# LISTA ROZKAZÓW i TRYBY ADRESOWANIA

## Lista rozkazów

- Rozkazy tworzące listę rozkazów można podzielić na kilka podstawowych grup, w zależności od ich przeznaczenia:
  - rozkazy przesłań, kopiowania,
  - rozkazy arytmetyczne i logiczne,
  - rozkazy sterujące wykonywaniem programu - skoków, obsługi pętli, wywołań i powrotów z podprogramu
  - wykonujące operacje na ciągach słów
  - rozkazy wejścia/wyjścia
  - rozkazy sterujące pracą procesora,
  - inne.
- Rozkazy mikroprocesorów mogą być:
  - jednobajtowe 8-bitów,
  - jednosłowowe np. 12, 14-bitowe, 24-bitowe
  - wielobajtowe, 16-bitowe, 32-bitowe lub więcej

# Mnemonik w języku asemblera

- Aby uprościć programowane w języku asemblera wprowadzono **mnemonik**, który zastępuje liczbę reprezentującą rozkaz procesora, słowem-kodem.
- Za pomocą kilku liter, kod-słowo, które oznacza konkretną czynność procesora zastępuje liczbę. Np. dla 8051 słowo "add" przedstawia rozkaz dodawania (ang. dodaj), czy "sub" (ang. subtract - odejmij).
- Mnemoniki powstały aby wyeliminować konieczność programowania komputerów za pomocą wpisywania liczb, które są naturalnym kodem którego używa procesor.
- Zamianą mnemoników na liczby (kod maszynowy) zajmują się odpowiednie programy - assemblery.
- Wprowadzenie mnemoników miało wielkie zalety. Pozwoliło procesowi programowania stać się łatwiejszym i mniej zawodnym, jako że ludzie łatwiej operują na słowach wyrażających procesorowi polecenia niż na liczbowych kodach, których zapamiętanie jest czasochłonne, a używanie dużo mniej efektywne i bardziej podatne na błędy.
- Zestawy mnemoników różnią się w zależności od typu procesora i producenta, choć część pozostaje zazwyczaj niezmienna, jak dodawanie, odejmowanie, mnożenie "mul" (od ang. multiply), czy dzielenie "div" (ang. divide).
- Producent oferujący mikroprocesor lub mikrokontrolera w dokumentacji dokładnie opisuje poszczególne rozkazy podając mnemoniki, efekt działania rozkazu i konkretną wartość liczbową rozkazu dla konkretnego trybu adresowania.
  - ADD A, Rn — A ←A + Rn — 0010 1rrr
  - SUBB A, @R0/1 — A ←A – (R0/1 ) – CY — 96 lub 97
  - INC A — A ←A + 1 — 04

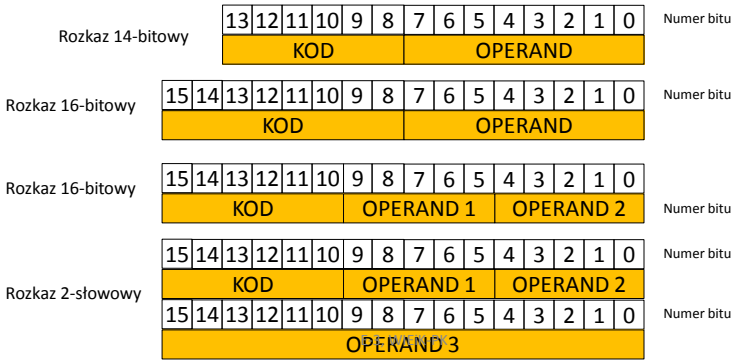


# Format rozkazu

- Formatem rozkazu nazywamy sposób rozmieszczenia informacji w kodzie rozkazu.
- Rozkazy, jak każdy inny rodzaj informacji w systemie mikroprocesorowym, są przechowywane w postaci kodów binarnych. Kod rozkazu musi zawierać informacje niezbędne do jego poprawnej realizacji. Informacje te muszą być rozmieszczone w rozkazie w określony sposób.
- Kod rozkazu musi zawierać określenie rodzaju wykonywanej operacji, czyli tak zwany kod operacji.
- Kod operacji musi być określony w początkowej części (pierwszym bajcie lub bajtach) kodu rozkazu w celu określenia, w jaki sposób ma przebiegać dalsza realizacja rozkazu przez mikroprocesor.
- Kod rozkazu może zawierać operandy (argumenty) i/lub adresy operandów wykonywanych operacji (dotyczy to także adresów wyników). Oczywiście w przypadku rozkazów wymagających argumentów informacja ta musi być zawarta w rozkazie.

# Format rozkazu

- Rozkazy mogą być liczbą 1-bajtową lub wielokrotnością bajtu (2-bajtowe, 3-bajtowe, …)
- W procesorach typu Harvard jednostka CPU może być np. 8-bitowa ale format rozkazu może być np. 12-bitowy, 14-bitowy lub 16-bitowy. Rozkazy wtedy są 1-słowowe (ang. one-word instruction ) lub wielosłowowe (ang. two-word instruction, … )
- Szerokość rozkazu może być większa od szerokości szyny danych CPU






# Rozkazy przesłań, kopiowania

- Wpisz do rejestru konkretną wartość (**MOV, LOAD, LD**)
- Wpisz do pamięci konkretną wartość
- Ładuj do rejestru roboczego zawartość pamięci lub ładuj do pamięci zawartość rejestru roboczego (**ST**)
- Kopiowanie (przesuwanie) danych z jednego miejsca pamięci w inne miejsce (**MOV**)
- Kopiowanie (przesuwanie) danych z układów (**IN, OUT**) wejścia/wyjścia do pamięci lub do rejestrów roboczych
- Zamiana zawartości dwóch rejestrów lub komórek pamięci (**XCH**)
- Zamiana bitów w rejestrze lub komórce pamięci (**SWAP**)

# Rozkazy logiczne

- Rozkazy logiczne operujące na bitach
  - AND, OR, XOR, NOT, ustaw bit, zeruj bit (**AND, OR, XOR, ANL, ORL, XRL, CPL, SETB, CLR, SBI, CBI, TEST, NOT**)
- Rozkazy logiczne operujące na bajtach, słowach
  - AND, OR, XOR, NOT, przesunięcie w prawo, lewo (**RL, RRC, ROL, ROR, LSL, SHL, SRL**)



# Rozkazy arytmetyczne

- dodawanie, odejmowanie, (**ADD, ADC, SUB, SBC**)
- mnożenie, dzielenie, (**MUL, DIV**)
- zwiększanie lub zmniejszanie o 1, (**INC, DEC**)
- porównywanie liczb, (**CMP, CP, CPI**)
- zamiana (konwersja) liczb, (**DA, CBW, CWD**)
- rotacje, przesunięcie, (**RL, RRC, ROL, ROR, LSL**)
- negacja lub przekształcenie na liczbę w kodzie uzupełnień do dwóch – U2 (**NEG**)
- Operacje arytmetyczne mogą być wykonywane:
  - na liczbach bez znaku,
  - na liczbach ze znakiem,
  - na liczbach kodowanych (np. w kodzie BCD, kodzie U2)
  - na liczbach stałoprzecinkowych, zmiennoprzecinkowych

# Skoki warunkowe i bezwarunkowe

- Skok warunkowy – skok do rozkazu poprzedzonego etykietą jeśli warunek jest prawdziwy
- Skoki warunkowe (*ang. jump, branch*) od bitu (1 lub 0), bajtu lub słowa (>, <, =, =>, <=, ≠, =0, ≠0) (**JB, JNB, JZ, JNZ, SBIC, SBIS, BRMI**)
- Skoki od przepełnienia, parzystości
- Skoki bezwarunkowe w konkretne miejsce programu
  - skoki krótkie i skoki dalekie, wywołanie podprogramu, zgłoszenie przerwania (**SJMP, LJMP, JMP, CALL, LCALL, ICALL, GOTO, SKIP**)
  - wyjście z podprogramu, wyjście z programu obsługi przerwania (**RET, RETURN, RETI**)



# Pętle programowe

- Rozkazy realizujące pętle programowe lub wspomagające realizację pętli (**LOOP**)
- Rozkazy kończące pętle (**LOOPZ, LOOPNZ**)

## Rozkazy sterujące pracą procesora

- Przełączanie trybów pracy procesora (**SLEEP**)
- Przełączanie procesorów, np. włączenie FPU(**ESC**)
- Zablokowanie dostępu do magistrali (**LOCK**)
- Zatrzymanie procesora (**HALT, HLT**)
- Rozkaz nic nie rób (**NOP**)
- Rozkaz oczekiwania na zdarzenie (**WAIT**)
- Rozkazy operujące na znacznikach w rejestrze stanu – ustawianie i zerowanie bitu (**CLI, CLD, STD, SDI**)
- Zerowanie watchdoga (**WDR, CLRWDT**)



## Złożone instrukcje

- Zapisywanie wielu rejestrów na stos lub ściąganie ze stosu jednym rozkazem
- Kopiowanie (przesuwanie) dużych bloków pamięci
- Złożone instrukcje na liczbach zmiennoprzecinkowych,
- Obliczanie wybranych funkcji matematycznych (np. sin(x), cos(x), pierwiastek)
- Wykonywanie elementarnych instrukcji testowania i ustawiania
- Instrukcje, które korzystają z jednostki ALU i pobierają dane z pamięci a nie z rejestrów
- Instrukcje typu SIMD (wiele danych jedna instrukcja),
- instrukcje typu MMX

## Przetwarzanie łańcuchów danych

**Łańcuch** to ciąg kolejnych bajtów, słów lub podwójnych słów w pamięci. Bloki te mogą się składać z: wartości liczbowych wartości lub wartości alfanumerycznych

- Wypełnianie łańcucha wartością (**STOSB, STOSW**)
- Kopiowanie łańcuchów (**MOVSB, MOVSW**)
- Odczyt elementu z łańcucha (**LOADSB, LOADSW**)
- Porównywanie łańcuchów (**CMPSB, CMPW**)
- Przeszukanie łańcucha (**SCASB, SCASW**)



## Rozkazy operacji na stosie

**STOS** jest to miejsce w pamięci danych do przechowywania pewnych danych takich jak rejestry lub zawartości komórek pamięci.

Stos - liniowa struktura danych, znaczeniowo odpowiadająca nazwie: dane dokładane są na wierzch stosu, i ściągane również z wierzchołka stosu. Stosuje się też określenie stosu jako pamięć typu LIFO (ang. Last In First Out).

Operacje na stosie:

- Deklarowanie segmentu stosu – początku, rozmiaru
- Odkładanie danych na stos (**PUSH**)
- Zdejmowanie danych ze stosu (**POP**)

# Tryby adresowania

- Przez tryby adresowania rozumie się sposób wskazywania na argumenty wykorzystywane w trakcie wykonywania instrukcji.
- Standardowe mikrokontrolery wykorzystują kilka różnych sposobów adresowania. Jednak ciągły rozwój techniki mikroprocesorowej powoduje wzbogacanie o nowe możliwości adresowania.
- Nowe mikrokontrolery typu embedded posiadają tak rozbudowane możliwości adresowania jak procesory stosowane we współczesnych komputerach.



# Tryb adresowania

- Argumenty operacji mogą znajdować się w kodzie programu, w rejestrach procesora lub w pamięci.
- W zależności od miejsca, gdzie znajdują się argumenty, stosuje się odpowiednie rozkazy, które występują w różnych odmianach. Mówimy, że są to rozkazy o różnych trybach adresowania (ang. addressing mode).
- Każdy układ mikroprocesorowy ma ustaloną liczbę trybów adresowania zależną od architektury procesora.
- Nie każdy rozkaz ma wszystkie tryby adresowania

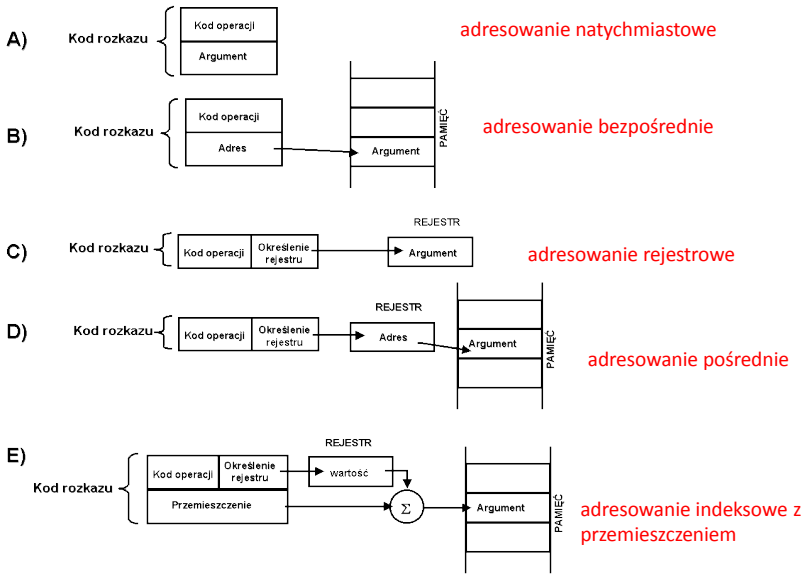
# Tryb adresowania

**Tryb adresowania** (z angielskiego *addressing mode*), sposób obliczania adresów argumentów rozkazów określony przez część operacyjną rozkazu.

- Można rozróżnić następujące podstawowe tryby adresowania:
  a) adresowanie natychmiastowe, w którym argument rozkazu zawarty jest w kodzie rozkazu,
  b) adresowanie bezpośrednie, w którym kod rozkazu zawiera adres komórki pamięci przechowującej argument rozkazu,
  c) adresowanie rejestrowe, w którym w kodzie rozkazu określony jest rejestr zawierający argument rozkazu,
  d) adresowanie pośrednie (rejestrowe pośrednie), w którym kod rozkazu zawiera określenie rejestru, bądź rejestrów zawierających adres komórki pamięci z argumentem rozkazu,
  e) adresowanie indeksowe z przemieszczeniem, w którym adres argumentu przechowywanego w pamięci obliczany jest jako suma zawartości rejestru i wartości (tzw. przemieszczenia) określonych w kodzie rozkazu.
  f) adresowanie pośrednie z predekrementacją
  g) adresowanie pośrednie z postinkrementacją



# Podstawowe tryby adresowania




Opis rysunkowy trybów adresowania

# Tryb natychmiastowy

- W **trybie natychmiastowym** pole adresowe zawiera bezpośrednio **operand** (ang. **operand**) czyli daną dla rozkazu. Długość operandu zależy od kodu operacji lub od kodu operacji i dodatkowego pola sterującego w rozkazie.
- Ten tryb adresowania stosujemy, gdy np. chcemy bezpośrednio z rozkazu ( bez odwoływania się do pamięci) załadować do rejestru prostą daną - bajt lub parę bajtów.

| 8051 | 8086 | AVR | PIC | ARM |
|---|---|---|---|---|
| mov A,#23h | mov AX,20 | ADDI r1, 15 | | |
| Add A, #10 | mov BH,4Ah | ORI r1, 30 | | |
| Mov DPTR,#1FFAh | | ADDI r1, 124 | | |
| | | | | |



# Adresowanie bezpośrednie

- **Adresowanie bezpośrednie** jest najbardziej podstawowym trybem adresowania. W tym trybie zawartość pola adresowego stanowi już finalny adres argumentu rozkazu w pamięci operacyjnej i nie podlega przekształceniu.
- Ten tryb stosujemy, gdy nie zależy nam na tym, aby nasz program był przesuwalny w pamięci operacyjnej, lecz jest przeznaczony do wykonania przy zapisie w ściśle określone miejsce w pamięci.

| 8051 | 8086 | AVR | PIC | ARM |
|---|---|---|---|---|
| mov A,23h | mov AX, CS:[40] | OR R18,R19 | | |
| Add A, P0 | mov AX, [40] | INC R4 | | |
| mul AB | | OUT PortA,R17 | | |
| | | ADD R16,R17 | | |

# Adresowanie pośrednie

- Przy **adresowaniu pośrednim** rozkaz zawiera adres komórki pamięci operacyjnej, w której zawarty jest finalny adres operandu rozkazu. W tym przypadku komórka pamięci wskazana przez adres rozkazu pośredniczy w określeniu finalnego adresu.
- Ten tryb stosujemy, gdy chcemy, aby finalny adres operandu rozkazu mógł być dynamicznie wstawiony do komórki pośredniczącej w adresowaniu w czasie wykonywania programu. Może tak być, gdy ten adres zależy od jakichś testów na wyniku operacji poprzedzającego rozkazu.



# Adresowanie indeksowe

- **Adresowanie indeksowe** jest inaczej nazywane **modyfikacją adresu przez indeksowanie**. W tym trybie wykorzystuje się specjalne rejestry procesora tzw. **rejestry indeksowe** (ang. **index registers**), które zawierają przesunięcie, który trzeba dodać do adresu istniejącego w rozkazie aby wyliczyć adres finalny operandu.
- Ten tryb adresowania pozwala przesunąć adres zawarty w rozkazie o wartość rejestru indeksowego. Używając tego trybu we wszystkich rozkazach programu, można osiągnąć możliwość wykonania programu przy załadowaniu w dowolne miejsce pamięci. W tym celu należy napisać program wstawiając do rozkazów programu adresy, które odpowiadają umieszczeniu pierwszej instrukcji programu pod adresem zerowym w pamięci. Następnie wiedząc, pod jakim finalnym adresem jest umieszczony pierwszy rozkaz programu, umieszczamy ten adres w rejestrze indeksowym.
- Dzięki operacji indeksowania rozkazów programu, wszystkie adresy operandów zostaną przesunięte o tę sama wartość - stad nazwa zawartości rejestru indeksowego : przesunięcie. Taka organizacja przesuwalności programu w pamięci nosi nazwę dynamicznej relokacji programu w pamięci.
  Rejestrów indeksowych w procesorze jest zwykle wiele, gdyż mogą one być przydzielone różnym programom lub fragmentom programów, które są wykonywane w tym samym okresie czasu. W tym przypadku, rejestr indeksowy do użycia, może być określony przy pomocy specjalnego pola wspomagającego kodowanie trybu adresowania w innym polu rozkazu.

# Adresowanie względne

- **Adresowanie względne** polega na modyfikacji adresu zawartego w rozkazie przez aktualną zawartość licznika rozkazów.
- Ten tryb adresowania dostarcza innego sposobu osiągnięcia dynamicznej przesuwalności adresów dostępu do danych, tj. gdy nie chcemy lub nie możemy znać przesunięcia całości programu w stosunku do adresu zerowego.
- Przy tym trybie adresowania, finalny adres danej jest wyliczany względem bieżącej zawartości licznika rozkazów, a więc do rozkazu wstawiamy przesunięcie danej w programie względem adresu następnego rozkazu, np. n komórek w przód lub w tył. Musimy tylko zapewnić, aby dana została umieszczona w spodziewanym miejscu pamięci, a ściślej w spodziewanej odległości od rozkazu, który z tej danej korzysta. Można to osiągnąć przez zaplanowanie w jakiej odległości od danego rozkazu, będzie umieszczony obszar danych programu.



# Adresowanie pośrednie indeksowe

- **Adresowanie pośrednie indeksowe** zapewnia jednoczesną możliwość zastosowania w programie adresowania pośredniego z modyfikacją adresu odczytanego z komórki pośredniczącej poprzez zawartość rejestru indeksowego.
- Umieszczony w rozkazie adres wskazuje na komórkę przechowującą adres danej, który może być tam wstawiany dynamicznie jako wynik obliczeń programu. Do tego adresu stosowane jest następnie indeksowanie poprzez zawartość rejestru indeksowego. Zawartość rejestru indeksowego może też być wstawiona dynamicznie w wyniku działania innej części programu. W zależności od zawartości rejestru indeksowego trafiamy zatem do innej komórki zawierającej operand.
- Adresowanie pośrednie indeksowe pozwala na dwupoziomowe dynamiczne określanie adresu danych w wyniku obliczeń wykonanych w poprzedzających fragmentach programu, metodą wpisania pewnego adresu bazowego a następnie modyfikacji tego adresu bazowego przez rejestr indeksowy.

# Adresowanie indeksowe pośrednie

- **Adresowanie indeksowe pośrednie** zapewnia najpierw modyfikację adresu zawartego w rozkazie przez zawartość rejestru indeksowego a następnie tak otrzymany adres jest stosowany do wskazania komórki pamięci, w której jest przechowywany finalny adres operandu rozkazu.
- Adresowanie indeksowe pośrednie pozwala na dynamiczne określanie adresów danych w programie w czasie wykonania programu, przy zapewnieniu przesuwalności w pamięci całego programu. Dla tego celu, program piszemy począwszy od adresu zerowego w pamięci stosując adresowanie indeksowe do wszystkich rozkazów z wyjątkiem tych, które działają na danych wybieranych dynamicznie. Te rozkazy adresujemy poprzez użycie adresowania pośredniego indeksowego, umieszczając adres komórki pośredniczącej (względem adresu zero) w bieżącym rozkazie.
- Po załadowaniu programu do pamięci, wpisywana jest do rejestru indeksowego wartość przesunięcia początku programu w stosunku do adresu zerowego ( robi to program ładujący). W trakcie wykonywania programu wszystkie adresy programu , łącznie z tymi, gdzie zastosowano adresowanie pośrednie oraz tymi. które wpisują adres danej do komórki pośredniczącej, zostaną zmodyfikowane.



# Adresowanie rejestrowe

- **Adresowanie rejestrowe** stosuje się, gdy dana dla rozkazu jest przechowywana w rejestrze. Część adresowa rejestru zawiera wtedy jedno lub więcej pól, w których znajdują się identyfikatory rejestrów.
- Adresowanie rejestrowe jest często stosowane w tym samym rozkazie razem z innymi trybami adresowania dotyczącymi pamięci operacyjnej. Wówczas rozkaz dotyczy zarówno pamięci operacyjnej jak i rejestrów procesora.

| 8051 | 8086 | AVR | PIC | ARM |
|---|---|---|---|---|
| add A,R0 | mov AX,CX | | | |
| mov A, R0 | | | | |
| | | | | |
| | | | | |

# Adresowanie pośrednie rejestrowe

- **Adresowanie pośrednie rejestrowe** polega na tym, że jako miejsce pobrania finalnego adresu operandu rozkazu stosuje się rejestr procesora, którego identyfikator umieszczony jest w polu adresowym rozkazu.
- Przy pomocy tego trybu adresowania można dynamicznie określić finalny adres operandu poprzez odpowiednie załadowanie zawartości rejestru, w zależności od przebiegu obliczeń w programie.



# Tryby adresowania 8051

- **Adresowanie rejestrowe** – operandem instrukcji jest jeden z rejestrów R0...R7 z aktywnego banku rejestrów, akumulator , rejestr B lub wskaźnik danych DPTR
- **Adresowanie bezpośrednie** – operand jest jawnie podanym 8 – bitowym adresem rejestru SFR, rejestru dolnego obszaru RAM lub bitu adresowalnego bezpośrednio
- **Adresowanie natychmiastowe** – operandem jest stała (8 lub 16 bitowa) umieszczona w kodzie programu (pamięć ROM)
- **Adresowanie pośrednie** – określenie adresu za pomocą zawartości R0 lub R1 (z aktywnego banku rejestrów) – odwołanie do przestrzeni 256 bajtów
- **Adresowanie sumą zawartości rejestru bazowego i indeksowego** – służy do kopiowania bajtu z pamięci programu.

## Tryby adresowania 8051

Mikrokontroler 8051 realizuje następujące tryby adresowania:

- natychmiastowe mov R0,#2Ah , mov 30h,#100 , add A, #1Ah, mov DPTR,#2FFAh
- rejestrowe mov A,R0 , add A,R5
- bezpośrednie mov R0,2Ah , mov 30h,32h , add A,32h, clr A, rrc A
- pośrednie zawartością rejestru mov A,@R0 , mov @R1,P3 , add A, @R1 , movx @R0, A
- pośrednie sumą zawartości rejestru bazowego i indeksowego movc A, @A+PC , movx A, @A+DPTR , movx @A+DPTR ,A
- Rozkazy w 8051 mogą być 1, 2 lub 3-bajtowe

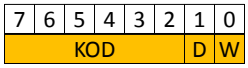


## Tryby adresowania 8051

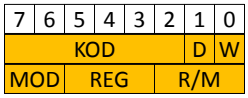
- **Immediate Addressing** **MOV A,#20h**
- **Direct Addressing** **MOV A,30h**
- **Indirect Addressing** **MOV A,@R0**
- **External Direct** **MOVX A,@DPTR**
- **Code Indirect** **MOVC A,@A+DPTR**

# Rozkazy i tryby adresowania 8086

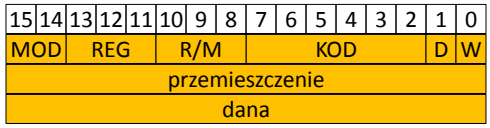
- Dla 8086 liczba bajtów rozkazu zależy od rodzaju rozkazu i może wynosić od jednego do sześciu. Pierwszy bajt zawiera sześciobitowy kod operacji oraz dwa bity.
  - Bit *D* określa kierunek transmisji (*0* - wynik operacji jest przesyłany z rejestru do pamięci, *1* - z pamięci do rejestru). W zależności od wartości tego bitu w rozkazie rozróżniane są operandy źródłowe i operandy przeznaczenia.
  - Bit W określa długość operandu danego rozkazu (0 - operacje bajtowe, 1 - operacje na słowie 16-bitowym).
- Jeżeli rozkaz jest wielobajtowy, to drugi bajt rozkazu określa sposób adresowania argumentów. Zawiera on trzy grupy bitów
- dwubitowa grupa *MOD* określa tryb adresowania
- trzybitowa grupa *REG* określa numer rejestru, w którym znajduje się operand
- trzybitowa grupa *R/M* określa sposób wyznaczenia miejsca operandu




Rozkaz 1-bajtowy




Rozkaz 2-bajtowy




Rozkaz 6-bajtowy



# Tryby adresowania 8086

Mikroprocesor 8086 realizuje następujące tryby adresowania:

- natychmiastowe
- rejestrowe
- bezpośrednie
- pośrednie
- bazowe
- indeksowe
- indeksowo-bazowe

# Tryby adresowania 8086

**Adresowanie natychmiastowe**

- W adresowaniu natychmiastowym argument pobierany jest bezpośrednio z rozkazu. W tym trybie wskazywany jest wyłącznie operand źródłowy. Np. **MOV AX, 20** – w rejestrze AX zostanie zapisana liczba 20.

**Adresowanie rejestrowe**

- W adresowaniu rejestrowym operandy znajdują się w rejestrach wewnętrznych mikroprocesora. Jeżeli operand znajduje się w pamięci, to zespół wykonawczy EY oblicza jego 16-bitowy adres (przesunięcie) wewnątrz segmentu. Zespół *BIU* oblicza adres rzeczywisty na podstawie otrzymanego przesunięcia (adresu efektywnego *EA*) i zawartości wybranego rejestru segmentowego. Np. **MOV AX, BX** – w rejestrze AX zostanie zapisana zawartość rejestru BX.

**Adresowanie bezpośrednie**

- W adresowaniu bezpośrednim adres operandu znajduje się bezpośrednio w rozkazie. Np. **MOV AX, [40]** – w rejestrze AX zostanie zapisana zawartość komórki pamięci (segment danych) o adresie 40.
- Standardowy adres operandu jest przesunięciem w segmencie danych (ds), można to nadpisać poprzez wskazanie innego segmentu. Np. **MOV AX, CS:[40]** - rejestrze AX zostanie zapisana zawartość z komórki pamięci (segment PROGRAMU(kodu)) o offsecie 40.

**Adresowanie pośrednie**

- W trybie adresowania pośredniego odwołujemy się do jednego z rejestrów roboczych procesora (np. BX) lub do komórki pamięci (np.19). W rejestrze (BX) zapisany jest numer komórki pamięci, do której trzeba sięgnąć aby odczytać tam zawarty adres i przenieść do drugiego rejestru (AX). Dla adresowania pośredniego z pamięci odczytujemy numer komórki pamięci z dwóch komórek (komórki 19 i komórki 20) w taki sposób, że zawartość tej pierwszej (19) stanowi ważniejszą część tego numeru , zaś zawartość drugiej komórki (20) mniej ważną część tego numeru. Dalej postępujemy podobnie jak przy adresowaniu pośrednim z rejestru – przenosimy zawartość do rejestru AX. Np. **MOV AX, [CX]** – w rejestrze AX zostanie zapisana zawartość komórki pamięci o adresie, który znajduje się w rejestrze CX.
- Wszystkie rejestry wskazują offset w segmencie danych (ds), poza rejestrem bp, który jest przesunięciem w segmencie stosu (ss). Można to nadpisać określając segment w rozkazie. Np. **MOV AX, ss:[CX]** - w rejestrze AX zostanie zapisana zawartość komórki pamięci z segmentu stosu o adresie, który znajduje się w rejestrze CX.



# Tryby adresowania 8086

**Adresowanie bazowe**

- Adresowanie bazowe jest to rodzaj adresowania pośredniego, gdzie rozkaz wskazuje na jeden z rejestrów bazowych *BX* lub *BP* i może zawierać 8-; lub 16-bitową wartość stanowiącą lokalne przemieszczenie. Adresem efektywnym jest suma zawartości rejestru bazowego i przemieszczenia. Np. **MOV AX, [BP]**

**Adresowanie indeksowe**

- Adresowanie indeksowe jest rodzajem adresowania pośredniego, gdzie adres efektywny jest sumą zawartości rejestru indeksowego *SI* lub *DI* i lokalnego przemieszczenia. Np. **MOV AX, [SI]**

**Adresowanie bazowo-indeksowe**

- W adresowaniu bazowo-indeksowym, adres efektywny jest sumą zawartości jednego z rejestrów bazowych, jednego z rejestrów indeksowych i lokalnego przemieszczenia. Np. **MOV AX, [SI+BP]**