255 słowo

1. ARCHITEKTURA PROCESORÓW INTEL X86

1.1. Historia

x86 to rodzina architektur (modelów programowych) procesorów firmy Intel, należących do kategorii CISC, stosowana w komputerach PC, zapoczątkowana przez i wstecznie zgodna z 16-bitowym procesorem 8086, który z kolei wywodził się z 8-bitowego układu 8085. Nazwa architektury wywodzi się od nazw pierwszych modeli z tej rodziny, których numery kończyły się liczbą 86. Wyróżnia się:

- x86 - procesory od 8086 (rok 1978) do 286, które były układami o architekturze 16-bitowej.
- x86-32 (IA-32) - procesor 80386 (rok 1985), w którym dokonano rozszerzenia słowa do 32 bitów, unikając jednak konieczności natychmiastowej wymiany wszystkich komputerów, poprzez zachowanie trybów zgodności z poprzednimi rozwiązaniami.
- x86-64 (AMD64) - procesory 64-bitowe. Architekturę (model programowy) takich procesorów, ze względu na wciąż zachowywaną wsteczną kompatybilność z pierwowzorami o architekturze x86, oznacza się symbolem x86-64. Rozwiązanie to zostało wprowadzone jednak przez firmę AMD, a dopiero później zaadoptowane przez Intela jako EM64T.

1.2. Organizacja pamięci (procesory 8086/8088)

- Procesory 8086 posiadają magistralę adresową składającą się z 20 linii adresowych, co oznacza że w trybie bezpośrednim mogą zaadresować ponad milion (2^{20} czyli 1 048 576) komórek pamięci.

- Pojedyncza komórka pamięci ma wielkość 1 bajta więc procesory te mogą zaadresować 1MB pamięci (2^{20} bajtów).
- Wszystkie komórki pamięci posiadają swoje adresy fizyczne. Najmniejszy adres to 0 a największy 1 048 575 (11111111111111111111b czyli FFFFFh).

pamięć (1 komórka = 8 bitów = 1 bajt)	*adres fizyczny (hex)*	*adres fizyczny (dec)*
	- FFFFFh	1048575
	- FFFFEh	1048574
	- FFFFDh	1048573
	- FFFFCh	1048572
...	...	...
	- 00006h	6
	- 00005h	5
	- 00004h	4
	- 00003h	3
	- 00002h	2
	- 00001h	1
	- 00000h	0

1.3. Jednostki pamięci

- Pojedyncza komórka pamięci to wielkość zbyt mała, żeby można było efektywnie wykonywać programy oraz zarządzać danymi i dostępną pamięcią (na jednym bajcie można zapisać wartości od 0 do 255 (FFh)).
- Wykorzystywane są jeszcze większe jednostki pamięci – takie jak: słowo (w tej 16-bitowej architekturze ma wielkość dwóch bajtów), podwójne słowo, poczwórne słowo, paragraf, strona, segment.

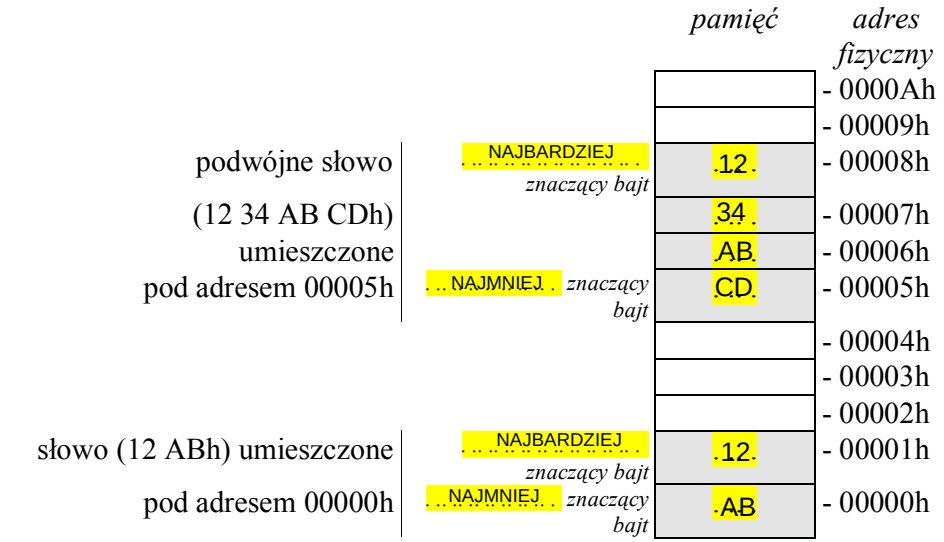
Jednostka	Nazwa (w asemblerze)	Liczba bajtów	Przykład
bajt	BYTE	1	1Ah
słowo	WORD	2	12 ABh
podwójne słowo	DWORD	4	12 34 AB CDh
poczwórne słowo	QWORD	8	
paragraf	PARA	16	
strona	PAGE	256	
segment	SEGMENT	65536	

- Część spośród jednostek pamięci przeznaczona jest głównie do PRZECHOWYWANIA danych a część do ZARZĄDZANIA pamięcią (ADRESACJI tych danych oraz rozkazów wchodzących w skład programu komputerowego).

1.4. Umieszczanie danych w pamięci

- Kiedy w pamięci należy umieścić daną WIĘKSZĄ niż jeden bajt (np. słowo czy podwójne słowo), problemem staje się KOLEJNOŚĆ umieszczania w pamięci poszczególnych jej bajtów.

- W architekturze x86 obowiązuje zasada umieszczania danych w pamięci nazywana LITTLE ENDIAN – zgodnie z nią – NAJMNIEJ znaczący bajt (ang. *low-order byte*) umieszczany jest pod adresem wskazanym jako adres danej, a kolejne bajty (bardziej znaczące) pod następnymi, STARSZYMI adresami.



- Forma zapisu little endian jest wykorzystywana przez procesory Intel x86, AMD64, DEC VAX.

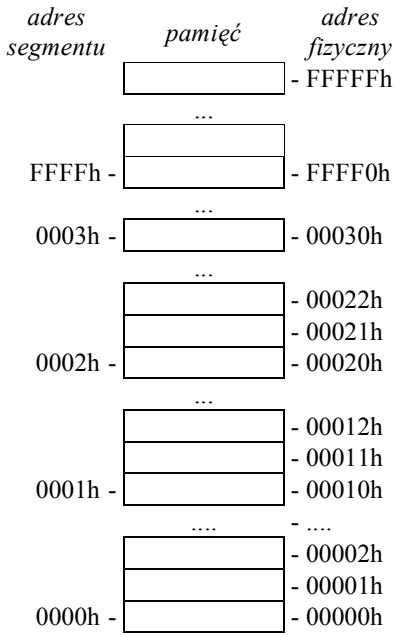
Uwaga: Istnieje „odwrotna" forma zapisu tzw. BIG ENDIAN, w której najbardziej znaczący bajt umieszczony jest jako pierwszy. Jest ona wykorzystywana takie procesory, jak np. SPARC, Motorola 68000, PowerPC 970, IBM System/360.
Ponadto istnieją także procesory, w których można przełączyć tryb kolejności bajtów, należą do nich na przykład PowerPC (do serii PowerPC G4), SPARC.

Etymologia nazw

Angielskie nazwy „big endian" i „little endian" pochodzą z książki Jonathana Swifta „Podróże Guliwera" i odnoszą się do mieszkańców Liliputu, których spór o to, czy ugotowane jajko należy tłuc od grubego (tępego), czy od cienkiego (ostrego) końca, doprowadził do podziału na dwa stronnictwa toczące ze sobą niekończące się, choć bezsensowne dysputy i wojny.

1.5. Segmenty pamięci

- Spośród wymienionych jednostek pamięci, na szczególną uwagę zasługuje PARAGRAF i SEGMENT.
- *Paragraf* to KOLEJNO 16 bajtów (komórek pamięci) poczynając od początku obszaru pamięci (adresu 0h).
- Komórka pamięci, w której zaczyna się paragraf nosi miano GRANICA PARAGRAFU i ma adres fizyczny podzielny przez 16 (10h) – np. komórki o adresach: 0, 16, 32, 48, 64 itd. (czyli w systemie szesnastkowym: 0h, 10h, 20h, 30h, 40h itd.).
- Na granicy każdego paragrafu zaczyna się SEGMENT.
- Segment składa się (zazwyczaj – niektóre są mniejsze) z 65536 komórek pamięci, a więc ma wielkość 65kB.
- Wielkość segmentu wynika z wielkości REJESTRÓW, które są 16 bitowe (2^{16} = 65 536). Uwzględniając właściwości paragrafów, należy zauważyć, że:
- Segmentów w 1MB pamięci jest 65536.



1.6. Adres logiczny a adres fizyczny

- Afizyczne, chociaż wydaje się bardzo proste i wygodne w użyciu, w praktyce są zastępowane przez ADRESY LOGICZNE.
- Adresacja logiczna jest możliwa dzięki podziałowi pamięci na segmenty (tzw. SEGMENTACJI *pamięci*).
- Segment to obszar pamięci, do której procesor w danej chwili może mieć DOSTĘP.
- Każda komórka w segmencie jest ponumerowana – numer komórki w segmencie to tzw. OFFSET. (PRZESUNIĘCIE), czyli offset jest to adres komórki (liczony od ZERA) w obrębie danego segmentu.

- Segment i offset . .. ADRES LOGICZNY .. . w postaci:

  . ..ADRES SEGMENTU.. . : . OFFSET .

- Przekształcenie adresu logicznego na adres fizyczny wygląda następująco:

  adres fizyczny komórki = (adres segmentu) * . 10h . + . OFFSET . .

- Ta sama komórka pamięci może mieć . .. WIELE . adresów logicznych – wynika to z faktu, że segmenty na siebie nachodzą.

Przykład

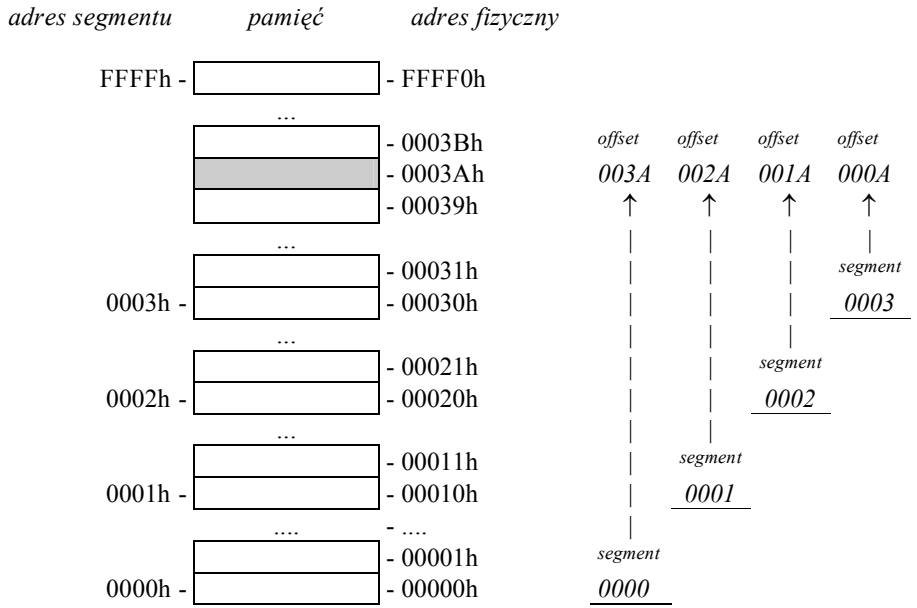
Komórka o adresie fizycznym 0003Ah posiada cztery adresy logiczne:

0000h: 003Ah,

0001h:002Ah,

0002h:001Ah,

0003h:000Ah.



0000h:003Ah = 0hx10h + 3Ah = 3Ah
0001h:002Ah = 1hx10h + 2Ah = 10h + 2Ah = 3Ah
0002h:001Ah = ...
0003h:000Ah = ...

Pytanie

Jak obliczyć dwa z możliwych adresów logicznych komórki o adresie fizycznym 09AB12h?

Pytanie

Ile maksymalnie adresów logicznych (w postaci segment:offset) może mieć komórka?

Inaczej pytanie to brzmi: *Do ilu maksymalnie segmentów może należeć jedna komórka?*

1.7. Rejestry

1.7.1. Podział rejestrów

Rejestr - jest to komórka pamięci dostępna bezpośrednio dla procesora. Jest ona elementem procesora.

rejestry ogólnego przeznaczenia

AX		akumulator (*accumulator*)
AH	AL	
BX		rejestr bazowy (base *register*)
BH	BL	
CX		rejestr zliczający (counter *register*)
CH	CL	
DX		rejestr danych (data *register*)
DH	DL	

rejestry wskaźnikowe i indeksowe

SP	wskaźnik stosu (stack *pointer*)
BP	wskaźnik bazy (base *pointer*)
SI	indeks źródła (source *index*)
DI	indeks przeznaczenia (destination *index*)

rejestry segmentowe

CS	segment kodu (*code segment*)
DS	segment danych (*data segment*)
SS	segment stosu (*stack segment*)
ES	segment dodatkowy (extra *segment*)

wskaźnik rozkazów

IP	wskaźnik rozkazów (instruction *pointer*)

znaczniki

Flags	rejestr znaczników (*flags pointer*)

1.7.2. Rejestry ogólnego przeznaczenia

Rejestry te wykorzystywane są głównie:

AX - akumulator - do operacji arytmetycznych i logicznych.

BX - rejestr bazowy - do adresowania pamięci.

CX - rejestr zliczający - jako licznik.

DX - rejestr danych - przy operacjach dzielenia i mnożenia oraz przy wysyłaniu i odbieraniu danych do i z portów.

Uwaga: Rejestry AX, BX, CX, DX (16 bitowe) można traktować jako pary rejestrów 8 bitowych:

AX = AH - AL

BX = BH - BL

CX = CH - CL

DX = DH - DL

1.7.3. Rejestry wskaźnikowe i indeksowe

SP - wskaźnik stosu - przechowuje offset do stosu. Wykorzystywany przy standardowych operacjach odczytywania i zapisywania danych na stos.

BP - wskaźnik bazy - służy do adresowania pamięci. Wykorzystywany przy niestandardowych operacjach zapisu i odczytu stosu.

SI - indeks źródła - wskazuje obszar, z którego pobierane są dane (czyli zawiera offset z segmentu danych).

DI - indeks przeznaczenia - wskazuje obszar, do którego wysyłane są dane (czyli zawiera offset z segmentu danych).

1.7.4. Rejestry segmentowe

CS - segment kodu - zawiera adres segmentu, w którym znajdują się aktualnie wykorzystywane rozkazy.

DS - segment danych - zawiera adres segmentu, w którym znajdują się dane (zmienne programu).

SS - segment stosu - zawiera adres segmentu stosu.

ES - segment dodatkowy - zawiera adres segmentu dodatkowego służącego najczęściej wymianie danych.

1.7.5. Wskaźnik rozkazów

IP - zawiera offset aktualnie wykonywanej instrukcji.

1.7.6. Rejestr znaczników

- Procesory 8086 mają 9 znaczników (flag).
- Samo przeznaczenie flag jest standardowe, tzn. flagi
  - albo informują o tym co zaszło w wyniku wykonywanej operacji,
  - albo wpływają na przebieg (sposób) jej wykonywania.
- Wartość 1 - na określonej pozycji oznacza, że znacznik jest ustawiony, wartość 0 - że nie jest ustawiony.

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
					OF	**DF**	**IF**	**TF**	**SF**	**ZF**	**0**	**AF**	**1**	**PF**	**1**	**CF**
1					**OV**	**DN**	**EI**		**NG**	**ZR**		**AC**		**PE**		**CY**
0					**NV**	**UP**	**DI**		**PL**	**NZ**		**NA**		**PO**		**NC**

CF (*carry flag*) - znacznik przeniesienia

Przyjmuje wartość 1 gdy na skutek wykonanego działania nastąpiło przeniesienie bitu z najbardziej znaczącego na zewnątrz lub pożyczka z zewnątrz do bitu najbardziej znaczącego (np. przy odejmowaniu).

Przykład

```
   1010 1110                0000 0001
 + 0111 0100              - 0000 0011
------------             -----------
 1 0010 0010   (CF = 1)     1111 1110    (CF = 1)
```

PF (*parity flag*) - znacznik parzystości

Przyjmuje wartość 1 gdy w wyniku wykonywanego działania liczba bitów o wartości 1 w mniej znaczącym bajcie wyniku jest parzysta.

Przykład

```
   0010 1100
 + 1011 0001
------------
   1101 1101
```

PF = 1.

AF (*auxiliary carry flag*) - znacznik przeniesienia pomocniczego

Przyjmuje wartość 1 gdy nastąpiło przeniesienie z bitu 3 na 4 lub pożyczka z bitu 4 na 3.

Uwaga: bity są numerowane od 0

ZF (*zero flag*) - znacznik zera

Przyjmuje wartość 1 gdy wynik ostatniej operacji arytmetycznej wynosi 0.

SF (*sign flag*) - znacznik znaku

Przyjmuje wartość 1 gdy najbardziej znaczący bit w otrzymanym wyniku jest równy 1.

TF (*trap flag*) - znacznik pracy krokowej

Jeżeli jego wartość jest równa 1 to po każdej wykonanej instrukcji procesora wywoływane jest tzw. przerwanie pracy krokowej.

IF (*interrupt flag*) - znacznik zezwolenia na przerwanie

Jeżeli jego wartość wynosi 1 to przerwanie sprzętowe ma być wykonane natychmiast po zgłoszeniu, a nie po skończeniu wykonywanego programu.

DF (*direction flag*) - znacznik kierunku

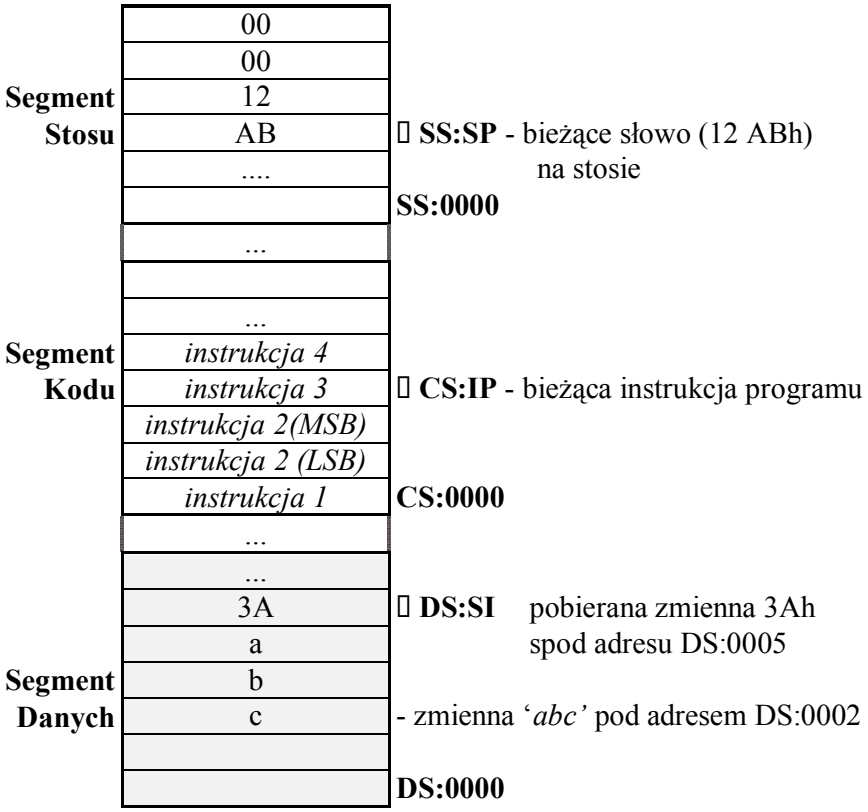
Jeżeli jego wartość wynosi 1 to dane (ciągi słów) będą pobierane w kierunku malejących adresów pamięci.

OF (*overflow flag*) - znacznik nadmiaru (przepełnienia)

Przyjmuje wartość 1 jeżeli przy wykonywaniu operacji arytmetycznej wystąpiło **przepełnienie** - tzn. przeniesienie na bit znaku (lub z bitu znaku została pobrana pożyczka), ale nie wystapila przeniesienie z bitu znaku na zewnatrz (lub pożyczka z zewnątrz na bit znaku) - tzn. CF=0. Stan tego znacznika jest istotny przy działaniu na liczbach ze znakiem.

```
   0110 1010    (+106)          1111 1111      (-1)
 + 0101 1001   +(+89)        + 1111 1111     +(-1)
------------    -----        -----------      ----
   1100 0011    (-67)        1 1111 1110      (-2)
   (OF = 1)                   (OF = 0, CF = 1)
```


1.8. Adresacja segmentów programu



MSB - Najbardziej Znaczący Bajt (Bit) *Most Significant Byte (Bit)*

LSB - Najmniej Znaczący Bajt (Bit) *Least Significant Byte (Bit)*

Przykładowy Segment Kodu

```
146B:0100 INC AX
146B:0101 INC BX
146B:0102 INC CX
146B:0103 INC DX
146B:0104 MOV AX,41
146B:0107 ADD AL,03
146B:0109
```

adres

```
-d CS:0100                  komorki pamieci                    ASCI podstwowe
146B:0100  40 43 41 42 B8 41 00 04-03 3E 43 04 00 0E 07 C3   @CAB.A.&.>C.....
146B:0110  BA 42 86 E9 65 FE BF 81-00 8B 36 92 34 00 5A 14   .B..e.....6.4.Z.
146B:0120  BE C6 DB 8B 74 09 03 C6-50 E8 0D FA 58 E8 5A 00   ....t...P...X.Z.
146B:0130  03 F1 2B C6 8B C8 E8 7B-F4 83 F9 7F 72 0B B9 7E   ..+....{....r..~
146B:0140  00 F3 A4 B0 0D AA 47 EB-08 AC AA 3C 0D 74 02 EB   ......G....<.t..
146B:0150  F8 8B CF 81 E9 82 00 26-88 0E 80 00 C3 8B 1E 92   .......&........
146B:0160  DE BE 1A D4 BA FF FF B8-00 AE CD 2F 3C 00 C3 A0   ...........</<...
146B:0170  DB E2 0A C0 74 09 56 57-E8 2A 21 5F 5E 73 0A B9   ....t.VW.*!_^s..
```

- Adresy ze stosu:

  SS:0000 - adres początku stosu

  SS:SP - adres bieżącego elementu na stosie

- Adres bieżącej instrukcji:

  CS:0000 - adres początku segmentu kodu

  CS:IP - adres bieżącej instrukcji programu

- Adresy danych:

  DS:0000 - adres początku segmentu danych

Uwaga: Przy odwoływaniu się do pamięci offset komórki z daną może być przechowywany tylko w rejestrach: **BX, BP, SI, DI** i w żadnym innym.

Uwaga: BX jest jednym rejestrem ogólnego przeznaczenia, który może przechowywać ten offset.

Przykład

DS:SI - adres w segmencie danych skąd pobierane są dane

DS:DI - adres w segmencie danych gdzie wysyłane są dane

DS:BX - adres w segmencie danych

Dla rejestrów **BX, SI, DI** domyślnym rejestrem jest DS : segment danych, tzn. użycie w instrukcji samego SI jako adresu danej oznacza, że chodzi o daną spod adresu DS:SI

Przedrostki zmiany segmentu

Jeżeli w przypadku rejestrów BX, SI, DI chcemy wskazać, że chodzi o inny segment, to można użyć przedrostków zmiany segmentu, takich jak: CS:, DS:, SS:, ES:, np.

MOV CS:SI,AX

Uwaga: *W programie DEBUG dane pobierane z pamięci należy pisać „bez dwukropka" - np. MOV ES[BX],AX*

1.9. Charakterystyka architektury IA-32

- **IA-32 (Intel Architecture 32 bit)** - 32-bitowy model programowy mikroprocesora opracowany przez firmę Intel. Nazywany także **x86-32** ponieważ opiera się na 32-bitowym rozwinięciu architektury rodziny x86.
- Architektura IA-32 zaliczana jest z reguły do kategorii CISC, choć technologie wprowadzane stopniowo w nowszych wersjach procesorów IA-32 spełniają także wiele cech procesorów RISC.
- Model IA-32 został wprowadzony w 1985 roku procesorem Intel 80386 i do dnia dzisiejszego jest najpopularniejszym modelem architektury stosowanym w komputerach, choć rozpoczął się już proces wypierania go przez model 64-bitowy EM64T (tzw. **x86-64**) i inne architektury 64-bitowe.

1.9.1. Tryby pracy

Procesory IA-32 posiadają trzy podstawowe tryby pracy, określające m.in. sposób zarządzania pamięcią i uprawnienia użytkownika.

- **tryb rzeczywisty** – tryb zgodny z najstarszymi procesorami rodziny x86 z Intel 8086 włącznie. W trybie tym występuje segmentacja pamięci, rozmiar segmentu jest stały i wynosi 64 KB. Przestrzeń adresowa ograniczona jest do 1 MB, do adresowania wykorzystuje się rejestry segmentowe oraz offset. W trybie tym współczesne procesory pracują jedynie od chwili uruchomienia do przekazania kontroli systemowi operacyjnemu.

- **tryb chroniony** - tryb inicjalizowany i w znacznej mierze kontrolowany przez system operacyjny. Pamięć może być zorganizowana w segmenty dowolnej wielkości, fizyczna przestrzeń adresowa ograniczona jest z reguły do 64 GB, liniowa przestrzeń adresowa do 4 GB. Rodzaj adresowania zależy od systemu operacyjnego - może być stosowany tzw. model płaski (bez segmentacji), model z segmentacją analogiczną do trybu rzeczywistego, lub - najczęściej - adresowanie nieliniowe (tzw. logiczne). W adresowaniu nieliniowym adres fizyczny jest zależny od wpisu w systemowej tablicy deskryptorów, na który wskazuje selektor. W trybie chronionym procesor wspiera wielozadaniowość, chroni przed nieupoważnionym dostępem do urządzeń wejścia/wyjścia.

- **tryb wirtualny V86** - odmiana trybu chronionego, która jest symulacją trybu rzeczywistego. Służy np. do uruchamiania programów MS-DOS.

- **tryb SMM** (System Management Mode) - jest to tryb przeznaczony do zarządzania sprzętem przez systemy operacyjne, niedostępny z poziomu użytkownika.

1.9.2. Podstawowe rejestry w architekturze IA-32

W procesorach opartych na modelu IA-32 dostępne są następujące rejestry:

- **Rejestry ogólnego przeznaczenia** (32-bitowe): EAX - rejestr akumulacji, EBX - rejestr bazowy, ECX - rejestr licznika, EDX - rejestr danych.

- **Rejestry wskaźnikowe i indeksowe** (32-bitowe): ESI - źródło, EDI - przeznaczenie, EBP - wskaźnik bazowy, ESP - wskaźnik stosu.

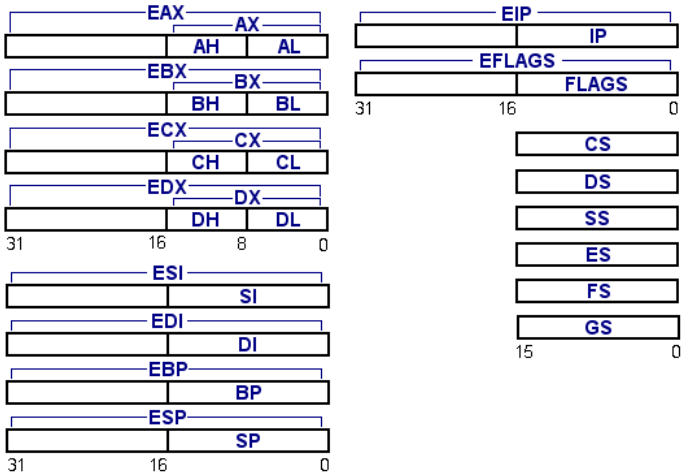
Z rejestrów ogólnego przeznaczenia można korzystać także jako rejestrów 16-bitowych (wykorzystywane jest wtedy młodsze 16 bitów rejestru 32-bitowego). Rejestry takie oznacza się z pominięciem litery E na początku symbolu. Dodatkowo, w przypadku rejestrów danych (EAX-EDX) można się odwoływać do ich 8-bitowych części - najmłodsze 8 bitów rejestru AX oznaczane jest przez AL, kolejne 8 przez AH. Odpowiednio najmłodsze bity rejestru BX oznacza się przez BL itd.

- **Rejestry segmentowe** - w procesorach IA-32 zdefiniowano sześć 16-bitowych rejestrów segmentowych, służących do określania adresu fizycznego bądź jako selektory w trybach stosujących segmentację pamięci. Są to: CS - rejestr segmentu kodu programu, DS - rejestr segmentu danych, SS - rejestr segmentu stosu, ES, FS, GS - rejestry pomocnicze dla danych.

- **Rejestr znaczników** - do opisu stanu procesora w architekturze IA-32 wykorzystuje się rejestr stanu procesora EFLAGS.

- **Wskaźnik instrukcji** - EIP - rejestr przechowujący adres aktualnie wykonywanego rozkazu, za jego pomocą procesor realizuje m.in. skoki, pętle, przejścia do podprogramów.

- **Inne rejestry** - rejestry technologii MMX i Streaming SIMD Extensions oraz rejestry kontrolne i do debugowania.



1.10. Rozkazy (instrukcje) procesora

- **Rozkaz** (instrukcja maszynowa) - jest to najprostrza operacja, którą procesor potrafi wykonac (a programista może zażądać od procesora).

- **Lista rozkazów** - zestaw wszystkich instrukcji (rozkazów), jakie potrafi wykonać dany procesor.

- **Asembler x86** - język programowania z rodziny asemblerów do komputerów klasy PC, które posiadają architekturę głównego procesora zgodną z x86. Trudność programowania w tym języku polega na tym, że każdy nowy procesor wprowadzający różne ulepszenia musi jednocześnie pozostawać kompatybilny z poprzednikami. W procesorach 80286

jest około 250 rozkazów, w 80486 już ok. 350, natomiast w procesorze Pentium 4 – ok. 580 (w procesorach firmy AMD jest ich ponad 621).

Składnia rozkazów

INSTRUKCJA [OPERAND_DOCELOWY],[OPERAND_ZRODLOWA]

Np.

dodaj *do czego?,co?*

ADD AL,03

04 03 lub 00000100 00000011
lsb

przesuń *gdzie?,co?*

MOV AX,41

B8 41 00 lub 10111000 1000001 00000000
lsb

powiększ o jeden *co?*

INC CX

41 lub 1000001

1.11. Dane (tryby adresowania)

Główne typy danych:

1) Dane natychmiastowe (adresowanie natychmiastowe)

Dane te podane są jako argument (operand) instrukcji - np. MOV BX, 0AB12h

2) Dane rejestrowe (adresowanie rejestrowe)

Dane te pobierane są z rejestrów - np. MOV BX, CX

3) Dane pobierane z pamięci (adresowanie bezpośrednie, adresowanie pośrednie)

Dane te pobierane są z pamięci w sposób:

bezpośredni - np. MOV BX, ZMIENNA; MOV DX,DS:0010;

pośredni - np. MOV BX,ES:[BX]; MOV [BX],DI

Użycie **nawiasów kwadratowych** oznacza, że odwołujemy się nie do zawartości rejestru, ale do wartości jaka jest przechowywana w komórce pamięci pod offsetem przechowywanym w tym rejestrze.

1.12. Wybrane instrukcje procesora

1.12.1. MOV

MOV - przeniesienie (skopiowanie) operandu źródłowego do operandu docelowego.

Ograniczenia instrukcji MOV

Instrukcja MOV **nie może:**

- przenosić danych bezpośrednio z jednej komórki pamięci do innej (np. MOV [SI],[BX] jest nieprawidłowe).
- przenosić bezpośrednio zawartości jednego rejestru segmentowego do innego (np. MOV CS,ES jest nieprawidłowe).
- przenosić danych natychmiastowych do rejestru segmentowego (np. MOV CS, 0B800H jest nieprawidłowe).
- przenosić jednej z 8 bitowych połówek rejestru do rejestrów 16 bitowych i odwrotnie. (np. MOV AH, BX jest nieprawidłowe).

a także - w programie DEBUG:

- przenosić danych natychmiastowych bezpośrednio do pamięci (np. MOV [2222], 0B800H jest nieprawidłowe).

1.12.2. ADD

ADD - dodawanie arytmetyczne (instrukcja dwuoperandowa)

ADD dodaje operandy źródłowy i docelowy i umieszcza wynik w operandzie docelowym.

Np.

MOV AX,0022h

MOV BX,0088h

ADD AX,BX

wynik?

AX = AA

MOV AX,0001h

ADD AX,0FFFFh

wynik?

AX = 0000h

Przykład

Co w wyniku wykonania powyższej operacji stało się z rejestrem flag?

```
AX=0001  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=146B  ES=146B  SS=146B  CS=146B  IP=0103    NV UP EI PL NZ NA PO NC

146B:0103 05FFFF        ADD     AX,FFFF
```

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=146B  ES=146B  SS=146B  CS=146B  IP=0106   NV UP EI PL . ZR . . AC
. . PE . . CY .
```

1.12.3. SUB

SUB - odejmowanie (instrukcja dwuoperandowa)

Odejmuje od operandu docelowego operand źródłowy i umieszcza wynik w operandzie docelowym.

Przykład

Wykonano operacje:

MOV AX, 0000h

SUB AX, 0001h

Jaki jest wynik i co stało się ze znacznikami?

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=000 0  DI=0000
DS=146B  ES=146B  SS=146B  CS=146B  IP=0106   NV UP EI PL NZ NA PE NC
146B:0106 2D0100          SUB     AX,0001

AX=FFFF  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=146B  ES=146B  SS=146B  CS=146B  IP=0109   NV UP EI . NG . . NZ . . .
. AC . PE . . CV . .
```

1.12.4. MUL

MUL - mnożenie liczb bez znaku.

MUL to instrukcja z . JEDNYM . operandem.

Pierwszy czynnik - operand (. ..tylko .. . rejestr ogólnego przeznaczenia lub komórka pamięci).

Drugi czynniki - wartość rejestru . AX

Iloczyn - w DX (. .. ..bardziej. .. . znaczące słowo) i AX (. .. .. znaczące słowo)

Uwaga: Jeżeli bardziej znaczące słowo iloczynu jest różne od zera, to znaczniki OF i CF są równe 1.

Przykład

MOV AX,FFFFh

MOV BX,1000h

MUL BX

wynik?

DX = . OFFF .

AX = . FOOO .

```
AX=FFFF  BX=1000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=146B  ES=146B  SS=146B  CS=146B  IP=0100   NV UP EI PL NZ NA PO NC
146B:0100 F7E3          MUL     BX

AX=. . F000 .. .  BX=1000  CX=0000  DX=. ..0FFF.. .  SP=FFEE  BP=0000
SI=0000  DI=0000
DS=146B  ES=146B  SS=146B  CS=146B  IP=0102   . OV . UP EI PL NZ NA PO
. CY .
```

1.12.5. DIV

DIV - dzielenie liczb bez znaku

DIV to instrukcja z . .. jednym .. . operandem.

Dzielnik - operand (. .. tylko.. . rejestr ogólnego przeznaczenia lub komórka pamięci).

Dzielna - musi być . .. dwa razy .. . dłuższa:

AX - gdy dzielnik jest jednobajtowy

DX i AX - gdy dzielnik ma wielkość słowa

Iloraz i reszta:

gdy dzielnik jednobajtowy, to iloraz w AL, a reszta w AH

gdy dzielnik ma wielkość słowa, to iloraz w AX, a reszta w DX

Dzielnik - 1 bajt, np. BL

Dzielna - 2 bajty - AX

Wynik: Iloraz - AL, Reszta - AH

Dzielnik - 2 bajty, np. BX

Dzielna - 4 bajty - DX i AX

Wynik: Iloraz - AX, Reszta - DX

Uwaga: Gdy iloraz nie mieści się w przeznaczonym na niego miejscu (przekracza, odpowiednio, FF lub FFFF) wygenerowanie zostanie przerwanie 00h - czyli „dzielenie przez zero".

Przykład

MOV AX,FFFFh

MOV DX,0000h

MOV BX,BBBBh

DIV BX

wynik?

AX = 0001 (iloraz)

DX = 4444 (reszta)

Przykład

MOV AX,FFFFh

MOV DX,0000h

MOV BX,BBBBh

DIV BL

wynik?

Z takiego dzielenia (FFFFh / BBh) iloraz wynosi 15Eh, reszta 55h.

co zrobi procesor? trzeba zweryfikowaÿ!

wywolane przerwanie ENT00h

1.12.6. AND, OR, XOR, NOT

AND, OR, XOR - instrukcje z dwoma operandami.

NOT - instrukcja z jednym operandem.

Wykonują operację logiczną bit po bicie i wynik umieszczają w operandzie docelowym.

AND - iloczyn logiczny.

OR - suma logiczna.

XOR - exclusive OR (suma symetryczna).

NOT - logiczna negacja (uzupełnienie do jedności).

1.12.7. NEG

Instrukcja z jednym operandem.

NEG - dopełnienie operandu do 2 (wynikiem jest kod uzupełnienowy).

1.12.8. INC, DEC

DEC - dekrementacja - zmniejszenie operandu o 1.

INC - inkrementacja - zwiększenie operandu o 1.

1.13. Program DEBUG

1.13.1. Charakterystyka programu

Debuger - program uruchomieniowy.

Służy do uruchamiania programów (com, exe) w trybie krokowym, pozwala na analizę ich działania oraz modyfikację w trakcie wykonania.

Uruchomienie programu

```
DEBUG [[dysk:][ścieżka]nazwa pliku [parametry pliku]]
```

Po uruchomieniu widać znak zachęty (-) i można wydawać polecenia.

1.13.2. Polecenia programu DEBUG

Uwaga: w poleceniach wielkość liter nie ma znaczenia.

? - wyświetla listę dostępnych poleceń

R - wyświetla stan wszystkich rejestrów

R *rejestr* - wyświetla stan *rejestru* i pozwala na jego zmianę, np.

```
-R AX
AX 00FF
: 12AB - nowa wartość rejestru
```

RF - wyświetla stan znaczników i pozwala na ich zmianę

A - tryb asemblacji, czyli wprowadzanie rozkazów dla procesora. [Enter] kończy tryb asemblacji.

T - wykonanie rozkazu w trybie krokowym. Naciśnięcie klawisza [T] powoduje wykonanie rozkazu i wyświetlenie rejestrów.

D *adres* - wyświetla 128 bajtów pamięci (8 wierszy po 16B) poczynając od wskazanego *adresu*. Samo „D" (bez *adresu*) wyświetla kolejne 128B pamięci.

E *adres bajt* - wprowadza *bajt* do pamięci pod wskazany *adres*. Polecenie to może wprowadzać także ciągi znaków (w apostrofach) lub ciągi bajtów (oddzielone spacjami), np.

```
E DS:0100 'Ala ma kota'
E DS:0100 AB CD EF 12
```

E *adres* - wprowadza podane bajty do kolejnych komórek pamięci - po podaniu wartości bajta należy nacisnąć [Spację].

F *adres L ile_bajtów 'ciąg_znaków'* - zapełnia (poczynając od podanego *adresu*) *ciągiem_znaków* określoną ilość bajtów, np.

```
F ES:0000 L 8 'ABC'
```

F *adres 'ciąg_znaków'* - zapełnia *ciągiem_znaków* blok 128B.

G *start koniec* - wykonuje rozkazy od adresu *start* do *koniec*.

G - wykonuje rozkazy od CS:IP.

H *wartość1 wartość2* - oblicza sumę i różnicę podanych wartości.

I *port* - odczytuje i wyświetla zawartość podanego portu.

O *port wartość* - wprowadza do portu podaną wartość.

Q - kończy program DEBUG

1.14. STOS

1.14.1. Charakterystyka Stosu

- Procesory 8088/8086 mają mechanizm zarządzania specjalnym obszarem pamięci zwanym **stosem**.
- W danej chwili **tylko jeden stos może być aktywny** (jest on wykorzystywany przez wszystkie programy i system operacyjny).
- Stos zorganizowany jest w systemie LIFO (Last In First Out).
- Do obsługi stosu przeznaczone są rejestry - SS:SP.
- Stos może mieć wielkość maksymalnie 64kB.
- Początek stosu jest pod adresem SS:0000, ale wszystkie operacje na nim odbywają się „od końca".
- Po utworzeniu stosu SP wskazuje jego koniec.
- Na stos można odkładać (i zdejmować) dane o wielkości 2 bajty.
- Na stosie nie można umieszczać danych natychmiastowych.
- Stos wykorzystywany jest do:

  przekazywanie danych z jednego rejestru do drugiego,

  krótkoterminowe przechowywanie danych.

1.14.2. Instrukcje do obsługi stosu

Do odkładania danych na stosie służą instrukcje:

PUSHF - odkłada zawartość rejestru flag na stosie,

PUSH - odkłada na stosie zawartość wskazanego rejestru lub daną z pamięci,

np. PUSH AX; PUSH ES; PUSH [BX] odloz na stosie zawartosc komorki o adresie BX

odloz na stosie zawartosc segentu AX

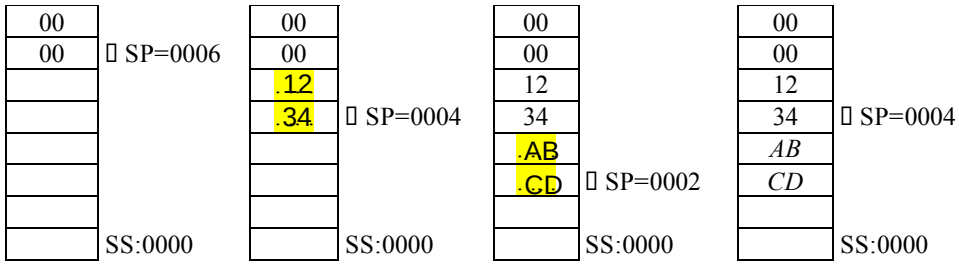
Do zdejmowania danych ze stosu służą instrukcje:

POPF - zdejmuje ze stosu słowo i umieszcza je w rejestrze flag,

POP - zdejmuje ze stosu daną dwubajtową, np. POP SI; POP [BX]

AX = 1234h
BX = ABCDh

PUSH AX **PUSH BX** **POP DX**



Uwaga: Najpierw zmniejszany jest wskaźnik SP o 2, a potem dana umieszczana na stosie.

Przykład

- PUSHF ; odłożenie na stos rejestru flag
- POP BX ; zdjęcie ostatniego słowa ze stosu i umieszczenie go w BX

1.15. Przerwania (sprzętowe i programowe)

1.15.1. Charakterystyka przerwań

- Przerwanie (interrupt) jest to zdarzenie (sygnał), które przerywa wykonywany przez procesor program i przekazuje sterowanie do specjalnego podprogramu - tzw. procedury obslugi przerwania.
- Wszystkie przerwania są ponumerowane - od 00 do FFh (225).
- **Numerów przerwań** jest 256, ale nie wszystkie są wykorzystywane.
- Źródłem przerwań może być: sprzet lub oprogramowanie.

1.15.2. Przerwania sprzętowe

- Przerwania te są wywoływane przez urządzenia (np. zegar, klawiatu ra, dysk twardy, port szeregowy, port równoległy).
- Przerwania sprzętowe mają podwojna numerację - poprzez numer przerwania i poprzez **numer przerwania** IRQ, np.

  klawiatura - nr przerwania 09h - IRQ1,

  port szeregowy - nr przerwania 0Ch - IRQ4,

- Przerwanie sprzętowe może być wygenerowane także przez procesor, np.

  00h - dzielenie przez 0,

  01h - praca krokowa - przerwanie po każdej instrukcji, gdy TF = 1,

  02h - przerwanie niemaskowalne (NMI - *Non Maskable Interrupt*) - wywoływane w sytuacji powstania powaznych bledow (np. błąd parzystości, brak zasilania), na rdzeniu procesora

  03h - punkt kontrolny (pułapka), tryb diagnostyczny pozwalajacy na zatrzymanie programu i naprawienie bledow

  04h - przepelnienie - przerwanie wywoływane w przypadku wystąpienia przepełnienia (OF = 1).

1.15.3. Przerwania programowe

- Przerwania programowe są wywoływane przez wykonywany program (za pomocą instrukcji INT).
- Przerwania udostępniają zestaw **gotowych** funkcji, dzięki którym można obsługiwać urządzenia i wykorzystywać możliwości systemu operacyjnego, np. przerwanie:

  10h - pozwala kontrolować pracę karty graficznej (np. tryb wyświetlania),

  13h - pozwala na bezpośrednią obsługę dysków,

  21h - umożliwia wywoływanie funkcji systemowych (np. utworzenie katalogu, wyświetlenie tekstu, utworzenie pliku, zmianę katalogu).

- Jedyna różnica w sposobie funkcjonowania między przerwaniami programowymi, a sprzętowymi to zdarzenie wywołujące to przerwanie:

  przerwania programowe - instrukcja INT,

  przerwania sprzętowe - sygnal elktryczny.

1.15.4. Dostęp do procedur obsługi przerwań

Jak uzyskać dostęp do procedur obsługi przerwań?

- Każda procedura obsługi przerwania znajduje się w pamięci pod własnym, ściśle określony adresem.

Skąd się tam wzięła?

- W momencie startu systemu (lub później) została:

  pobrana z plików systemu operacyjnego i zapisana w pamięci,

  przekopiowana z BIOS-u do pamięci (lub znajduje się w BIOS-ie, a dostęp do niej jest przez przestrzeń adresową procesora),

  umieszczona przez jakiś program (np. sterownik).

Sterowniki, BIOS, systemy operacyjne stale się rozwijają.

Skąd brać aktualne adresy procedur obsługi przerwań?

Zasada jego jest następująca:

Adres w pamięci gdzie znajduje się procedura obsługi przerwań może się zmieniać, ale sam **numer procedury jest** stały.

1.15.5. Tabela wektorów przerwań

Adres tabeli - początek pamięci - 0000:0000

Liczba elementów tabeli - 256 (ponieważ tyle jest przerwań)

Podstawowy element - tzw. wektor przerwania

Zawartość elementu - adres (segment:offset) początku procedury obsługi przerwania

Wielkość elementu - 4B

Wielkość tabeli - 1 KB (256x4B)

Tabela jest uzupełniana przez system operacyjny i BIOS podczas startu systemu.

		adres logiczny
wektor FFh (adres procedury obsługi przerwania o numerze FFh czyli 255)	segment (MSB)	- 0000:03FF
	segment (LSB)	- 0000:03FE
	offset (MSB)	- 0000:03FD
	offset (LSB)	- 0000:03FC
	...	
wektor 21h (adres procedury obsługi przerwania o numerze 21h czyli 33)	segment (MSB)	
	segment (LSB)	
	offset (MSB)	
	offset (LSB)	- 0000:0084
	...	
wektor 2 (adres procedury obsługi przerwania o numerze 2)	segment (MSB)	
	segment (LSB)	
	offset (MSB)	
	offset (LSB)	- 0000:0008
wektor 1 (adres procedury obsługi przerwania o numerze 1)	segment (MSB)	
	segment (LSB)	
	offset (MSB)	
	offset (LSB)	- 0000:0004
wektor 0 (adres procedury obsługi przerwania o numerze 0)	segment (MSB)	- 0000:0003
	segment (LSB)	- 0000:0002
	offset (MSB)	- 0000:0001
	offset (LSB)	- 0000:0000

- Część wektorów przerwań zarezerwowane jest do obsługi przerwan sprzetowych.
- Wektory przerwań (przerwania) o numerach od 0 do 1Fh są obsługiwane przez BIOS (system operacyjny może niektóre przechwytywać).
- Pozostałe przerwania są obsługiwane przez system operacyjny (część z nich jest przeznaczona na potrzeby użytkownika).

Zasada numerowania przerwań:

Numer przerwania - to numer pozycji w tabeli wektorów przerwań.

1.15.6. Wywoływanie przerwania programowego

1.15.6.1. Charakterystyka wywoływania przerwań

- Przerwanie programowe wywołujemy za pomocą instrukcji INT, np. INT 21h funkcja INT:numer przerwania
- Wywołując funkcję musimy określić, ktora funkcja ma być wywołana oraz podać jej ewentualne argumenty.
- Wszystkie funkcje są ponumerowane.
- Wywołując funkcję wykorzystujemy następujące rejestry:

  rejestr AH - służy do podania numeru funkcji,

  inne rejestry (lub komórki pamięci) - służą do przekazywania ewentualnych argumentow funkcji.

Przykład

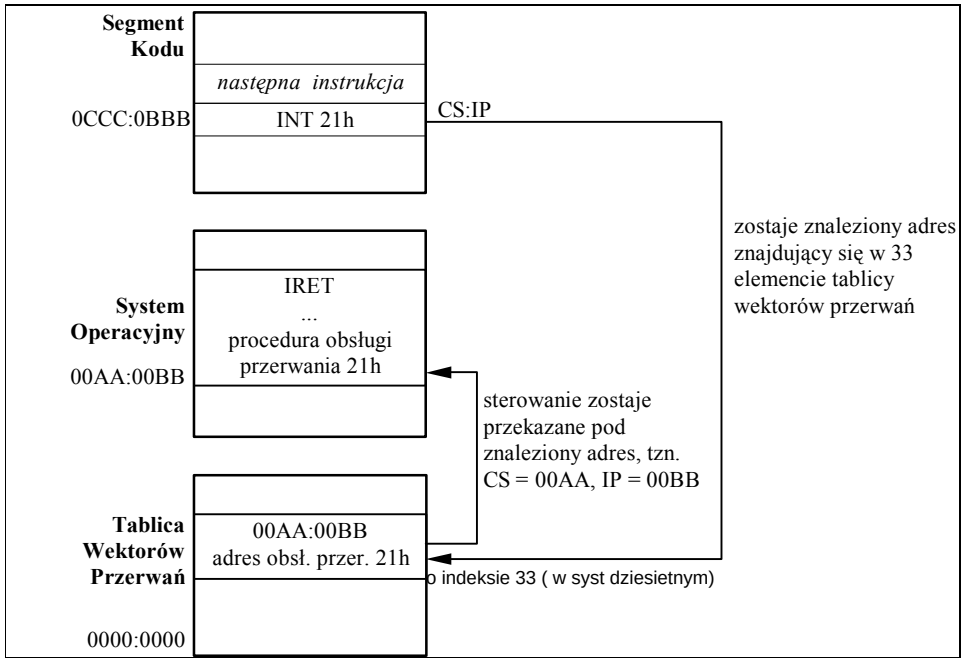
Opis funkcji: Funkcja systemowa o nr **09h** dostępna jest przez przerwanie **21h**. Służy ona do wyświetlenia łańcucha znaków z **segmentu danych**. Argumentem funkcji jest **offset** wyświetlanego łańcucha i argument ten musi być umieszczony się w rejestrze **DX**.

Polecenie: Wyświetl łańcuch znaków znajdujący się w zmiennej *nazwisko*.

Rozwiązanie:

```
MOV DX, OFFSET nazwisko ; umieszczenie w DX offsetu zmiennej nazwisko
MOV AH, 09h             ; podanie numeru funkcji
INT 21h                 ; wywołanie przerwania 21h
```

1.15.6.2. Schemat wywołania przerwania



Jak wrócić do kolejnej instrukcji programu?

1.15.6.3. Schemat powrotu z przerwania

- Instrukcja INT:

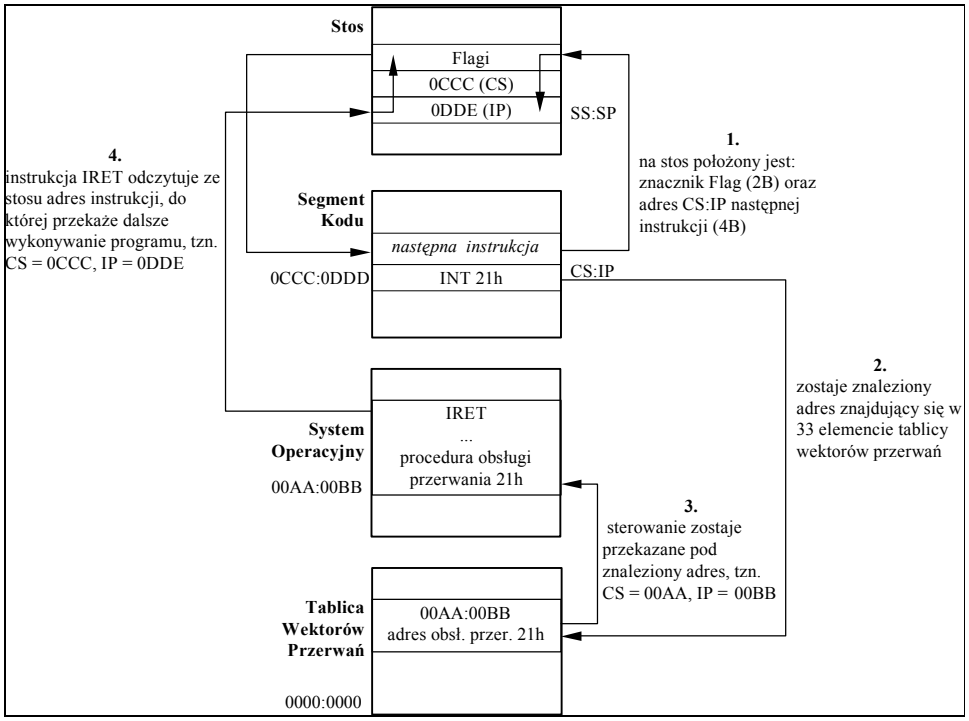
  odkłada na . stos . adres . następnej . (po sobie) instrukcji,

  odkłada na STOS zawartość rejestru . flag .,

  wywołuje przerwanie.

- Na zakończenie przerwania wywoływana jest instrukcja . IRET ., która:

  . pobiera ze stosu . położony tam element (adres kolejnej instrukcji odłożony przez INT) i pod ten adres przekazuje sterowanie.



1.15.6.4. Przechwytywanie przerwań

Przerwania sprzętowe i programowe mogą być . przechwytywane . - tzn. programista może napisać . własną . procedurę obsługi danego przerwania.

Pytanie

W jaki sposób odbywa się przechwytywanie przerwań?

1.15.7. Obsługa urządzeń wejścia/wyjścia (porty, kanały DMA)

1.15.7.1. Porty

- Port - miejsce w wyróżnionej . przestrzeni adresowej . wejścia/wyjścia, identyfikowane przez swój adres będący liczbą od 0 do . 65535 (. FFFFh .).
- Każdy port pozwala . wysylac / pobrac . bajt (lub słowo) do lub /z rejestru.
- Porty służą do komunikacji z urządzeniami . zewnetrznymi . (klawiatura, karta graficzna, itd.).
- Przykładowo, za pomocą portów można:

  zapalić (zgasić) diody na klawiaturze,

  ustalić stan przełącznika NumLock lub CapsLock,

  uzyskać informację o błędzie zerowej ścieżki na dysku,

  uzyskać informację o numerze sektora dla operacji odczytu,

  odczytać stan przycisków joysticka,

wysłać dane na port szeregowy lub równoległy,

uzyskać informacje o stanie drukarki (włączona, brak papieru, zajęta),

uzyskać informacje o atrybutach wyświetlanego przez kartę obrazu,

pobrać bieżący czas systemowy.

- Operacje na portach wykonuje się za pomocą rozkazów IN (pobranie) i OUT (wysłanie).

IN - odczyt danej z portu. z punktu widzenia procesora, czyli ze procesor wysyla do portu jakis rozkaz

Odczytana wartość umieszczona zostaje w rejestrze AL (AX)

Składnia:

```
IN AL,adres_portu        ; adres_portu - nie większy niż FFh
IN AX, DX                ; DX - zawiera adres portu
```

OUT - zapisanie danej z rejestru AL (AX) do portu

Składnia:

```
OUT adres_portu,AL       ; adres_portu - nie większy niż FFh
OUT DX,AX                ; DX zawiera adres portu
```

Przestrzeń adresowa układów wejścia/wyjścia

Adres Portu	**Nazwa układu**
000 - 01F	Kontroler DMA nr 1 (8237A-5)
020 - 03F	Kontroler przerwań nr 1 (8259A) - obsługuje przerwania zgłaszane np. przez zegar czasu rzeczywistego, czy klawiaturę.
040 - 05F	Generatory programowalne
060 - 06F	Kontroler klawiatury (8042)
070 - 07F	Zegar czasu rzeczywistego, CMOS
080	Port używany przez POST do sprawdzania urządzeń
080 - 08F	Rejestr stron DMA

0A0 - 0BF	Kontroler przerwań nr 2 (8259A)
0C0 - 0DF	Kontroler DMA nr 2 (8237A-5)
0F1	Reset koprocesora arytmetycznego
0F8 - 0FF	Porty koprocesora arytmetycznego
1F0 - 1F8	Kontroler dysków twardych
200 - 207	Game port
278 - 27F	Port równoległy nr 2
2F8 - 2FF	Port szeregowy nr 2
378 - 37F	Port równoległy nr 1
3B4 - 3BA	Sterownik VGA (mono)
3C0 - 3DA	Karta VGA
3F0 - 3F7	Kontroler dysków elastycznych
3F8 - 3FF	Port szeregowy nr 1

1.15.7.2. Kanały DMA

DMA (Direct Memory Access) - bezpośredni dostęp do pamięci - używany do szybkiego przesyłania bloków pamięci do urządzeń wejścia/wyjścia bez udzialu procesora. Wykorzystywany m.in. przez napędy dysków, dyskietek, karty dźwiękowe, karty graficzne.

Odpowiedzialny za transmisję danych jest tzw. kontroler DMA.

1.16. Programowanie w Asemblerze (TASM)

1.16.1. Charakterystyka programowania

- Kod źródłowy programu piszemy w dowolnym edytorze ASCII - tradycyjne rozszerzenie plików to ASM.
- Po napisaniu programu należy poddać go kompilacji i linkowaniu.
- Kompilator (**tasm.exe**) w procesie kompilacji tłumaczy kod źródłowy do tzw. pliku obiektowego (rozszerzenie OBJ) - są to tzw. relokowalne *moduły kodu maszynowego*

```
c:\tasm>tasm.exe program.asm
Turbo Assembler Copyright (c) Borland International

Assembling file:   program.asm
Error messages:    None
Warning messages:  None
Passes:            1
Remaining memory:  442k
```

- Linker (**tlink.exe**) w procesie linkowania (łączenia, konsolidacji) tworzy z jednego lub kilku plików obiektowych postać wykonywalna programu (plik EXE).

```
c:\tasm>tlink.exe program.obj
```

- Podzielenie procesu tworzenia postaci wykonalnej programu daje możliwość:
  - tworzenia fragmentów kodu, które mogą być sprawdzane, testowane i poprawiane niezależnie od siebie,

  - tworzenie zbiorów procedur (bibliotek), które mogą być później wielokrotnie wykorzystywane.

1.16.2. Struktura programu asemblerowego

Komentarze

Znakiem rozpoczynającym komentarz jest średnik (;)

Zmienne

Definiowanie zmiennych odbywa się poprzez podanie nazwy zmiennej, dyrektywy definicji (DB - definiuj bajt, DW - definiuj słowo, DD - definiuj podwójne słowo) i - ewentualnie - jej wartości.

Składnia:

nazwa_zmiennej dyrektywa wartość_zmiennej

Np.

```
zmienna DB 2Ah              ;zmienna wielkości bajta
zmienna DW 34CDh            ;zmienna wielkości słowa
tekst DB „Ala ma”,” kota”   ;ciąg znaków
zmienna DB ?                ;zmienna bez podanej
                            ;wartości
```

Funkcje

OFFSET - zwraca offset podanej zmiennej

SEG - zwraca segment podanej zmiennej

Uwaga: W kodzie programu liczba szesnastkowa nie może rozpoczynać się od „litery” (A, B, C, D, E, F). Do takiej liczby należy dopisać z przodu zero, tzn. zamiast FFh musi być 0FFh.

Sekcje programu

Pisanie programu wymaga zdefiniowania kilku podstawowych sekcji. W tych sekcjach należy:

- określić model pamięci – sekcja **.MODEL**
- ustalić wielkość stosu – sekcja **.STACK**
- zdefiniować zmienne – sekcja **.DATA**
- wprowadzić kod programu – sekcja **.CODE**

Program kończy się dyrektywą **END**.

Uwaga: Model pamięci mówi o tym, w jaki sposób program będzie wykorzystywał pamięć operacyjną:

- tiny – łączna wielkość kodu i danych nie może być większa niż 64KB,
- small – segment kodu nie większy niż 64KB, segment danych nie większy niż 64KB; jeden segment kodu i jeden danych,
- medium – segment danych nie większy niż 64KB, segment kodu o dowolnej wielkości; wiele segmentów kodu i jeden segment danych,
- compact – segment kodu nie większy niż 64KB, segment danych o dowolnej wielkości; wiele segmentów danych i jeden segment kodu,
- large – segment kodu większy niż 64KB, segment danych większy niż 64KB; wiele segmentów kodu i danych,
- huge – podobnie jak large, ale zmienne (np. tablice) mogą być większe niż 64KB.

Przykładowo:

```
.MODEL tiny          ;określenie tzw. modelu pamięci
.STACK 100h          ;określenie wielkości stosu
.DATA                ;obszar definicji zmiennych
zmienna db 'Jan Kowalski', '$'
.CODE                ;obszar kodu

...                  ;kod programu

mov AH,4Ch           ;zakończenie programu -
int 21h              ;funkcja 4Ch z przerwania 21h

END                  ;koniec kodu programu
```


Program z procedurą:

```
.MODEL tiny
.STACK 100H
.DATA
zmienna db 'Jan Kowalski', '$'
.CODE

...
call NazwaProcedury    ;wywołanie procedury
...

mov AH,4Ch             ;zakończenie programu
int 21h

NazwaProcedury:        ;definicja procedury
...                    ;kod procedury
RET                    ;instrukcja powrotu
                       ;z procedury
END
```

1.16.3. Przykładowe programy

1.16.3.1. Program wyswietl.asm

Program wyświetlający komunikat i (poniżej) jego pierwszy znak

```
.MODEL tiny
.STACK 100H
.DATA
```

```
komunikat db 'Dzien dobry',13,10,'$'
.CODE

;Do wyświetlenie na ekranie ciągu znaków służy funkcja 09
;z przerwania 21h. Składnia:
;AH  - numer funkcji, DS:DX - adres ciągu znaków
;Uwaga: ciąg musi kończyć się znakiem '$'

mov BX,SEG komunikat        ;nie wolno bezpośrednio przenieść
mov DS,BX                   ;do DS segmentu zmiennej
mov DX, OFFSET komunikat    ;do DX offset zmiennej

mov AH,9                    ;numer funkcji
int 21h                     ;numer przerwania

;Do wyświetlenia znaku na ekranie służy funkcja 02
;z przerwania 21h. Składnia:
;AH - numer funkcji, DL - znak (jego kod) do wyświetlenia

mov DX,seg komunikat            ;segment zmiennej do DX
mov DS,DX                       ;a potem do DS
mov DL,DS:[offset komunikat]    ;znak jest pod tym adresem

mov AH,02h                      ;nr funkcji
int 21h                         ;nr przerwania

mov AH,4Ch                      ;zakończenie programu
int 21h

END
```