petla1
```



```
petla2:
        dec bl
        mov ah,2
        mov dl,ds:[reszty+bx]
        add dl,48
        INT 21H
        LOOP petla2
KONIEC:
        mov ah,4ch
        int 21h
end
```