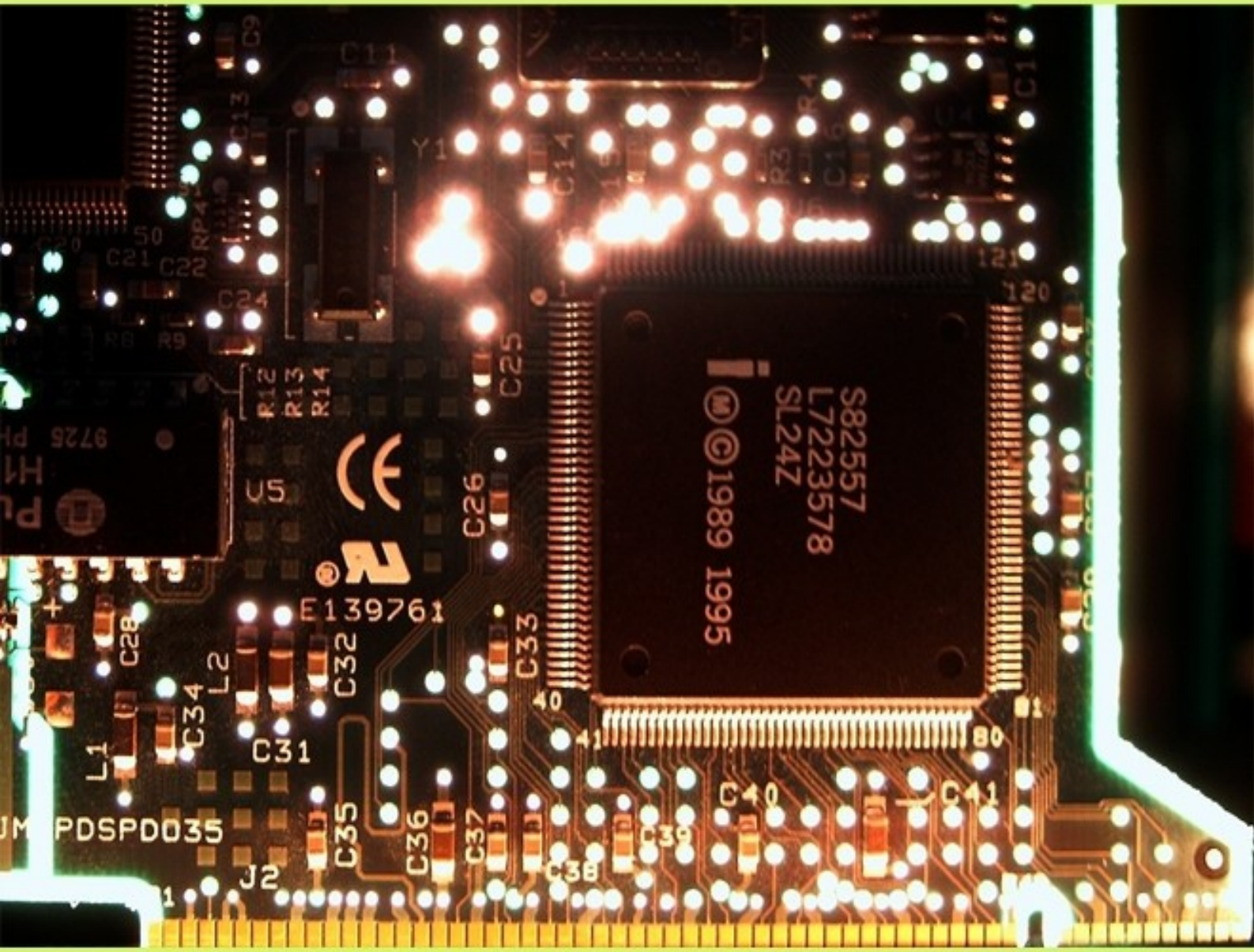
Asembler od podstaw



Stanisław Kruk

Stanisław Kruk

Asembler od podstaw

darmowy fragment

ASEMBLER OD PODSTAW

Stanisław Kruk

Skład i łamanie: Patrycja Kierzkowska

Korekta: Anna Matusewicz

Jędrzejów 2007

ISBN: 978-83-60320-87-7

Wydawnictwo Publikacji Elektronicznych Escape Magazine

ul. Spokojna 14

28-300 Jędrzejów

http://www.escapemag.pl

Spis treści

Rozdział 1
O językach programowania

Nie bez powodu i z pewną wyniosłością, można powiedzieć tak: języków programowania jest wiele, ale Asembler jest jeden. To jest prawda i tylko prawda. Zanim zaczęto posługiwać się językami programowania, Algol, B, Basic, C, Cobol, Fortran, Pascal itp. programy pisano w Asemblerze, który naówczas miał postać ciągów zerojedynkowych. Wymienione tu języki programowania Algol, B, Basic, C, Cobol, Fortran, Pascal należą do kategorii języków wysokiego poziomu, z właściwą dla nich składnią, zbiorem słów kluczowych, instrukcji, dyrektyw itp. Język Asembler jest językiem niskiego poziomu i chociaż podobnie jak języki wysokiego poziomu dysponuje określonym zbiorem słów kluczowych, instrukcji, dyrektyw, to jednak przy jego pomocy możemy całkowicie ujarzmić naszą maszynę cyfrową. Co nie jest możliwe, posługując się językami wysokiego poziomu.

Programista asemblerowy jest trochę podobny do pierwszoklasisty, który konstruuje słowa, wyrazy, znając pojedyncze literki języka. O ile alfabet języka naturalnego, którym to posługujemy się na co dzień, składa się z kilku dziesiątek znaków, o tyle alfabet języka Asembler ma zaledwie dwa cyfrowe znaki: **0** i **1**, zero i jedynkę. Okazuje się, że tworząc różnej długości ciągi cyfrowe złożone z tych dwóch znaków 0, 1, możemy opisać wszystko to, co dzieje się wokół nas, na ziemi i na niebie.

Rozdział 2
Natura języka Asembler

Język Asembler, podobnie też jak i inne języki programowania nie należą – rzecz jasna – do języków naturalnych, czyli języków, którymi posługują się ludzie. Jednak dosyć dziwnym może się wydawać fakt, że Asembler to w ogóle jakiś język, bowiem zaledwie posługuje się dwoma znakami cyfrowymi, **0** i **1**; alfabety języków naturalnych zawierają przecież kilka dziesiątek znaków. Asembler jest językiem maszyny a dokładniej, językiem procesora, opartym o dwa sposoby komunikacji, albo *prąd płynie* albo *nie płynie*, **1** albo **0**. Okazuje się, że za pomocą tylko tych dwóch cyfr można wyrazić (zakodować) dowolną liczbę, symbol, znak itp.

Z drobnych, małych, elementów da się zbudować bardziej finezyjną i wymyślniejszą budowlę aniżeli z dużych elementów. Ta oczywista prawda nabrała właściwego wymiaru dopiero wówczas, gdy dokładnie zrozumiano rolę binarnego *wyrażania czegokolwiek* za pomocą dwóch cyfr **0** i **1**.

Rozdział 3
Środowisko Asemblera

Asembler, w odróżnieniu od języków wysokiego poziomu (np. Basic, C, Pascal) nie potrzebuje specjalnego i „wygodnego" środowiska, bowiem *urodził* się on na najniższym poziomie językowym, *leżąc na gołym sprzęcie*. Do napisania programu asemblerowego wystarczy najzwyklejszy, czysty, czyli niedający różnych niepotrzebnych znaków, edytor; do napisania programów w językach wysokiego poziomu te *spartańskie* warunki też mogą wystarczyć, jednakże tamte środowiska są zazwyczaj rozbudowane. Gdy posiadamy już gotowy program asemblerowy w postaci tekstowej, poddajemy go odpowiedniemu przetworzeniu w taki sposób, by ze „zwykłego" tekstu zrobił się uruchamialny w systemie (DOS/Windows) program o rozszerzeniu COM lub EXE czy też DLL. W odróżnieniu od programów źródłowych napisanych w języku C, Pascal itp., które tłumaczy się do postaci programów wykonywalnych kompilując je, programy asemblerowe poddaje się asemblacji, po której powstaje „półprodukt" o rozszerzeniu OBJ. Następnie ów „półprodukt" przetwarza się do postaci wykonywalnej poprzez konsolidację (linkowanie). Po konsolidacji otrzymujemy gotowy do wykonania program typu COM lub EXE.

W ten sposób z asemblerowego pliku tekstowego o rozszerzeniu ASM zawierającego rozkazy i instrukcje języka Asembler powstaje wpierw plik obiektowy, OBJ, z którego ostatecznie, dopiero po zlinkowaniu, tworzony jest plik wykonywalny o rozszerzeniu COM, EXE czy też DLL - dla systemu Windows. *Bardzo początkujący* programista asemblerowy na razie nie musi się tym wszystkim przejmować, wystarczy, żeby umiał zrobić prosty, wykonywalny, program (tylko typu COM). Tym narzędziem jest program **DEBUG** (zazwyczaj występuję jako EXE), jest to narzędzie bardzo proste w obsłudze i dość skuteczne. Obsługa jego programu opisana zostanie w dalszych rozdziałach, teraz trzeba jednak zapamiętać tylko tyle, że program DEBUG jest podstawowym narzędziem polecanym początkującym programistom asemblerowym. Program ten dołączany jest do systemu operacyjnego, zarówno do wersji DOS jak też i Windows.

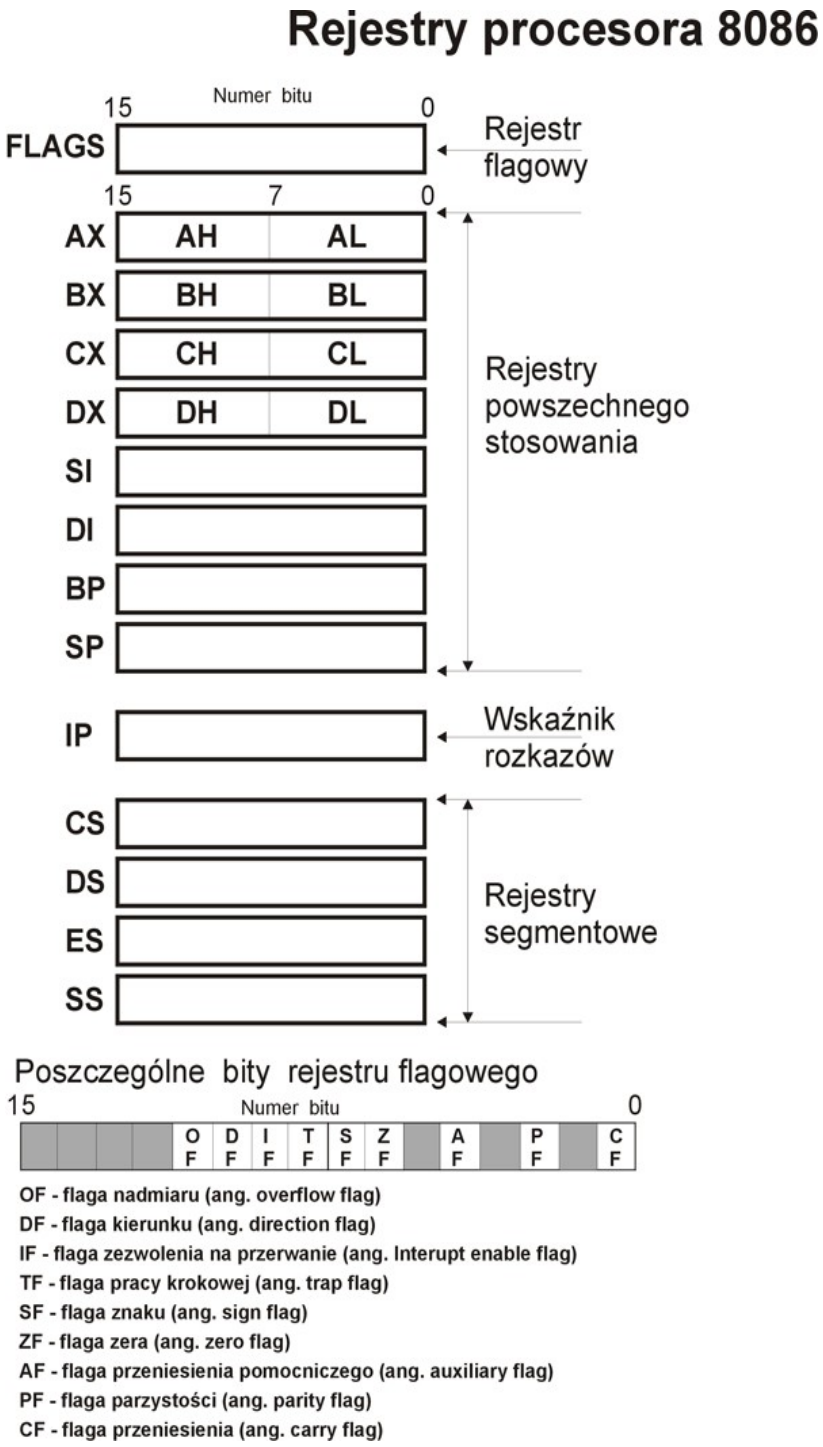
W tym miejscu należy wyjaśnić, iż słowo Asembler zapisywane wielką literą oznacza język programowania, gdy natomiast mowa będzie o programie do przetwarzania, do asemblacji, wówczas asembler – jako program - piszemy małą literą.

Rozdział 4
Rejestry i rozkazy.
Istota programowania

Rejestry są bardzo szybkimi w działaniu elementami elektronicznymi posiadającymi zdolność zapamiętywania informacji.

Wewnątrz procesora 8086/8088 znajduje się 14 rejestrów, za ich pomocą można przesyłać i przetwarzać dane. Rejestry te są w stanie czasowo przechowywać dane, adresy pamięci, wskaźniki rozkazów i stanów oraz znaczniki (flagi) sterujące; poprzez nie procesor ma dostęp do ponad 1 MB pamięci i do 64 KB portów I/O. Każdy z rejestrów ma swoją rolę do spełnienia. Można je pogrupować według podobnych zadań:

- rejestry powszechnego zastosowania,
- rejestry segmentowe,
- rejestry wskaźnikowe i indeksowe,
- wskaźnik rozkazów,
- rejestr znaczników (rejestr flagowy) – FLAGS.



Ryc. 1 Rejestry procesora 8086

Rozkazy - pewnego rodzaju polecenia operujące głównie na rejestrach; w języku bardziej informatycznym - **rozkaz** jest to elementarna operacja, jaką może wykonać procesor. Jeśli zechcemy przesłać (skopiować) jakąś liczbę z jednego rejestru procesora do drugiego, musimy wówczas rozkazać procesorowi, na jakich rejestrach (wyszczególniamy nazwy rejestrów: rejestr źródłowy, rejestr docelowy) ma wykonać to kopiowanie (podajemy nazwę rozkazu) oraz bezpośrednio kopiowaną liczbę. Oprócz tego rodzaju poleceń – rozkazów - zależnych tylko i wyłącznie od specyfiki procesora używa się też poleceń związanych z samym środowiskiem asemblera - jako programu. Jednakże, na razie, ta wiedza nam wystarczy, albowiem *bardzo początkujący* programista asemblerowy używa programu uruchomieniowego DEBUG, który nie ma szczególnych środowiskowych wymagań. Już teraz należy jednak wiedzieć, iż rozkazy procesora operują nie tylko na jego własnych rejestrach, ale i na całym szeroko pojętym otoczeniu procesora: pamięć, porty.

4.1. Rejestry powszechnego zastosowania

Osiem rejestrów powszechnego zastosowania (każdy o długości 16 bitów) jest używanych do najczęściej stosowanych rozkazów związanych z kopiowaniem, wskaźnikami do określonych miejsc w pamięci czy licznikami. Każdy z tych ośmiu rejestrów może być załadowane danymi znajdującymi się w pamięci, jak też z nich można załadować do pamięci, można też ich używać do operacji arytmetycznych i logicznych.

Przykłady:

```
MOV AX,[Liczba]     ;prześlij do rejestru AX liczbę zawartą w komórce pamięci pod adresem o nazwie Liczba
MOV AX,51           ;prześlij do rejestru AX liczbę dziesiętną o wartości 51
MOV DX,10           ;prześlij do rejestru DX liczbę dziesiętną o wartości 10
ADD AX,DX           ;dodaj dwie liczby uprzednio przesłane (51, 10), a wynik dodawania prześlij do rejestru AX
```

Uwaga! Już na samym początku zapisywania programów asemblerowych, (tych pisanych nawet na kartce), należy wyrabiać sobie dobry nawyk komentowania zapisywanych linii programów. Z uwagi na to, iż długie programy asemblerowe mogą zawierać niekiedy tysiące linii kodu, a takie programy pisać można nawet całymi tygodniami, wówczas umieszczanie komentarza tuż za każdą linią programu jest często sprawą wręcz *zbawienną*, zwłaszcza gdy na czas jakiś odeszliśmy od pisania takiego „tasiemca". W języku Asembler komentarze umieszczamy po znaku średnika.

Spostrzegawczy Czytelnik, patrząc na przykłady rozkazów, zapewne zaraz się domyśli, że nazwy tych rozkazów mają zapewne swój źródłosłów w języku angielskim. I taka też jest prawda. Nazwa rozkazu MOV jest skrótem od MOVE (z ang. przesuń, posuń, *prześlij*), ADD (z ang. dodawać). Patrząc na rysunek 4.1 nie trudno w tych rozkazach rozpoznać nazwy rejestrów, AX, DX. Nazwy rozkazów to nazwy mnemoniczne lub krótko – mnemoniki; MOV,

ADD, SUB, CLC, FSTP itp., itd. – to są właśnie mnemoniki. Operacje zakodowane w rozkazie mają zawsze swój określony (i ten sam) kierunek.

Przykłady rozkazów:

MOV DX,10

Rozkaz ten oznacza: prześlij liczbę o wartości dziesiętnej 10 (miejscem źródłowym liczby 10 jest sam rozkaz) do rejestru o nazwie DX (miejsce docelowe).

MOV [X],AH

Rozkaz ten oznacza: prześlij liczbę zawartą w rejestrze AH (AH – miejsce źródłowe) do komórki pamięci operacyjnej znajdującej się pod adresem o nazwie X (X – miejsce docelowe).

~~MOV [X],[Y]~~ – takie przesłanie między komórkami pamięci nie jest możliwe.

ADD AX,DX

Rozkaz ten oznacza: dodaj liczbę zawartą w rejestrze DX (miejsce źródłowe) do liczby zawartej w rejestrze akumulatora, AX (miejsce docelowe), zaś wynik umieść (też) w rejestrze AX (miejsce docelowe).

Mimo iż rejestry powszechnego zastosowania mają wspólne cechy, to każdy z tych rejestrów z osobna, ma swą specyfikę. Zapoznajmy się teraz bliżej z poszczególnymi rejestrami powszechnego zastosowania.

Rejestr AX – znany jako akumulator. Używany zawsze tam, gdzie zachodzi potrzeba mnożenia, dzielenia. Jest to najbardziej efektywny rejestr używany w operacjach arytmetycznych, logicznych, przesyłania danych. Dolna, 8-bitowa część rejestru AX nosi nazwę AL (ang. *A-Low*), część górna, też 8-bitowa, nosi nazwę AH (ang. *A-High*). Taki podział rejestru na dwie 8-bitowe części jest wygodny podczas działań na danych 1-bajtowych, tworząc tym samym dwa niezależne rejestry, poza tym jest on pozostałością po dawnych 8-bitowych rejestrach.

Rejestr BX – może wskazywać położenie, lokalizację w pamięci. 16-bitowa wartość zapamiętana w tym rejestrze może być po części użyta do adresowania położenia w pamięci. Na przykład, do rejestru AL ładowana jest zawartość pamięci spod (umownego) adresu.

Przykłady:

```
MOV AX,0     ;prześlij do rejestru AX liczbę o wartość 0
MOV DX,AX ;(prześlij) kopiuj zawartość rejestru AX do rejestru DX
MOV BX,16   ;prześlij do rejestru BX liczbę dziesiętną o wartości 16
```

```
MOV AL,[BX]          ;prześlij do rejestru AL zawartość komórki pamięci
```

(operacyjnej) spod adresu o wartości 16

Domyślnie rejestr BX, wraz z rejestrem segmentowym DS, jest używany jako wskaźnik pamięci. Rejestr BX może być traktowany jako dwa 8-bitowe rejestry – BH i BL. Jeśli Czytelniku nie do końca zrozumiałeś wszystkie podanych dotąd przykładów, a zwłaszcza tego związanego z przesyłaniem do pamięci czy z pamięci (problem adresowania pamięci!), to nie przejmuj się tym. O tych kwestiach napisano w różnych miejscach niniejszej książki i to nie przy okazji jak uczyniono powyżej, ale odrębnie.

Rejestr CX – używa się go głównie jako licznika odliczającego powtarzające się fragmenty programów bądź pojedynczych rozkazów. Rejestr CX może być traktowany jako dwa 8-bitowe rejestry, CH i CL.

Rejestr DX – używa się go głównie jako wskaźnika adresów w rozkazach IN i OUT – rozkazy I/O (wejścia/wyjścia). Nie ma bowiem innego sposobu zaadresowania portów, aniżeli użycie rejestru DX. Następujący przykład pokazuje zapis danej o wartości 21 do portu o numerze 44:

```
MOV AL,21   ;prześlij do rejestru AL wartość 21
```

```
MOV DX,44   ;prześlij do rejestru DX wartość 44, liczba ta będzie nr. portu
(następny rozkaz)
OUT DX,AL   ;wyprowadź bajt z portu 44 do rejestru AL
```

Rejestr DX może być też użyty w operacjach mnożenia i dzielenia. Jeśli dzielimy 32-bitową liczbę (dzielną) przez liczbę 16-bitową (dzielnik), „górne" 16 bitów dzielnej musi być umiejscowione w DX; po podzieleniu reszta z dzielenia zapamiętywana jest w rejestrze DX. („Dolne" 16 bitów dzielnej musi być umieszczone w rejestrze akumulatora AX, wynik dzielenia, iloraz, zapamiętywany jest w rejestrze AX). Podobnie jest też podczas mnożenia dwóch liczb 16-bitowych, górne" 16 bitów iloczynu jest zapamiętywane w rejestrze DX, „dolne" zaś 16 bitów iloczynu w rejestrze AX). Rejestr DX może być traktowany jako dwa 8-bitowe rejestry, DH i DL.

4.2. Rejestry wskaźnikowe i indeksowe

Rejestr SI – rejestr ten podobnie jak też **rejestr BX**, może być użyty jako wskaźnik pamięci.

Przykłady:

```
MOV AX,21   ;prześlij do rejestru AX liczbę o wartość 21
MOV DS,AX   ;prześlij do rejestru DS zawartość rejestru AX
MOV SI,51   ;prześlij do rejestru SI liczbę dziesiętną o wartości 51
MOV AL,[SI] ;prześlij do rejestru AL zawartość pamięci spod adresu DS:SI,
DS:[0051]
```

Rejestr DI – jest bardzo podobny w użyciu do rejestru SI. Może być użyty jako wskaźnik pamięci; przybiera specjalne własności, gdy zostanie użyty w rozkazach związanych z ciągami znaków.

Przykłady:

```
MOV AX,0     ;prześlij do rejestru AX liczbę o wartości 0
MOV DS,AX  ;prześlij do rejestru DS zawartość rejestru AX
MOV DI,512   ;prześlij do rejestru DI liczbę dziesiętna o wartości 512
ADD BL,[DI]  ;dodaj do BL zawartość pamięci spod adresu DS:DI, DS:[0512]
```

Rejestry DI i **SI** wraz z rejestrem **DS** związane są z adresowaniem ciągów znakowych, z tym, że rejestr DI występuje zawsze, gdy adresowanie dotyczy źródła danych, zaś rejestr SI występuje wówczas, gdy to adresowanie dotyczy miejsca przeznaczenia (celu). Rejestry SI i DI użyte jako wskaźniki pamięci z niełańcuchowymi rozkazami odnoszą się zawsze względem rejestru segmentowego DS.

Przykład:

```
CLD          ;znacznik kierunku DF przyjmuje wartość 0
MOV DX,0     ;prześlij do rejestru DX liczbę o wartości 0
MOV ES,AX  ;prześlij do rejestru ES zawartość rejestru AX
MOV DI,2048;prześlij do rejestru DI o wartości 2048
STOSB        ;prześlij bajt z rejestru AL do pamięci adresowanej rejestrami
ES:DI, ES:[2048]
```

Rejestr BP – podobnie jak BX, SI, DI – może być użyty jako wskaźnik pamięci, jednakże z pewną różnicą. Podczas gdy rejestry BX, SI, DI, wskazując na adres w pamięci, odnoszą się względem rejestru segmentowego DS, to rejestr BP, służąc za wskaźnik pamięci, odnosi się do rejestru SS, rejestru segmentu stosu. (Adresowanie poprzez segment stosu, stosowane jest zwłaszcza w procedurach języków C, Pascal i dokonuje się ono właśnie przy użyciu rejestru BP).

Przykład:

```
PUSH BP          ;odłóż na stos zawartość rejestru BP
MOV BP,SP        ;prześlij do rejestru BP zawartość rejestru SP
MOV AX,[BP+8]    ;prześlij do rejestru AX zawartość stosu spod adresu
SS:[BP+8]
```

Rejestr SP – znany jest jako wskaźnik stosu. Należy do ostatnich rejestrów powszechnego stosowania. Rejestr SP daje położenie bieżącego wierzchołka stosu i jest analogiczny do rejestru IP. Umieszczanie wartości na stosie (ang. pushing) dokonywane jest za pomocą rozkazu PUSH, a zdejmowanie wartości ze stosu (ang. popping) odbywa się przy użyciu rozkazu POP.

Przykład:

```
MOV BX,7    ;prześlij do rejestru BX liczbę o wartości 7
PUSH BX     ;odłóż na stos zawartość rejestru BX
POP AX      ;zdejmij ze stosu zawartość rejestru BX (uprzednio odłożoną,
jeśli ten rozkaz POP AX występuje tuż po PUSH BX) i (następnie) prześlij ją
do AX
```

4.3. Rejestry segmentowe

W zwykłym trybie pracy procesora 8086 ma on zdolność „widzenia" tylko 1MB pamięci. W celu jej łatwego zaadresowania, pamięć została podzielona na 64KB segmenty. Adresowaniem tych segmentów, kawałków pamięci, zajmują się właśnie rejestry segmentowe; szczegóły tworzenia adresu wykraczają poza ramy niniejszej książki, a poza tym dla *bardzo początkującego* programisty nie wnoszą one nic ważnego w kwestii nauki programowania. Jedynie istotną kwestią jest krótkie omówienie rejestrów segmentowych.

Rejestr CS związany jest z segmentem kodu, **rejestr DS** z segmentem danych, **rejestr SS** z segmentem stosu, natomiast **rejestr ES** (ang. extra segment) jest dodatkowym rejestrem, który może być potrzebny w „awaryjnych" sytuacjach. Należy dodać, iż pełny adres położenia rozkazu w segmencie kodu wskazywany jest przez parę rejestrów: rejestru **CS** i rejestru **IP** (patrz dalej: rejestr wskaźnika rozkazów). Ta para rejestrów zapisywana jest w postaci: **CS:IP**. **Uwaga!** Rejestru CS nie można załadować. Zrobić to może tylko sama specyfika programu, jednakże nie z bezpośredniej woli programisty.

4.4. Wskaźnik rozkazów

Rejestr IP – nazywany jest wskaźnikiem rozkazów i zawiera zawsze tzw. offset pamięci, miejsce, w którym zawarty jest następny rozkaz do wykonania. Bazowy adres segmentu kodu zawarty jest w rejestrze CS. Pełny adres logiczny wykonywanego rozkazu wskazywany jest więc przez parę rejestrów CS:IP.

Gdy jeden rozkaz jest wykonywany, wskaźnik rozkazów, IP, ustawiany jest do następnego adresu pamięci, pod którym znajduje się rozkaz do wykonania. Zazwyczaj następnym rozkazem w pamięci jest właśnie rozkaz, który będzie wykonywany. Jednakże niektóre rozkazy, takie jak rozkazy wywołania i skoku, mogą spowodować rozgałęzienie w wykonywaniu się kodu programu, a tym samym w rejestrze IP nie wystąpi kolejna wartość offsetu, który wskazuje następny rozkaz do wykonania.

4.5. Rejestr znaczników

Rejestr znaczników (rejestr flagowy) – FLAGS jest czternastym i ostatnim rejestrem procesora 8086. Rejestr ten jest zbiorem poszczególnych bitów kontrolnych, zwanych znacznikami (flagami), które wskazują wystąpienie określonego stanu. Znaczniki mogą być wykorzystywane zarówno przez procesor, jak też bezpośrednio przez programistę. Niedoświadczony programista nie powinien dokonywać jakichkolwiek zmian stanu znaczników rejestru FLAGS, jeśli nawet potrafi dostać się do tego rejestru.

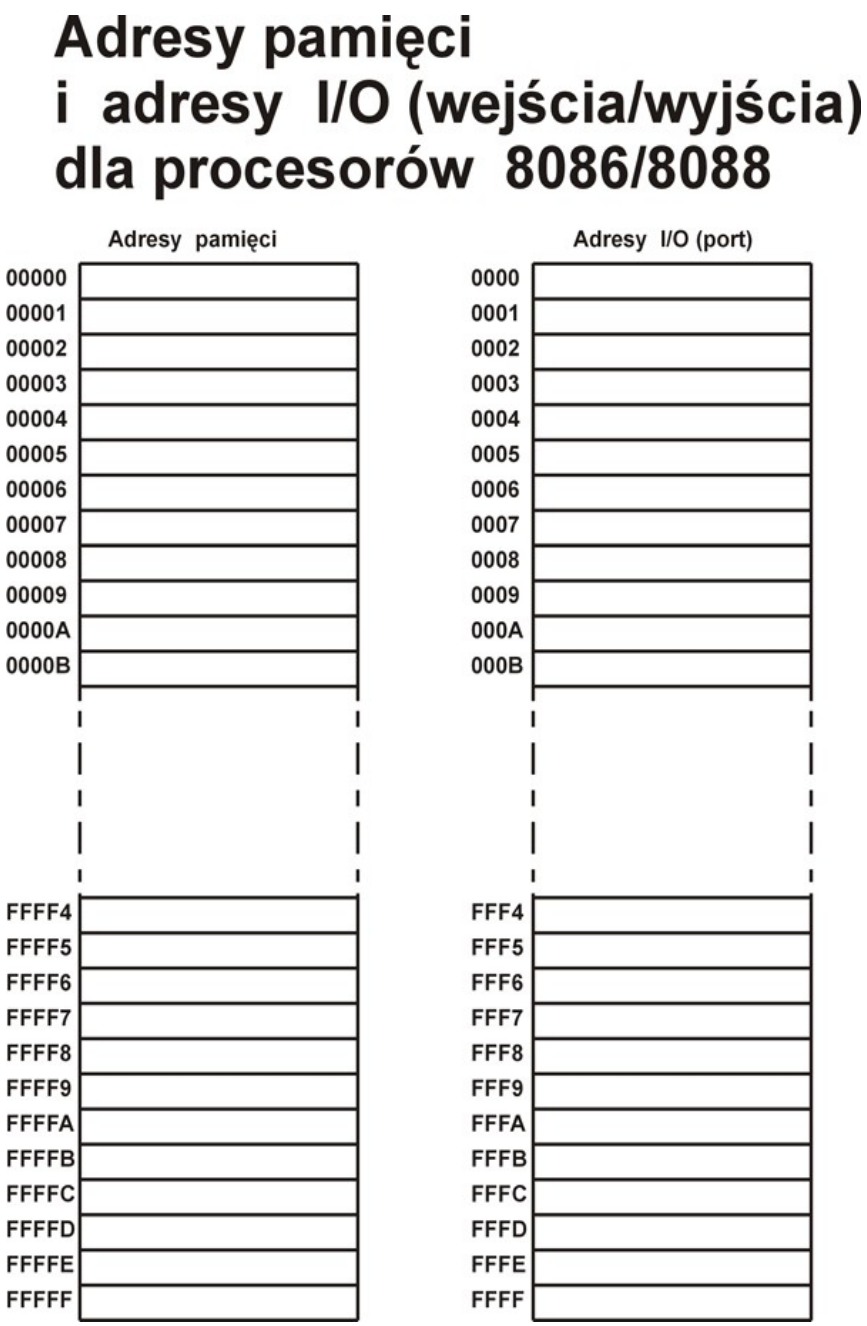
Rozdział 5
Pamięć operacyjna: przechowywanie odwrotne, stos

Areną zdarzeń dla procesora jest pamięć operacyjna. Stąd procesor pobiera kody i dane, które tworzą program. W odpowiedniej komórce pamięci znajdować się musi właściwa *pożywka* dla procesora. To co my nazywamy programem, w pamięci komputera jest zbiorem „chaotycznych" znaczków, tekstów itp. Procesor pobiera te „krzaczki" z pamięci i je wykonuje. Jak on wie, które ma pobrać, w jakim czasie oraz dlaczego akurat te, a nie inne? Otóż każda komórka pamięci ma dokładnie określone „współrzędne", czyli swój adres. Mimo że nie jest to adres w tradycyjnym tego słowa znaczeniu, jednakże równie precyzyjnie wskazuje on miejsce komórki w odniesieniu do całej pamięci.

W tym miejscu powiedzmy jeszcze parę słów na temat sposobu przechowywania liczb w pamięci. Każda komórka pamięci ma wielkość 8 bitów (1 bajt) i ściśle określony adres. Zazwyczaj większość operacji wykonywanych jest jednak na dłuższych liczbowych porcjach aniżeli 1 bajt, na porcjach dwubajtowych, cztero- czy nawet dziesięciobajtowych. Gdy do pamięci wpiszemy na przykład liczbę 1234, wówczas cyfry 34 będą występować „wcześniej", pod młodszym adresem, niżeli cyfry 12. Na pierwszy rzut oka wydawałoby się, że powinno być odwrotnie, ale jednak tak nie jest. Taki rodzaj przechowywania informacji w pamięci nazywa się przechowywaniem odwrotnym. Podczas pracy z *dłuższymi* niż 1 bajt liczbami musimy uważać, aby nie popełnić błędów związanych z tego rodzaju przechowywaniem informacji. Jest w komputerze taki rodzaj pamięci, która sposobem swej organizacji przypomina stos talerzy czy książek. Zróbmy doświadczenie z książkami. Ułóżmy stos książek, na przykład 10. Po jakimś czasie chcemy wyjąć drugą od spodu książkę. Co wobec tego robimy? Na pewno nie wyciągamy jej ze stosu na siłę, bo stos zawaliłby się. Książkę tę trzeba uzyskać normalnie, zdejmując kolejne książki z wierzchołka, aż do niej dojdziemy. Proszę zwrócić uwagę na oczywistą właściwość stosu: książka położona jako ostatnia, zostanie zdjęta jako pierwsza. Tłumacząc to na język stosu komputerowego, możemy powiedzieć: informacja zapisana jako ostatnia zostanie odczytana jako pierwsza, i odwrotnie.

Rozdział 6
Przestrzeń adresowa procesorów 8086/8088; adresowanie

Procesor 8086 obsługuje urządzenia wejścia i wyjścia w dwojaki sposób – za pomocą rozkazów wejścia/wyjścia (rozkazów I/O; ang. input-output) i za pomocą adresowania pamięci. Niektóre wejścia i wyjścia urządzeń są kontrolowane przez porty, które określone są adresami wejść-wyjść.

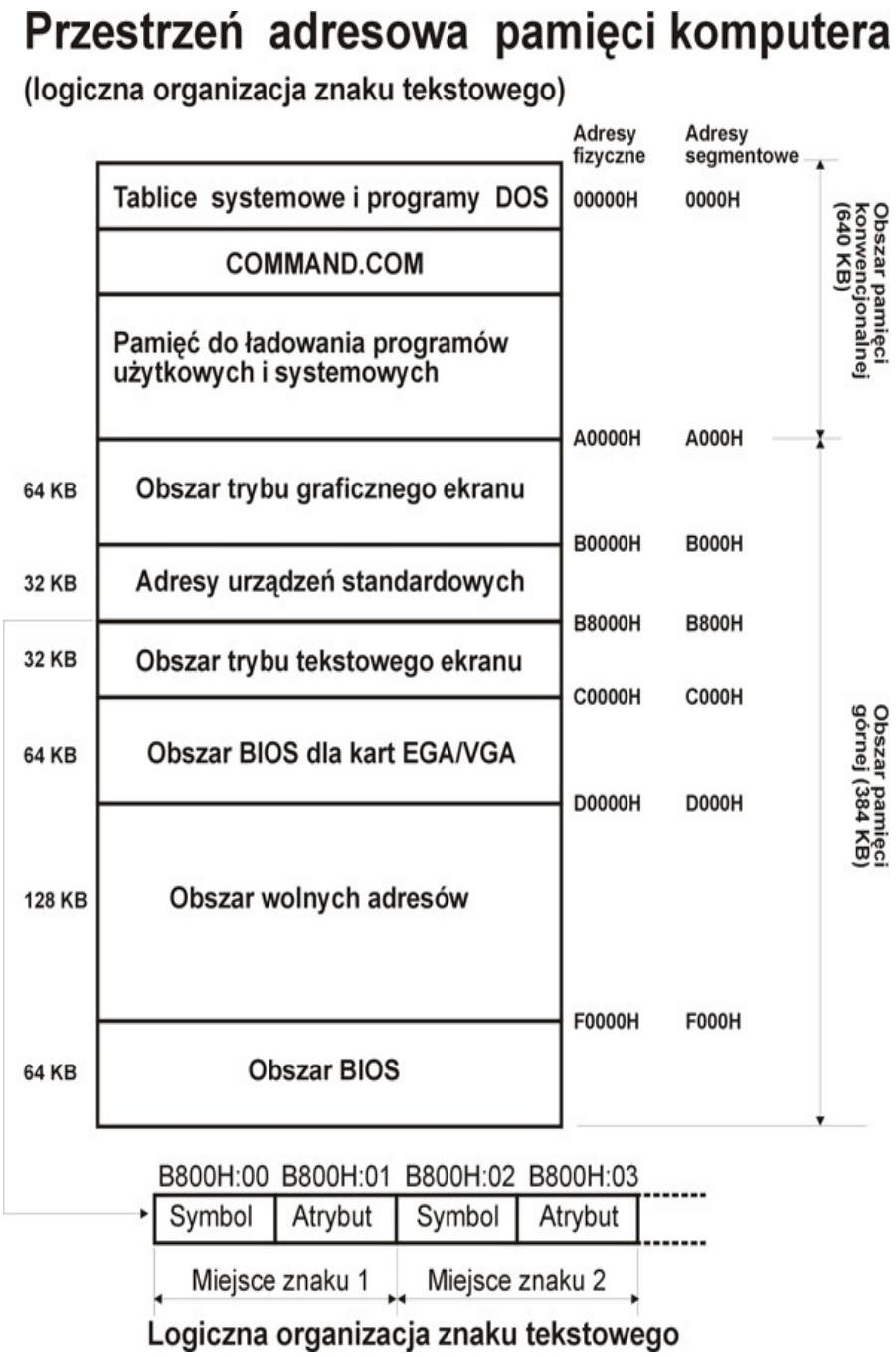


Rys. 6.1: Adresy pamięci i adresy I/O (wejścia/wyjścia) procesorów 8086/8088

Przestrzeń adresowa I/O jest o wiele mniejsza od 1 MB przestrzeń pamięci i dla procesorów 8086 ma ona wartość 64 KB (patrz rys. 6.1.); w praktyce to tylko 4 KB. Tym samym przestrzeń adresowa I/O nie jest używana do zapamiętywania wartości, ale raczej do zapewnienia właściwego sterowania urządzeniami wejścia/wyjścia, np.: modemem, drukarką, kartą dźwiękową itd. Adresy wejścia/wyjścia I/O mogą być dostępne za pomocą specjalnych rozkazów, IN i OUT, które tylko do tego celu służą. Na przykład rozkaz OUT DX,AL przesyła zawartość rejestru AL do portu I/O przez wybrany rejestr DX. Należy zaznaczyć, iż niektóre wyjścia urządzeń mają odwzorowania w pamięci i są one kontrolowane przez adresy pamięci, a nie przez porty I/O. Tak częściowo rzecz się ma z monitorami, które mają swe odwzorowania w 1 MB pamięci, a równocześnie, dla niektórych ich funkcji, można nimi sterować za pomocą rozkazów I/O.

Nie wnikając w szczegóły adresowania pamięci, które na pewnym, wyższym, etapie nauki programowania w Asemblerze może być dość złożone, powiemy tylko o elementarnych sposobach dotarcia do wybranego miejsca w komputerze, a chodzi tu głównie o dotarcie do pamięci operacyjnej, gdyż jak dostać się do portów opisano już powyżej.

Pamięć operacyjna komputera PC została podzielona na różnego rodzaju „kawałki” (patrz rys. 6.2.).



Rys. 6.2 :Przestrzeń adresowa pamięci komputera

Dane zawarte w tych różnych polach pamięci operacyjnej można przeglądać, niektóre zaś można nawet zmieniać. Gdy nawet wiemy co i gdzie należy zmieniać to zmian tych trzeba zawsze dokonywać w sposób bardzo rozważny, bo w najlepszym razie zawiesimy system. Niestety istnieje też nawet groźba uszkodzenia pewnych elementów sprzętowych, na przykład *grzebanie* na ślepo w rejestrach sterowników sprzętowych, monitora czy dyskowych, może spowodować ich fizyczne uszkodzenia.

W kwestii adresowania pamięci należy jeszcze dodać rzecz następującą: w celu ułatwienia procesorowi, no i programistom, poruszania się po jednomegabajtowej pamięci podzielono ją na 16 segmentów, w każdym segmencie jest po 65536 (64 KB) jednobajtowych komórek. Adres dowolnej komórki pamięci składać się będzie z dwóch części – numeru segmentu i położenia komórki w tym segmencie, nazywanego offsetem. Pełny adres komórki pamięci tworzy para rejestrów CS:IP jako SEGMENT i OFFSET.

Rozdział 7
Przerwania; wektory przerwań

Przerwania są to działania, za pomocą których zewnętrzne układy – w stosunku do jednostki centralnej, CPU, (ang. Central Procesor Unit) – sygnalizują zajście jakiegoś zdarzenia (np. wciśnięcia klawisza) i żądają określonego działania. W chwili, gdy zachodzi przerwanie, sterowanie nad komputerem przejmuje program obsługi przerwań, który zwykle znajduje się w pamięci ROM. Program ten przywoływany jest przez załadowanie jego adresu segmentowego do odpowiednich rejestrów procesora; przypomnijmy, iż pełny adres zawarty jest w parze rejestrów CS i IP. Adresy segmentowe potrzebne do zlokalizowania programów obsługi przerwań nazwane zostały wektorami przerwania.

Wektory przerwań ustawione są w stan początkowy w czasie uruchamiania komputera i wskazują na programy obsługi przerwań zawarte w pamięci ROM (ROM – pamięć tylko do odczytu). Te wektory przerwań przechowywane są w postaci tabeli na początku zwykłej pamięci RAM, cztery bajty na każdy wektor przerwań (RAM – pamięć do odczytu i zapisu, rozciąga się ona na obszarze 640 KB, od 0 do 640 KB). Wartości tych bajtów zapisywane są w pamięci w sposób odwrotny (np. odczytany w tej tabeli jednolity ciąg czterobajtowy:

54FF00F0 należy odczytać jako parę F000:FF54 (SEGMENT:OFFSET) stanowiącą adres komórki pamięci gdzie znajdować ma się jakiś ważny program systemowy wbudowany tam już od początku istnienia komputera.

O ile tego programu nie można zmienić czy zamazać, o tyle można zmienić w tabeli wektorów przerwań adresy, które wskazują gdzieś tam, w głąb pamięci komputera, na różne ważne dla systemu programy, o krótszym czy dłuższym kodzie. Można też programowo zamazać fragment lub całą tabelę wektorów przerwań, ale wówczas zawiesimy system, nawet jak to zrobimy w sesji Windows. I wreszcie można w sposób całkiem świadomy napisać samemu program, który na czas jakiś podmieni „stary" program systemowy z pamięci ROM na aktualnie napisany przez nas, umiejscawiając go w pamięci RAM. Cała tabela wektorów przerwań zawiera 255 adresów wskazujących na te różne programy. Używając w Asemblerze przerwań, będziemy je zapisywać w postaci INT z numerem przerwania, na przykład INT 10H czy INT 21H itp., itd. Do podanych przykładów nieprzypadkowo wybrane zostały przerwania INT 10H i INT 21H. Przerwanie 10H (dziesiętnie 16) jest przerwaniem należącym do systemu BIOS obsługującym ekran monitora, natomiast przerwanie 21H (dziesiętnie 33) jest przerwaniem należącym do systemu operacyjnego DOS; system Windows nie jest oparty na *filozofii* przerwań programowych, aczkolwiek istnieje oddziaływanie pomiędzy tablicą wektorów przerwań a środowiskiem systemu Windows.

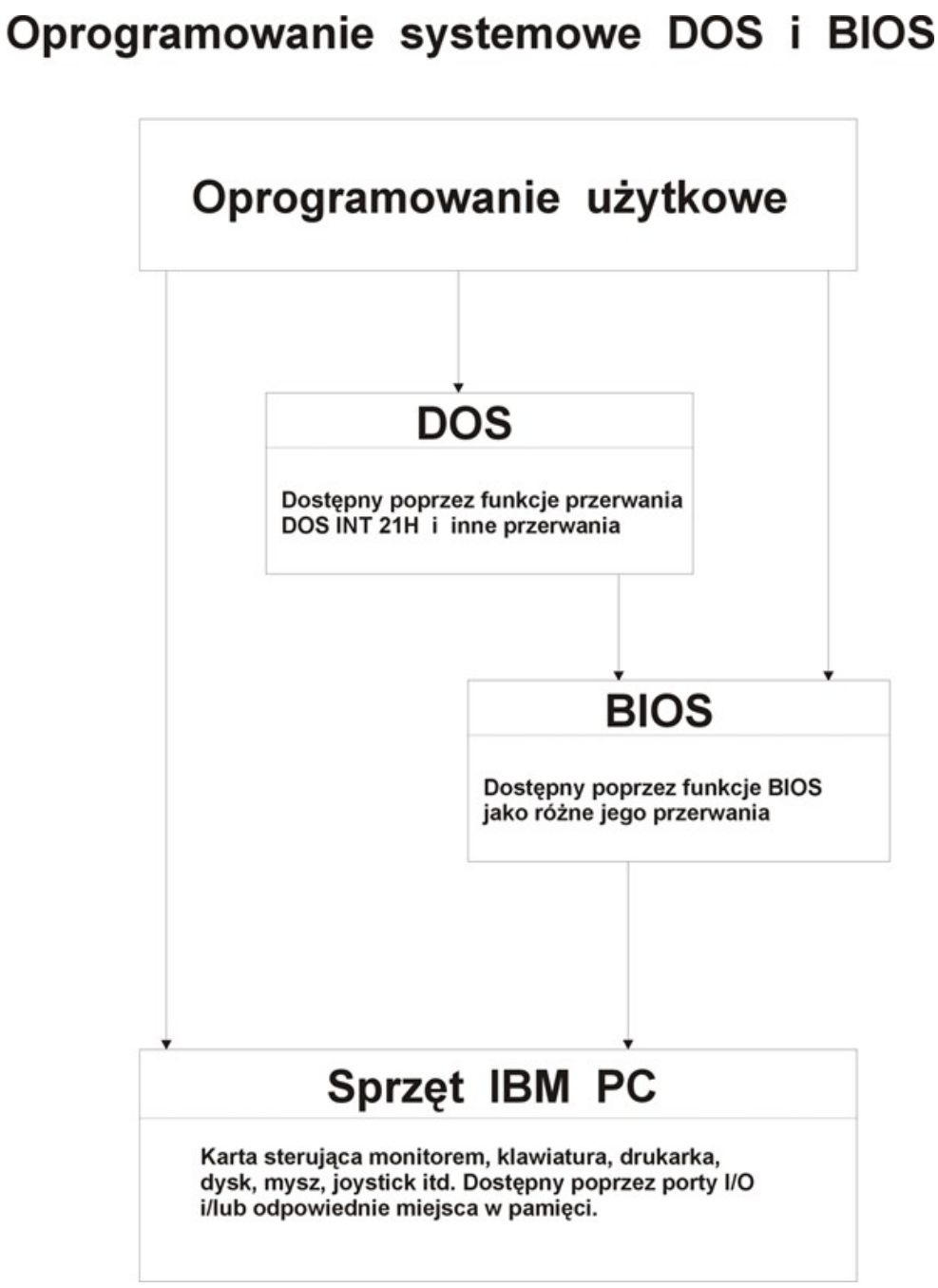
Rozdział 8
Systemy: BIOS, DOS, Windows

Oprogramowanie systemowe kieruje całą złożonością połączeń występujących pomiędzy poszczególnymi urządzeniami. Dwoma głównymi składnikami tego oprogramowania są BIOS (Basic Input/Output System) i DOS (ew. Windows). Programy zawarte w pamięci BIOS logicznie umieszczone są między naszymi programami, wraz z DOS/Windows, a sprzętem. BIOS z jednej strony otrzymuje od (naszych) programów wykonania standardowych usług związanych z obsługą urządzeń wejścia/wyjścia. Usługi te wzywane są przez programy za pomocą kombinacji numeru przerwania (przerwania BIOS) i numeru usługi.

Użycie tej kombinacji wskazuje na rodzaj usługi, np. obsługa drukarki. W drugą stronę BIOS komunikuje się z urządzeniami komputera, monitorem, dyskami itd., używając odpowiednich dla danego urządzenia kodów rozkazów. Od tej strony BIOS obsługuje też przerwania sprzętowe, które są generowane przez urządzenia „chcące zwrócić na siebie uwagę". Na przykład, naciśnięcie klawisza powoduje, że klawiatura generuje przerwanie, by o tym co zaszło, zaraz zawiadomić BIOS.

Oprogramowanie BIOS zapisane jest w kostce pamięci ROM, znane pod nazwą ROM-BIOS (Read Only Memory – Basic Input/Output System) i można odczytać je tylko jako ciąg asemblerowych rozkazów. DOS (Disc Operating System) ew. Windows jest programem, który kontroluje komputer i jego zasoby od momentu jego włączenia aż do wyłączenia.

Poprzez funkcje DOS programy użytkowe mogą czytać pliki, zapisywać je do pamięci, kontrolować naciskanie klawiszy klawiatury, uruchamiać inne programy, ustawiać datę i czas. Funkcji DOS używa się w celu wsparcia operacji związanych ze wprowadzaniem plików przy użyciu klawiatury, wyprowadzeniem pliku na ekran monitora bądź na drukarkę. Tam gdzie tylko powinno użyć się funkcji DOS należy zdecydowanie ich używać, aczkolwiek w niektórych przypadkach wyraźnie trzeba użyć funkcji BIOS.



Rys.8.1 Oprogramowanie systemowe BIOS i DOS

BIOS odmiennie niż programy DOS nie jest ładowany z dysku, lecz jest zapamiętany w pamięci ROM (ang. Read Only Memory), z której można tylko go odczytywać, chociaż nowe rodzaje tzw. flash-BIOS mogą być programowalne. BIOS jest oprogramowaniem komputera PC położonym najniżej; funkcje BIOS uzupełniają DOS w zakresie kontroli nad sprzętem.

Ze względu na pewne różnice w oprogramowaniu BIOS-ów powinniśmy raczej używać funkcji DOS niż BIOS, by tym samym uniknąć konfliktów programowych dla różnych modeli komputerów PC. Spośród wielu zastosowań funkcji BIOS jednym z nich jest użycie go w celu sterowania monitorem ekranowym. Tylko przez przywoływanie funkcji BIOS można ustawić tryb pracy monitora, mieć kontrolę nad kolorami, sposobem wyświetlania itp.

Pełna wersja ebooka dostępna pod:

http://www.escapemag.pl/293801-asembler-od-podstaw