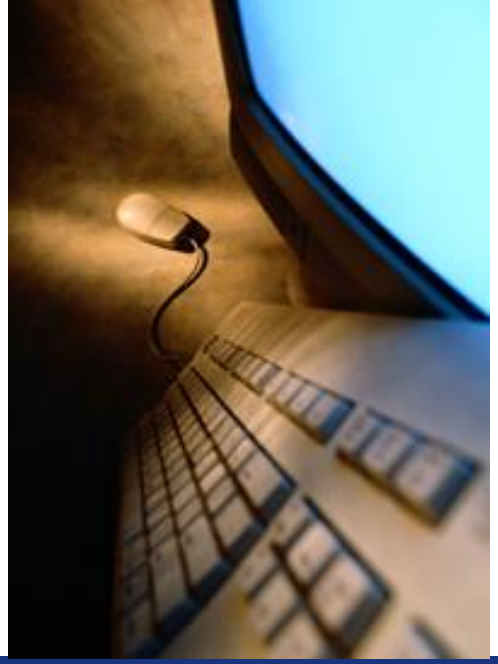
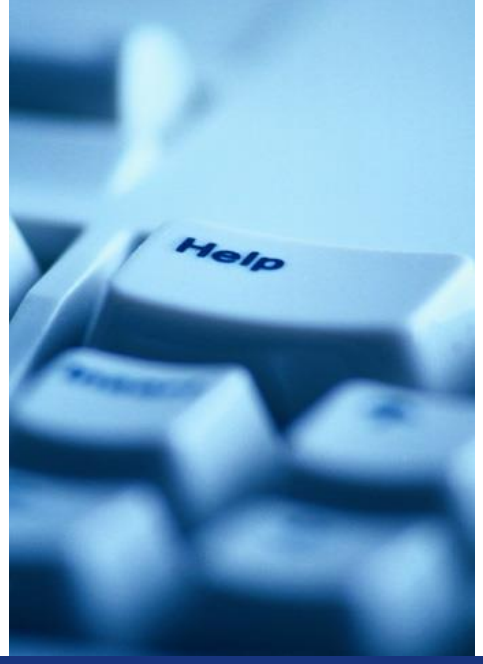
Instytut
Elektroniki
i Technik
Informacyjnych





Programowanie Niskopoziomowe

Wykład 2

Reprezentacja danych

dr hab. inż. Piotr Kisała, prof. PL

Plan wykładu

1. systemy liczbowe
2. organizacja danych
3. liczby ze znakiem i bez znaku
4. przesunięcia i obroty
5. arytmetyka zmiennoprzecinkowa
6. zestawy znaków

Wprowadzenie

Jaki system ?

- Większość nowoczesnych systemów komputerowych nie przedstawia wartości liczbowych używając systemu dziesiętnego używają systemu binarnego lub systemu "dopełnienia do dwóch". U2

System dziesiętny - powtórzenie

- **Uznawany mylnie za podstawowy, naturalny.**
- **Widząc liczbę zapisaną jako 123, nikt nie zastanawia się nad znaczeniem poszczególnych cyfr.**
- **Formalnie liczba 123 interpretowana jest w systemie dziesiętnym następująco:**
  - $1*10^2+2*10^1+3*10^0$
  - co daje następującą sumę 100+20+3

System dziesiętny - powtórzenie

- W pozycyjnym systemie liczbowym każda cyfra znajdująca się na lewo od znaku pozycji dziesiętnej (w języku polskim znakiem tym jest przecinek) reprezentuje iloczyn wartości z zakresu od zera do dziewięciu i rosnącej potęgi liczby dziesięć. Na przykład wartość **123,456** można rozpisać następująco:

  - $1*10^2 + 2*10^1 + 3*10^0 + 4*10^{-1} + 5*10^{-2} + 6*10^{-3}$
  - czyli
  - 100+20+3+0,4+0,05+0,006

System dwójkowy (binarny)

Najprostszy dla komputera, gdzie coś jest albo włączone, albo wyłączone.

System operuje na liczbach zwanych bitami (bit = binary digit = cyfra dwójkowa). Bit przyjmuje jedną z dwóch wartości 0 lub 1.

Na bajt składa się 8 bitów. Jednym bajtem można przedstawić 2^8=256 możliwości.

System dwójkowy (binarny)

Wartość pozycji	8 7 6 5 4 3 2 1 128 64 32 16 8 4 2 1
Podstawa potęga	$2^7 = 128$ $2^6 = 64$ $2^5 = 32$ $2^4 = 16$ $2^3 = 8$ $2^2 = 4$ $2^1 = 2$ $2^0 = 1$
Liczba używanych symboli	2
Symbole	0 , 1

System dwójkowy (binarny)

- Przeliczenie liczby zapisanej w systemie dwójkowym na dziesiętny jest proste. Podobnie jak w systemie dziesiętnym, każdą cyfrę mnożymy przez odpowiednią potęgę podstawy (podstawa wynosi 2 w systemie dwójkowym, 10 w systemie dziesiętnym.

$10110 = (1 \times 2^4 = 16) + (0 \times 2^3 = 0) + (1 \times 2^2 = 4) + (1 \times 2^1 = 2) +$
$(0 \times 2^0 = 0) = 22\ (16 + 0 + 4 + 2 + 0)$

Konwersja dziesiętny > binarny

liczba 192 > 11000000

192/2 = 96 z resztą 0
96/2 = 48 z resztą 0
48/2 = 24 z resztą 0
24/2 = 12 z resztą 0
12/2 = 6 z resztą 0
6/2 = 3 z resztą 0
3/2 = 1 z resztą 1
1/2 = 0 z resztą 1

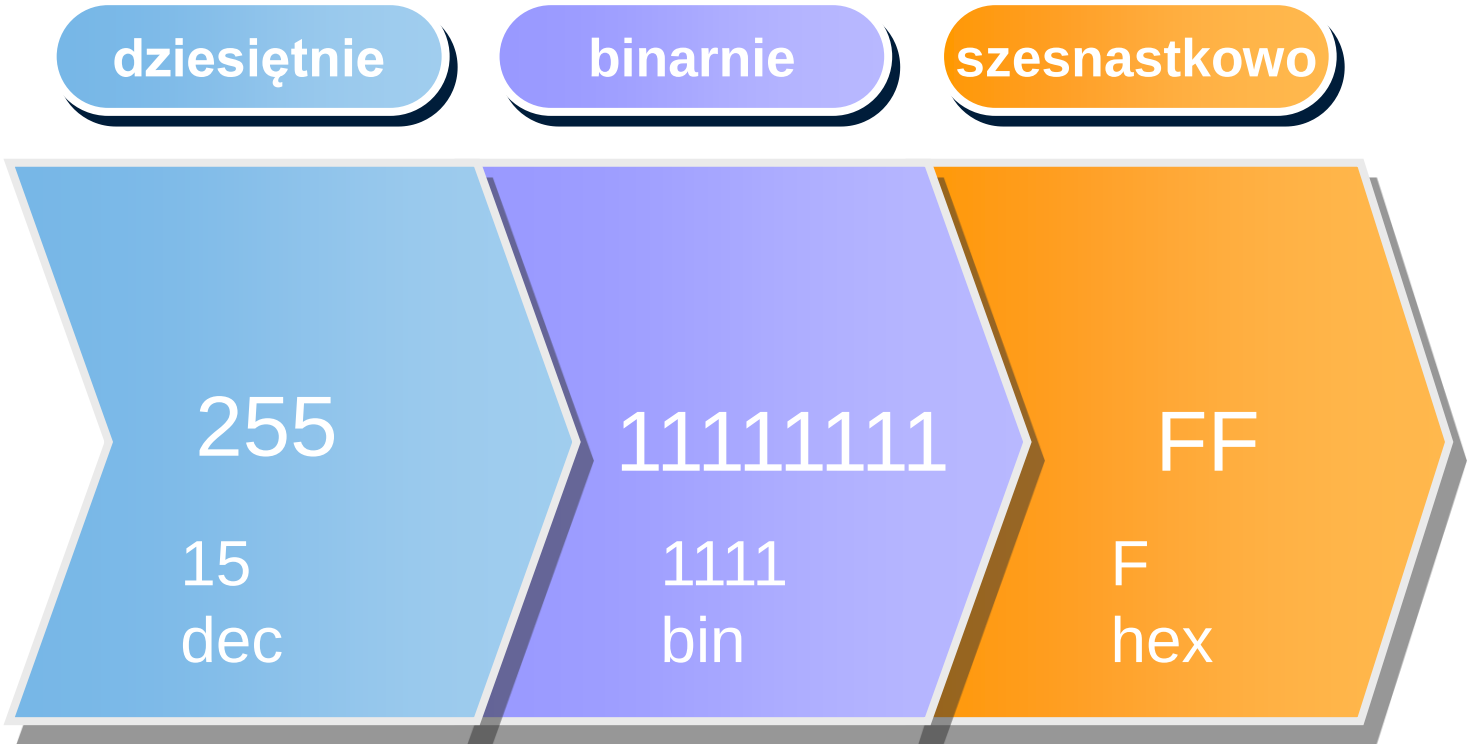
System szesnastkowy

Bajt podzielony na dwie równe części, 4-bitowe. Zatem bajt można reprezentować **dwoma znakami**, nie ośmioma.

Na każdy znak składa się 2^4 = **16** możliwości. Stąd wzięła się nazwa szesnastkowy

Powstaje problem: cyfr jest tylko 10, a trzeba mieć 16. Co zrobić? Postanowiono liczbom 10-15 przyporządkować odpowiednio znaki **A-F**.

porównanie systemów



Organizacja danych

- **Z czysto matematycznego punktu widzenia liczba może zawierać dowolną ilość cyfr. W świecie komputerów nie ma takiej swobody – komputery manipulują porcjami cyfr dwójkowych (bitów) o określonej liczności. Odpowiednie instrukcje procesora manipulują pojedynczymi**
  - bitami,
  - półbajtami (czwórkami bitów),
  - bajtami (ósemkami bitów),
  - słowami (grupami 16 bitów),
  - słowami podwójnymi (grupami 32 bitów),
  - słowami poczwórnymi (grupami 64 bitów),
  - słowami „jeszcze dłuższymi” (po 128 bitów i więcej).
- **Rozmiary te nie zostały przyjęte dowolnie.**

Bity

- **Najmniejsza jednostka informacji dla komputera binarnego.**
- **Wprawdzie może przyjmować tylko jedną z dwóch wartości, ale:**
  - może przedstawiać stan dowolnego dwustanowego obiektu:
    - jedynkę i zero,
    - prawdę i fałsz,
    - załączenie i wyłączenie,
    - płeć męską bądź żeńską,
    - rację i jej brak,
  - może reprezentować wybór pomiędzy wartościami 723 a 1245,
  - może reprezentować kolor niebieski albo czerwony
  - wartości przyjmowane przez jeden bit mogą odnosić się do dwóch niezależnych obiektów:
    - bit może reprezentować kolor niebieski albo liczbę 3256.

Półbajty

- Ang: *nibble* to zbiór czterech bitów.
- Półbajty są stosowane ze względu na dwa elementy:

liczby BCD
binary coded numbers wykorzystywane są do reprezentowania liczb dziesiętnych w postaci dwójkowej;
do reprezentacji dowolnej cyfry dziesiętnej potrzeba bowiem czterech bitów

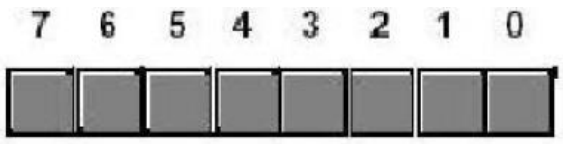
liczby szesnastkowe
można je reprezentować czterema bitami. Liczba kombinacji czterech bitów pozwala na pełne odwzorowanie liczb szesnastkowych

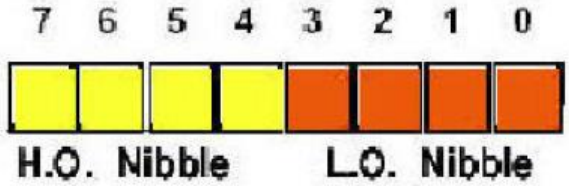
Półbajty

- W ramach jednego półbajta można zakodować (reprezentować) 16 różnych wartości:
  - 0000
  - 0001
  - 0010
  - 0011
  - 0100
  - 0101
  - 0110
  - 0111
  - 1000
  - 1001
  - 1010
  - 1011
  - 1100
  - 1101
  - 1110
  - 1111

Bajty

- Bez wątpienia najważniejsza struktura danych procesorów z rodziny 80x86.
- Składają się z ośmiu bitów
- Są najmniejszą adresowalną jednostką danych mikroprocesora 80x86.
- Zarówno pamięć operacyjna, jak i adresy wejścia-wyjścia są adresowane bajtowo.
- W języku asemblera procesora 80x86 są najmniejszą jednostką, do jakiej można się bezpośrednio odwołać.





Bajty jako znaki

- Podstawowym zastosowaniem bajta jest wykorzystanie go do przechowywania kodów znaków.
- Znaki wprowadzane z klawiatury, wyświetlane na ekranie monitora i drukowane przez drukarkę zapisywane są w komputerze jako wartości liczbowe.
- Wartości te zostały ustalone zgodnie ze standardem ASCII.

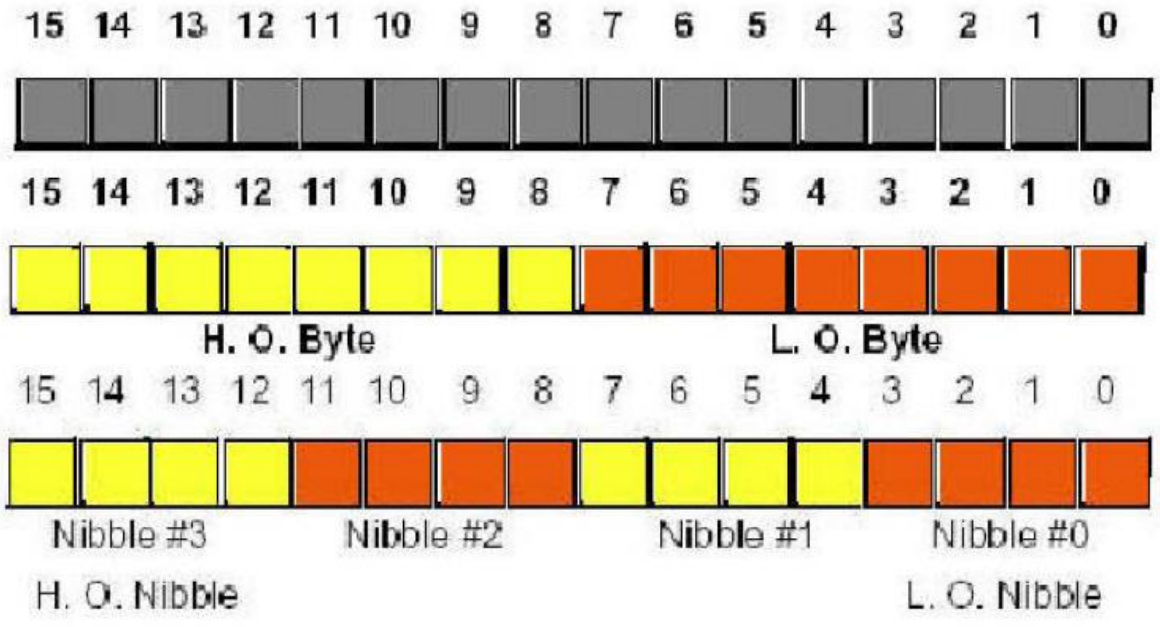
Bajty w asemblerze

```
static
    byteVar:    byte:
```

- Typ `byte` jest typem częściowo niezdefiniowanym.
- Jedyną informacją o rodzaju wartości przechowywanych w zmiennych typu `byte` jest informacja o ich bitowym rozmiarze.
- W zmiennych tych można przechowywać:
  - niewielkie liczby całkowite ze znakiem,
  - niewielkie liczby całkowite wyłącznie dodatnie,
  - znaki, itd.

Słowa

- jest zestawem 16 bitów,
- bity są numerowane od 0 do 15,
- bit zerowy jest bitem najmniej znaczącym,
- bit piętnasty jest bitem najbardziej znaczącym,



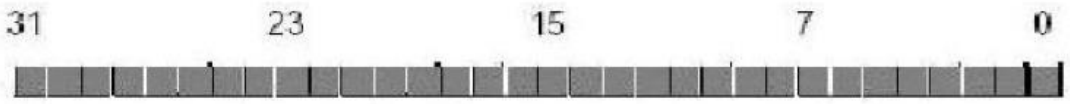
Słowa w asemblerze

```
static
    w:      word:
```

- Typ `word` reprezentuje jeden z 65 536 możliwych znaków.
- Tak duży zbiór znaków pozwala na zakodowanie nie tylko cyfr i liter alfabetu łacińskiego, ale również znaków charakterystycznych dla alfabetów narodowych, w tym alfabetów języków azjatyckich.

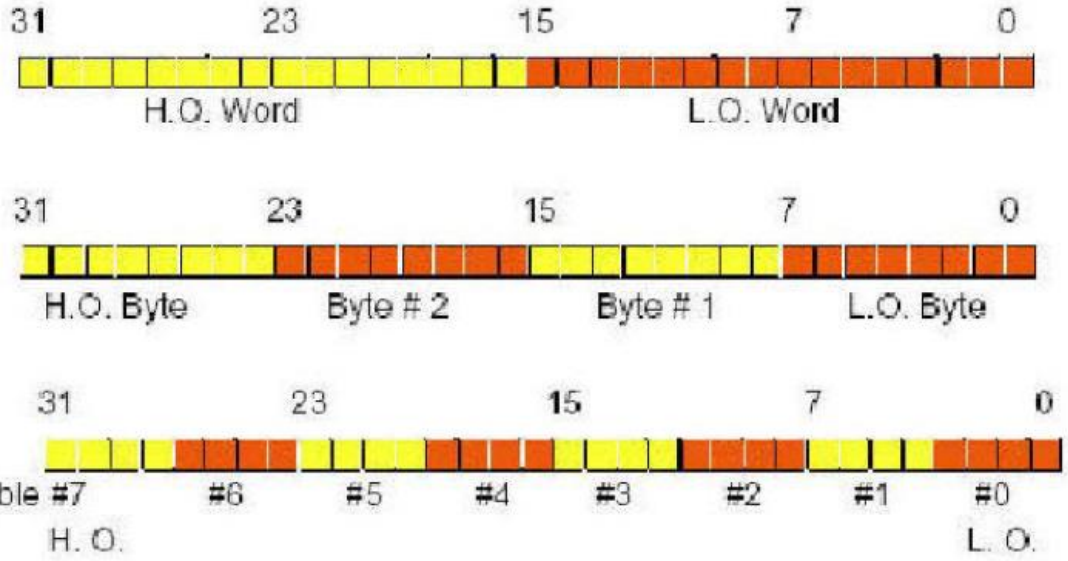
Podwójne słowa

- para słów,
- 32 bity,
- zazwyczaj wykorzystywane do reprezentowania zbioru liczb całkowitych ze zbioru od -2 147 483 648 do 2 147 483 647 (w przypadku liczb bez znaku zakres ten rozciąga się od 0 do 4 294 967 295),
- zazwyczaj mieszczą też 32-bitowe wartości zmiennoprzecinkowe



Podwójne słowa

```
static
    d:    dword:
```



Słowa poczwórne i długie

- Najpowszechniej wykorzystywane są bajty, słowa i podwójne słowa,
- Obiektami 64-bitowymi (słowo poczwórne) potrafi manipulować jednostka MMX.
- Do zmiennych zmiennoprzecinkowych czasami wykorzystuje się słowa długie (128-bitowe obiekty). Są one podstawą rozkazów SSE implementowanych w najnowszych procesorach z rodziny 80x86.

```
static
    q:    qword:
    l:    lword:
```

- rejestry są zazwyczaj 32-bitowe więc nie można tymi obiektami manipulować bezpośrednio za pomocą instrukcji mov, add, sub.

Operacje logiczne na bitach

- W algebrze Boole'a (podobnie jak w zwykłej algebrze) istnieją operacje elementarne które można wykonywać nad wartościami logicznymi. Są to:
  - negacja (zaprzeczenie logiczne - NOT)
  - alternatywa (suma logiczna - OR)
  - koniunkcja (iloczyn logiczny - AND)

- Bity są idealne do reprezentowania wartości logicznych. Wartość logiczną fałszu będziemy reprezentować stanem bitu 0, a wartość prawdy stanem 1.

Negacja - zaprzeczenie logiczne

Jest to operacja jednoargumentowa, tzn. rezultat zależy tylko od jednego argumentu. Wynikiem jest wartość logiczna przeciwna do tej, którą ma argument negacji.

Negacja	
a	NOT a
0	1
1	0

Jeśli operację negacji wykonamy nad słowem binarnym, to w wyniku otrzymamy słowo, w którym wszystkie bity mają wartości przeciwne do słowa wyjściowego.

NOT 11101111001000110010010010010101001001 11

00010000110111001101101101101010110110 00

Alternatywa - suma logiczna

Jeśli porównamy tabelkę alternatywy z tabliczką dodawania w systemie binarnym, zauważymy duże podobieństwo. Różnica jest w ostatnim wierszu - dla obu argumentów równych 1 alternatywa daje 1, a suma binarna 10. Oczywiście jest to spowodowane faktem, iż operatory logiczne dają w wyniku zawsze wartości logiczne, czyli 0 lub 1.
Mnemotechnicznie zapamiętajmy, iż alternatywa daje tylko wtedy 0, gdy wszystkie argumenty mają wartość 0 i 1 w przypadku przeciwnym.
Jeśli wykonamy operację alternatywy nad dwoma słowami binarnymi, to w wyniku otrzymamy słowo binarne, którego bity są wynikami operacji alternatywy nad odpowiadającymi im bitami argumentów.

Alternatywa		
a	b	a OR b
0	0	0
0	1	1
1	0	1
1	1	1

```
                    11000101001011
OR                  10101011011000
                    ---------------
                    11101111011011
```

Koniunkcja - iloczyn logiczny

Koniunkcja przyjmuje wartość 1, gdy wszystkie jej argumenty mają wartość 1 i 0 w przeciwnym wypadku.
Jeśli wykonamy operację koniunkcji nad dwoma słowami binarnymi, to w wyniku otrzymamy słowo binarne, którego bity są wynikami operacji koniunkcji nad odpowiadającymi im bitami argumentów.

Koniunkcja		
a	b	a AND b
0	0	0
0	1	0
1	0	0
1	1	1

```
      110001010010 11
AND   101010110110 00
      ---------------
      100000010010 00
```

Różnica symetryczna - suma modulo dwa

Operacja różnicy symetrycznej nie jest operacją elementarną, jednakże często przydaje się przy wielu okazjach. Jest to operacja dwuargumentowa

Wynikiem różnicy symetrycznej jest 1 tylko wtedy, gdy dokładnie jeden z argumentów ma wartość 1 i 0 w przeciwnym przypadku.
Jeśli wykonamy operację różnicy symetrycznej nad dwoma słowami binarnymi, to w wyniku otrzymamy słowo binarne, którego bity są wynikami tej operacji nad odpowiadającymi im bitami argumentów

Różnica symetryczna		
a	b	a XOR b
0	0	0
0	1	1
1	0	1
1	1	0

```
    110001010011
XOR 101010110000
    011011100011
```

Liczby ze znakiem i bez znaku

- Aby skutecznie reprezentować liczby ujemne w systemie dwójkowym, należy w jakiś sposób ograniczyć zakres dopuszczalnych wartości.
- Trzeba przyjąć skończoną liczbę bitów.
- Ze względu na architekturę sprzętową procesorów należy ograniczyć tę liczbę do 8, 16, 32, 64 bądź 128 bitów.

- Nie można faworyzować liczb dodatnich lub ujemnych.
- Trzeba pamiętać o wartości zero.

- Np. w ramach 256 wartości, 128 będzie ujemnych (-128 do -1) i 128 nieujemnych (od 0 do 127).
- Dla szesnastu bitów analogicznie: od -32 768 do 32 767

Reprezentacja liczb ujemnych

- W procesorach z rodziny 80x86 przyjęto notację z uzupełnieniem do dwóch.
- Najstarszy bit określa znak
- 0 – liczba jest dodatnia
- 1 – liczba jest ujemna
  - jeśli bit najstarszy ma wartość zero, to liczba traktowana jest jako dodatnia i interpretuje się ją zgodnie z regułą systemu dwójkowego.
    W przypadku kiedy najstarszy bit ma wartość jeden, liczba jest traktowana jako ujemna a jej zapis odpowiada notacji U2.

przykłady liczb 16-bitowych

100H	dodatnia
8000	ujemna
FFF	dodatnia
FFFF	ujemna

U2

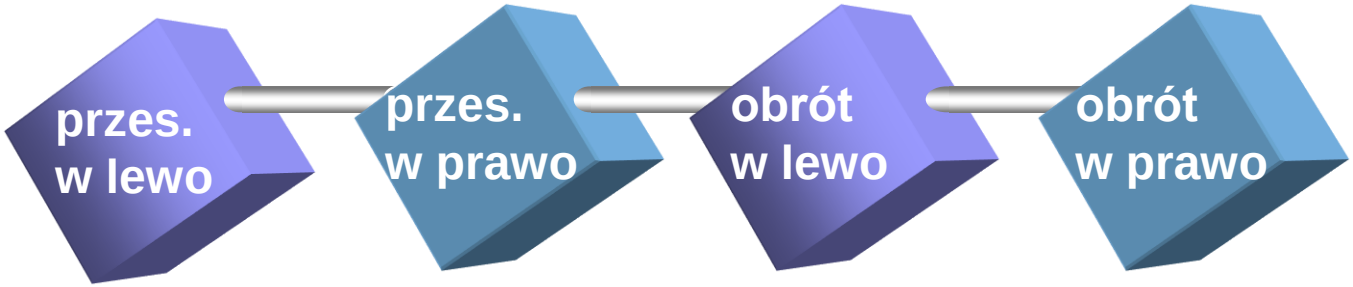
- Aby dokonać konwersji liczby dwójkowej do zapisu z uzupełnieniem do dwóch, należy skorzystać z następującego algorytmu:

1. Odwrócić wszystkie bity liczby – wykonać operację NOT.
2. Dodać do wyniku wartość jeden.

przykład:

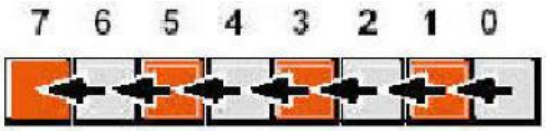
zapis liczby - 5:	%1111_1011
po zanegowaniu bitów:	%0000_0100
po dodaniu jedności:	%0000_0101

Przesunięcia i obroty



przes. w lewo	przes. w prawo	obrót w lewo	obrót w prawo
Każdy bit jest przesuwany na pozycję swojego lewego sąsiada	Każdy bit jest przesuwany na pozycję swojego prawego sąsiada	Bity wypychane z operandu z jednej strony są do niego wprowadzane z drugiej	Bity wypychane z operandu z jednej strony są do niego wprowadzane z drugiej

Przesunięcie bitowe w lewo



- Bit zerowy jest w tej operacji zerowany a wartość bitu najstarszego staje się przeniesieniem powstałym w wyniku wykonania operacji przesunięcia.
- Procesory 80x86 udostępniają instrukcję maszynową przesunięcia bitowego w lewo – shl
  - shl (*rozmiar-przesunięcia, operand docelowy*);
  - shl (*1, dest*)

- Warto zauważyć, że przesunięcie bitowe w lewo odpowiada mnożeniu wartości przez wielokrotność liczby 2. Czyli, przesunięcie w lewo o jedno miejsce to pomnożenie przez 2, o dwa miejsca — pomnożenie przez 4, o trzy miejsca — pomnożenie przez 8 itd

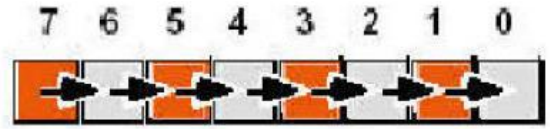
Przesunięcie bitowe w lewo

- Jeśli najstarszy bit zostanie przesunięty poza operand, jego wartość ląduje w znaczniku przeniesienia.
- Po wykonaniu przesunięcia można wykrywać utratę informacji, sprawdzając bezpośrednio po wykonaniu instrukcji shl (1, dest) wartość znacznika przeniesienia (stosując na przykład konstrukcję if (@c) then....

  - Dokumentacja procesorów firmy Intel sugeruje, że w przypadku rozmiaru przesunięcia większego niż jeden, wartość znacznika przeniesienia jest nieokreślona. Zwykle co prawda znacznik ten zawiera ostatni bit wypchnięty poza operand docelowy, ale nie jest to gwarantowane !

Przesunięcie bitowe w prawo

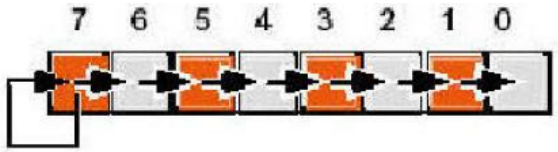
- Realizowane podobnie, tyle że bity przemieszczają się w przeciwnym kierunku:



- Operacja jest w procesorach 80x86 realizowana instrukcją shr. Składnia identyczna jak shl:
  - shr (*rozmiar-przesunięcia, operand-docelowy*);
- Bit zerowy jest wypychany poza operand i ląduje w znaczniku przeniesienia.
- Jeśli przesuwanie w lewo odpowiada operacji mnożenia przez dwa, to przesuwanie w prawo powinno być tożsame z operacją dzielenia przez dwa (a ogólnie – dzielenie przez podstawę systemu liczbowego).
- Przesunięcie w prawo realizuje operację dzielenia **wyłącznie dla liczb bez znaku**.

Arytmetyczne przesunięcie w prawo

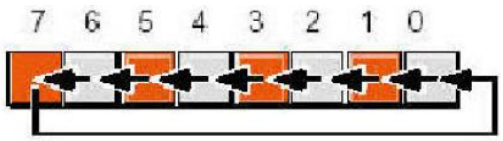
- Działa jak zwykłe przesunięcie w prawo (zwane dla odróżnienia bitowym lub logicznym), z jednym wyjątkiem: zamiast zerować najstarszy bit, kopiuje jego dotychczasową wartość, co zapobiega modyfikacji znaku liczby.



- Procesor 80x86 implementuje operację przesunięcia arytmetycznego w prawo instrukcją sar (ang.: *shift arithmetic right*).
  - sar (rozmiar-przesunięcia, operand-docelowy);

Obrót w lewo

- Bity wypychane z operandu z jednej strony są do niego wprowadzane z drugiej strony.



- Procesory 80x86 implementują te operacje za pośrednictwem instrukcji rol (obrót w lewo) oraz ror (obrót w prawo). Składnia:
  - rol (rozmiar-obrotu, operand-docelowy);
  - ror (rozmiar-obrotu, operand-docelowy);

- Kiedy rozmiar obrotu określany jest jako 1, bit który jest wypychany z operandu, jest – poza skopiowaniem go na pozycję z drugiej strony operandu – kopiowany do znacznika przeniesienia.

Arytmetyka zmiennoprzecinkowa

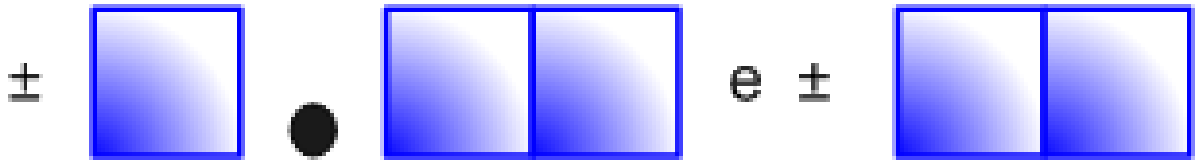
1. Arytmetyka liczb rzeczywistych nie pozwala na reprezentowanie wartości ułamkowych. Arytmetyka zmiennoprzecinkowa daje wyniki bliższe rzeczywistym.

2. Arytmetyka zmiennoprzecinkowa nie odpowiada jednak regułom standardowej algebry.

3. Liczby całkowite w komputerze nie są pełnymi odpowiednikami liczb całkowitych znanych z teorii, ponieważ komputer reprezentuje je za pośrednictwem skończonej liczby bitów.

Liczby zmiennoprzecinkowe

- **Większość formatów zmiennoprzecinkowych przyjmuje reprezentowanie liczby rzeczywistej za pomocą zapisu wykładniczego przy zarezerwowaniu pewnej liczby bitów na mantysę liczby i pewnej – zwykle mniejszej - liczby bitów na wykładnik.**
- **W efekcie liczby zmiennoprzecinkowe reprezentują jedynie podzbiór liczb rzeczywistych o określonej liczbie cyfr znaczących.**
- **Przykład:**
  - nasz format liczby zmiennoprzecinkowej będzie dopuszczał mantysę o trzech cyfrach znaczących oraz dziesiętny dwucyfrowy wykładnik. Zarówno mantysa, jak i wykładnik, będą liczbami ze znakiem.



Zmiennoprzecinkowe - przykład

- Przy dodawaniu i odejmowaniu dwóch liczb w zapisie wykładniczym należy doprowadzić te liczby do postaci, w której ich wykładniki są takie same. Na przykład przy sumowaniu liczb **1,23e1** i **4,56e0** należy sprowadzić obie do identycznego wykładnika.
- Można to zrobić, zapisując 4,56e0 jako 0,45e1 i wtedy wykonać dodawanie.
- Da to wynik **1,686e1**, ale:
  - Otrzymany wynik nie mieści się na trzech znaczących cyfrach, konieczne jest więc albo zaokrąglenie, albo obcięcie wyniku do trzech cyfr znaczących.
- Dokładniejszy, choć i tak już niedokładny rezultat powstanie w wyniku zaokrąglenia.
- Zaokrąglimy więc otrzymaną wartość do 1,69e1.
- Jak widać niedostatki precyzji (liczby cyfr czy bitów wykorzystanych w obliczeniach) wpływają natychmiast na dokładność obliczeń.

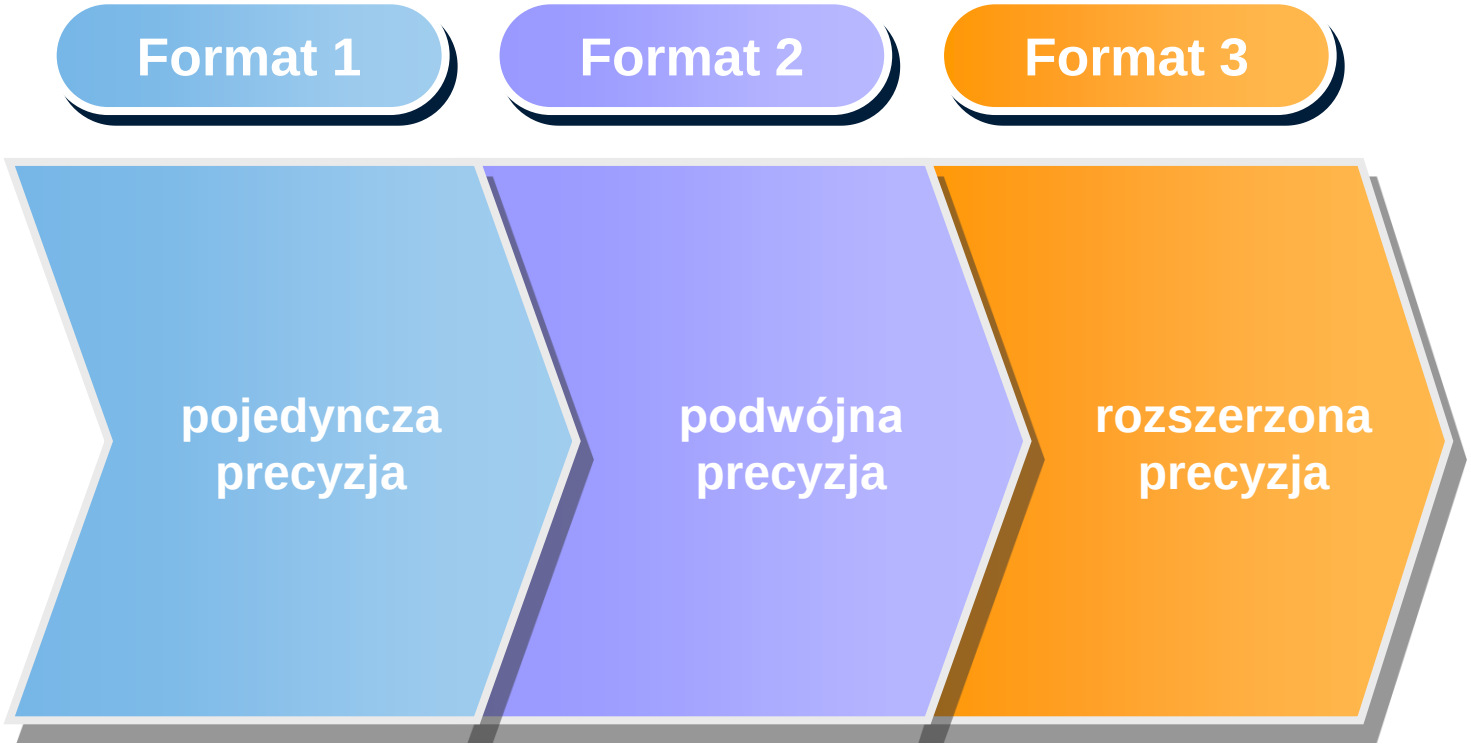
Utrata dokładności - problem

- **Utrata dokładności w wyniku wykonania pojedynczego obliczenia nie jest zwykle na tyle duża, aby się nią przejmować w zastosowaniach innych niż czysto matematyczne. Jednak jeżeli obliczenia wykonywane są kaskadowo, to znaczy wynik jednej zmiennoprzecinkowej operacji staje się operandem następnej, to błąd ulega kumulacji, co przy odpowiednio długiej sekwencji obliczeń może stanowić poważny problem.**

- **Przykład:**
  - dodajemy liczby **1,23e3** i **1,00e0**.
  - po wyrównaniu wykładników otrzymamy 1,23e3+0,001e3. Sumą obu wartości jest, po zaokrągleniu **1,23e3**.
  - Wydaje się, że obliczenie przebiega jak najbardziej poprawnie – przy trzech cyfrach znaczących dodanie do większej liczby liczby znacznie mniejszej nie wpływa mocno na wartość pierwszej.
  - **A co by było gdyby dodawanie to wykonać dziesięć razy ???**

Formaty przyjęte przez IEEE

- Firma Intel wprowadziła trzy formaty zmiennoprzecinkowe:



IEEE 754

- **Liczby zmiennoprzecinkowe są komputerową reprezentacją liczb rzeczywistych zapisanych w formie wykładniczej (naukowej). Aby uprościć arytmetykę na nich, przyjęto ograniczenia zakresu mantysy i eksponentu oraz wprowadzono inne założenia, które reguluje norma IEEE 754 (dla liczb zapisanych w kodzie dwójkowym).**
- **Liczbę zapisuje się jako ciąg zer i jedynek przyjmując umowny podział na "pola":**

			w jednostce zmiennoprzecinkowej		
S	E	M	G	R	X/S
znak	wykładnik w kodzie spolaryzowanym	ułamek	bity specjalne		

S – znak, jest zawsze jednobitowy i ma wartość 0 jeśli liczba jest dodatnia lub 1 jeśli jest ujemna

E – wykładnik (inaczej: eksponent, cecha), ma długość zależną od długości całej liczby i kodowane jest w kodzie $2^{k-1}-1$

M – moduł (inaczej mantysa) ułamka. Ma wartość z przedziału [1,2). Zapisuje się go bez poprzedzającej go jedynki z kropką (tzw. bit ukryty). Innymi słowy, zapisujemy jedynie część ułamkową modułu przyjmując, że zawsze (pomijając wyjątki, o których później) część całkowita wynosi 1.

IEEE 754 – c.d.

- Bity specjalne występują jedynie w wewnętrznej reprezentacji liczy w jednostce zmiennoprzecinkowej i wykorzystywane są do zniwelowania efektu utraty dokładności przy wykonywaniu działań, przez zaokrąglenie.
- Zarówno liczba wejściowa jak i wyjściowa takiego układu nie posiada tych bitów. Oznacza to, że liczby 32/64/128 bitowe, w jednostce zmiennoprzecinkowej, po uzupełnieniu bitami specjalnymi, maja długość odpowiednio 35/67/131 bitów.

rozmiar liczby	wykładnik E	moduł ułamka M	nazwa
32	8	23	single
64	11	52	double
128	16	111	extended

Zmiennoprzecinkowe nieskończoności

- Za pomocą liczb zmiennoprzecinkowych można również zakodować nieskończoności. Kodem zarezerwowanym dla tych wartości jest wykładnik składający się z samych jedynek i moduł z samych zer.

Wyróżnia się **minus nieskończoność** (przykład dla kodu 32-bitowego):
$-\infty$ = 1 1111 1111 000 0000 0000 0000 0000 0000

Plus nieskończoność (przykład dla kodu 32-bitowego):
∞ = 0 1111 1111 000 0000 0000 0000 0000 0000

Liczby zmiennoprzecinkowe w asemblerze

reprezentowane są za pomocą literałów zmiennoprzecinkowych:
liczbę może poprzedzać nieobowiązkowy znak (plus lub minus), w przypadku braku znaku zakłada się, że liczba jest dodatnią,

za ewentualnym znakiem występuje jedna bądź więcej cyfr dziesiętnych,

cyfry te uzupełnione są przecinkiem dziesiętnym (kropka) oraz jedną większą liczbą cyfr dziesiętnych.

całość może być uzupełniona literą „e” lub „E”, nieobowiązkowym znakiem oraz kolejnymi cyframi dziesiętnymi.

1.234 3.75e2 -1.0 1.1e-1 1e+4 0.1 -123.456e789 25e0

deklaracje liczb zmiennoprzecinkowych

Do wykorzystania są następujące typy: real32, real64, real80. Podobnie jak w ich odpowiednikach całkowitych liczba kończąca nazwę typu określa rozmiar (w bitach) zmiennych tego typu.

real32 – odpowiada liczbom o pojedynczej precyzji

real64 – odpowiada liczbom o podwójnej precyzji

real80 – odpowiada liczbom o rozszerzonej precyzji

Wyprowadzanie liczb zmiennoprzecinkowych

służy do tego jedna z procedur:

stdout.putr32, stdout.putr64, stdout.putr80

składnia wywołania jest identyczna dla wszystkich procedur:

stdout.putr32(lzmiennoprz, szerokość, liczba-cyfr-po-przecinku)

pierwszym argumentem powinna być wartość zmiennoprzecinkowa, która ma zostać wyprowadzona na standardowe wyjście programu

argument musi mieć odpowiedni rozmiar – w wywołaniu stdout.putr80 należy przekazywać wartości zmiennoprzecinkowe rozszerzonej precyzji

Wyprowadzanie liczb zmiennoprzecinkowych

Wywołanie:

stdout.putr32(pi,10,4);

powoduje wyprowadzenie napisu:

_ _ _ _ 3.1416

Wyprowadzanie liczb zmiennoprzecinkowych w postaci wykładniczej

służą do tego procedury:

stdout.pute32, stdout.pute64, stdout.pute80

wywoływane następująco:

stdout.pute32(liczba-zmiennoprz , szerokość);

szerokość określa wszystkie pozycje po przecinku (razem z pozycjami wykładnika)

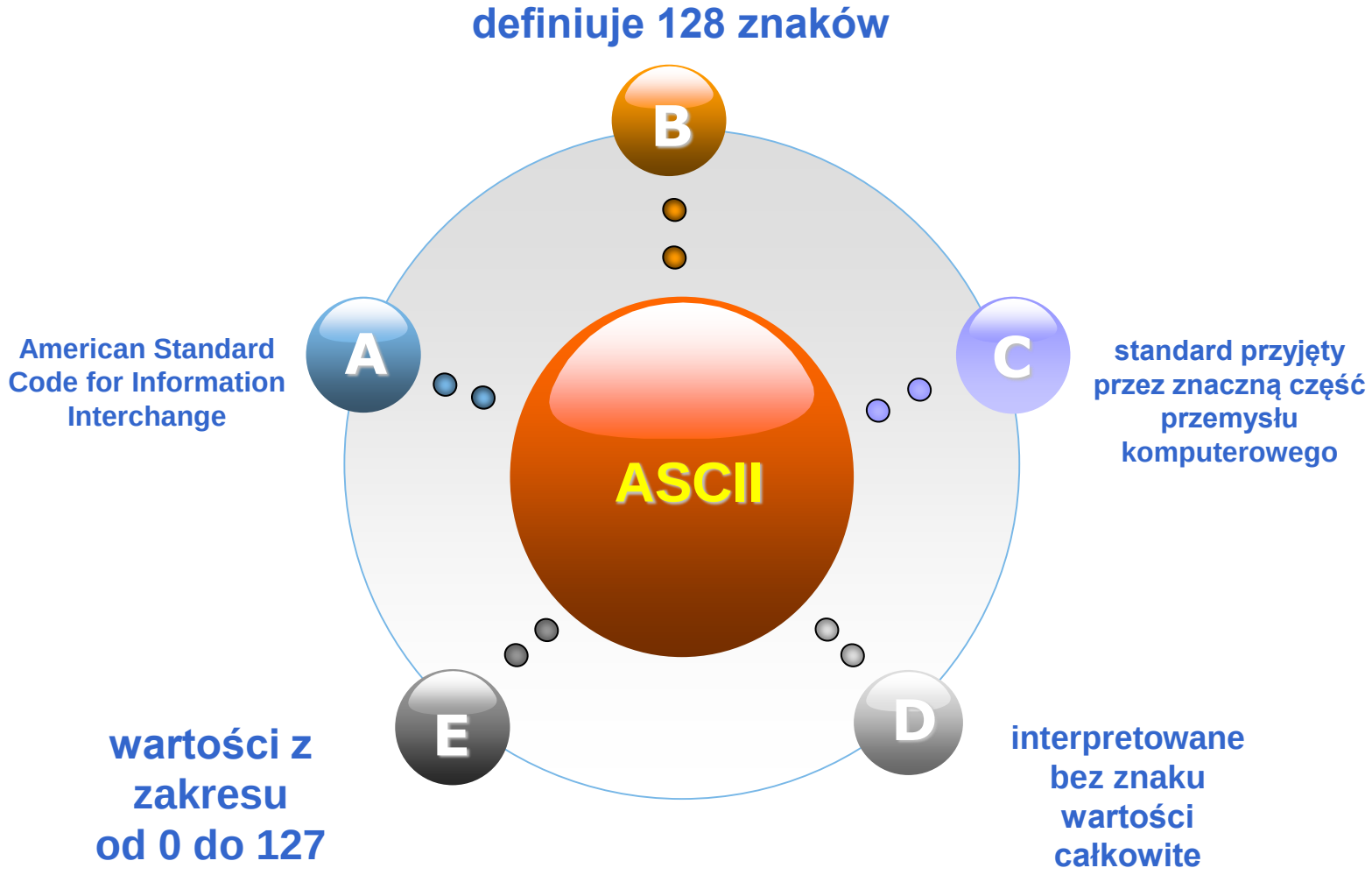
Wyprowadzanie liczb zmiennoprzecinkowych w postaci wykładniczej

Można również wykorzystać procedurę stdout.put, jeśli w wywołaniu tej procedury znajdzie się nazwa obiektu zmiennoprzecinkowego jego wartość zostanie skonwertowana do postaci napisu zawierającego notację wykładniczą liczby.

Szerokość ustalana jest automatycznie.
Jeśli procedura stdout.put ma konwertować liczby zmiennoprz. do zapisu dziesiętnego należy argument zmiennoprz. określać następująco:
nazwa:szerokość:liczba-cyfr-po-przecinku

np.
stdout.put(„Pi = ”, pi:5:3);
spowoduje wyprowadzenie na wyjście napisu:
Pi = 3.141

Znaki – zestaw znaków ASCII



Zestaw znaków ASCII

Zestaw kodów ASCII (American Standard Code for Information Interchange) definiuje odwzorowanie 128 znaków do interpretowanych bez znaku wartości całkowitych z zakresu od 0 do 127.

Standard przyjęty przez znaczną część przemysłu. Jeżeli więc do reprezentowania znaku „A” wykorzystywana jest liczba 65, zgodnie z zestawem znaków ASCII, wtedy można mieć pewność że liczba ta zostanie zinterpretowana jako znak „A” również przez drukarkę czy terminal.

Grupy znaków ASCII

Zestaw znakó ASCII podzielony jest na 4 grupy po 32 znaki.

- Pierwsze 32 są to znaki niedrukowane (sterujące), np. powrót karetki, znak wysuwu wiersza, znak cofania.
- Kolejne 32 to znaki specjalne, przystankowe i cyfry.
- Trzecia grupa to znaki wielkich liter alfabetu. Jako, że znaków w alfabecie łacińskim jest 26, sześć kodów przypisanych zostało do różnych znaków specjalnych.
- Czwartą grupę stanowią kody 26 małych liter alfabetu, pięć znaków specjalnych i znak sterujący DELETE.

ASCII w asemblerze

Literały znakowe w języku HLA mogą przyjmować jedną z dwóch postaci: pojedynczego znaku otoczonego znakami pojedynczego cudzysłowu albo wartości (z zakresu 0 do 127) określającej kod ASCII poprzedzonej znakiem kratki (#):

‘A’ #65 #41 #0100_0001

wszystkie powyższe literały reprezentują ten sam znak „A”

ASCII w HLA

Warto przyjąć zasadę, że znaki drukowalne określane są w kodzie programu literałami w pierwszej z prezentowanych form, czyli jako znaki ujęte w znaki pojedynczego cudzysłowu.

Znak kratki i literał liczbowy należy stosować wyłącznie do określania znaków niedrukowalnych: specjalnych i sterujących, ewentualnie do określenia znaków z rozszerzonej części zestawu ASCII, które w kodzie źródłowym mogą być wyświetlane niepoprawnie.

literał łańcuchowy ≠ literał znakowy

- Literały łańcuchowe zawierają zero bądź więcej znaków i są ograniczane znakami podwójnego cudzysłowu
- Literały znakowe zawierają zaś pojedynczy znak i są ograniczane znakami pojedynczego cudzysłowu. W szczególności należy uświadomić sobie fakt, że:

'A' ≠ "A"

ASCII w HLA

Aby zadeklarować w HLA zmienną znakową należy skorzystać z typu danych o nazwie char. Zmienne znakowe można przy deklaracji od razu inicjalizować:

```
static
        znakA:                  char:='A'
        znak_rozszerzony:       char:=#128
```

jako że zmienne są obiektami 8-bitowymi, można nimi manipulować za pośrednictwem ośmiobitowych rejestrów i odwrotnie – zawartość takiego rejestru można skopiować do zmiennej znakowej

Wprowadzanie i wyprowadzanie znaków

służą temu procedury:

stdout.putc, stdout.putSize, stdout.put, stdin.getc, stdin.get

stdout.putc(zmienna-znakowa);

- procedura wyprowadza na standardowe wyjście programu pojedynczy znak określony wartością argumentu wywołania procedury
- argument może zostać określony jako stała lub zmienna znakowa bądź rejestr 8-bitowy

stdout.putcSize

procedura pozwala na wyprowadzenie znaków z określeniem szerokości wyprowadzanego napisu i wypełnienia

stdout.putcSize(zmienna-znakowa, szerokosc, wypełnienie);

procedura wyprowadza znak określony zmienną znakową, umieszczając go w napisie o określonej szerokości. Jeżeli bezwzględna wartość argumentu szerokość jest większa niż jeden, napis zostanie uzupełniony znakami *wypełnienia*. Jeśli argument szerokość zostanie określony jako ujemy, wyprowadzany znak będzie wyrównany do lewej krawędzi napisu. Wartość dodatnia powoduje wyrównanie do prawej krawędzi napisu.

Znaki są zwykle wyrównane do lewej.

stdout.put a wartości znakowe

Wartości znakowe mogą być też wyprowadzane za pośrednictwem uniwersalnej procedury wyjścia stdout.put. Jeśli na liście wywołania tej procedury znajduje się zmienna znakowa, kod procedury automatycznie wyświetli odpowiadający jej znak, np.:

```
stdout.put( "Znak c = ",c," ' ",nl );
```

Pobieranie znaków

za pomocą procedur stdin.getc i stdin.get.

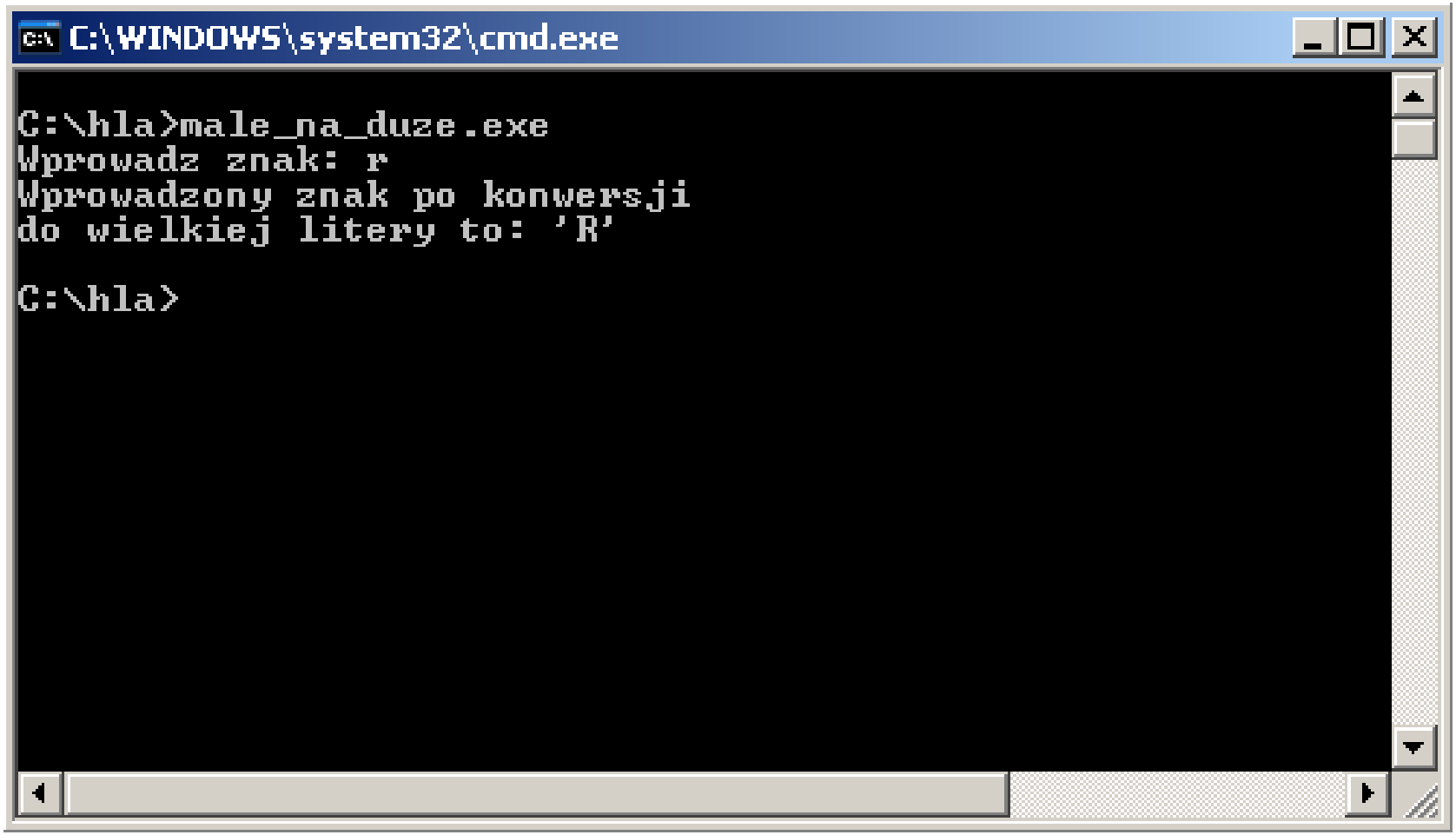
Procedura stdin.getc nie przyjmuje żadnych argumentów. Jej działanie ogranicza się do wczytania z bufora urządzenia standardowego wejścia pojedynczego znaku i umieszczenia go w AL. Po wczytaniu do rejestru można tą wartością manipulować na miejscu albo skopiować do zmiennej w pamięci.

Wczytywanie i wyprowadzanie znaków stdin.getc()

wiedząc, że małe i duże litery różnią się w kodzie ASCII pozycją 6-go bitu napisać program, który wczytuje małe litery z klawiatury, zamienia je na duże i wyświetla na ekranie.

wykorzystać procedurę stdin.getc()

działanie programu



jeden z możliwych sposobów realizacji zadania

male_na_duze.hla - Notatnik

Plik Edycja Format Widok Pomoc

```
program male_na_duze;
#include( "stdlib.hhf" );

static
        c:      char;

begin male_na_duze;

        stdout.put( "Wprowadz znak: " );
        stdin.getc();
        if( al>='a' ) then
                if( al<='z' ) then
                        and($5f,al);
                endif;
        endif;
        stdout.put( "Wprowadzony znak po konwersji",nl,"do wielkiej litery to: '" );
        stdout.putc( al );
        stdout.put( "'",nl);

end male_na_duze;
```

Wczytywanie i wyprowadzanie znaków stdin.get

Zmodyfikować program tak, aby wczytywanie znaków odbywało się za pomocą zwykłej procedury stdin.get

jeden z możliwych sposobów realizacji zadania

male_na_duze_2.hla - Notatnik

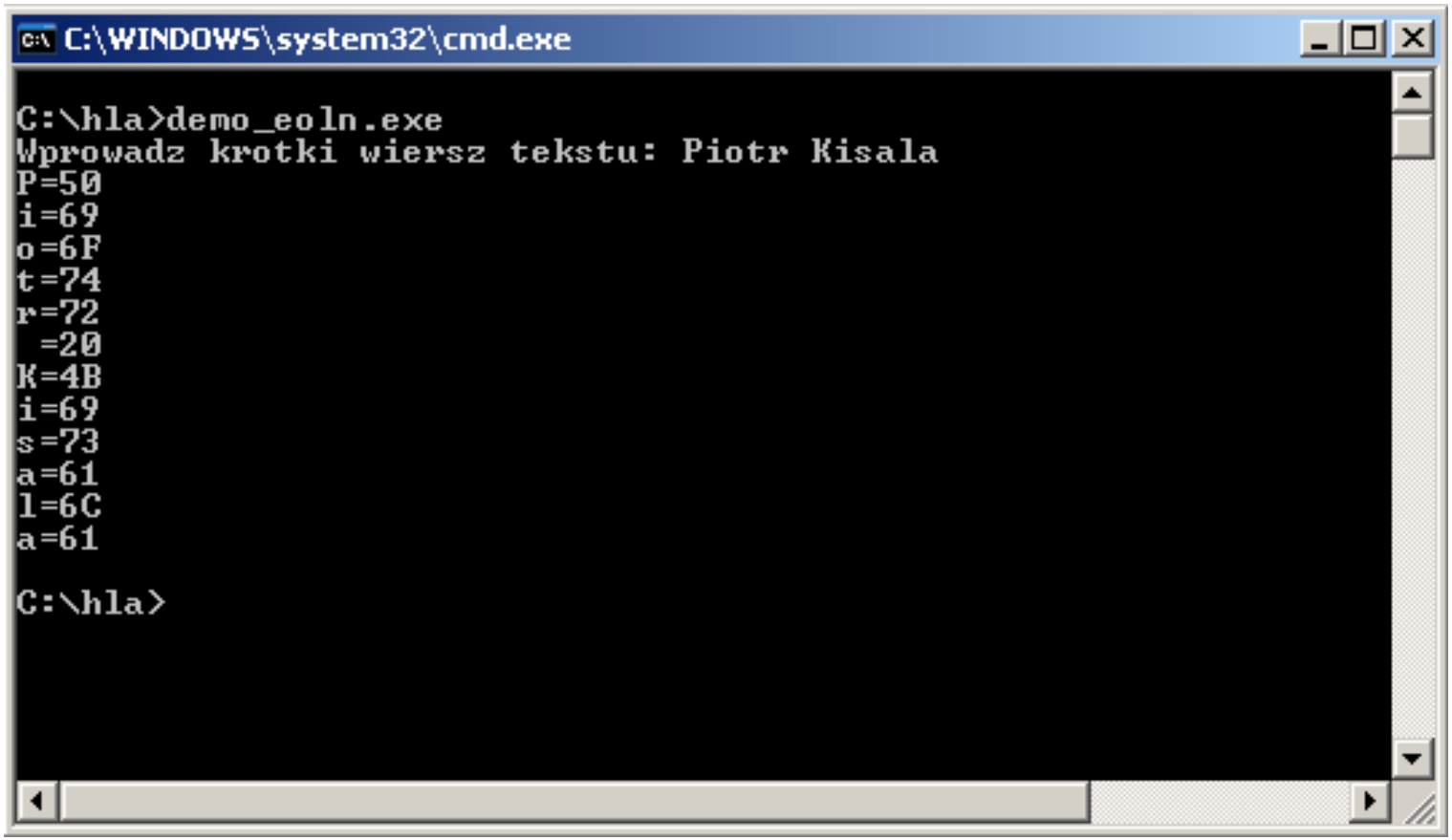
Plik Edycja Format Widok Pomoc

```
program male_na_duze;
#include( "stdlib.hhf" );

static
        c:      char;

begin male_na_duze;

        stdout.put( "wprowadz znak: " );
        stdin.get(c);
        if( c >='a' ) then
                if( c <='z' ) then
                        and($5f,c);
                endif;
        endif;
        stdout.put( "wprowadzony znak po konwersji",nl,"do wielkiej litery to: '",c,"'",nl );

end male_na_duze;
```

wczytywanie kilku znaków

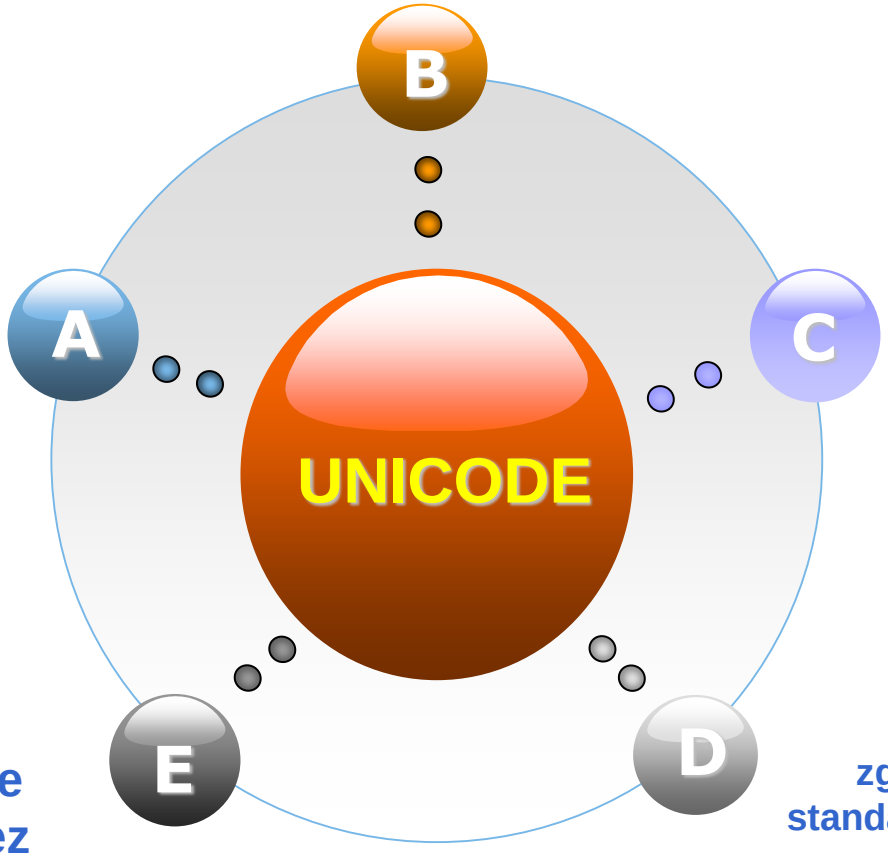
Napisać program wczytujący z klawiatury ciąg znaków (np. "Imię Nazwisko"), wyprowadzający następnie każdy ze znaków w kolumnie pionowej jednocześnie dekodując wartości poszczególnych znaków na postać liczbową szesnastkową, zgodnie z następnym slajdem.

działanie programu



Zestaw znaków UNICODE

pozwala na zdefiniowanie 65536 znaków



A — każdy znak reprezentowany jest słowem szesnastobitowym

B

C — dużo większy zakres znaków niż w przypadku standardu ASCII

D — zgodny ze standardem ASCII

E — wypieranie ASCII przez UNICODE

Podsumowanie

szybka powtórka

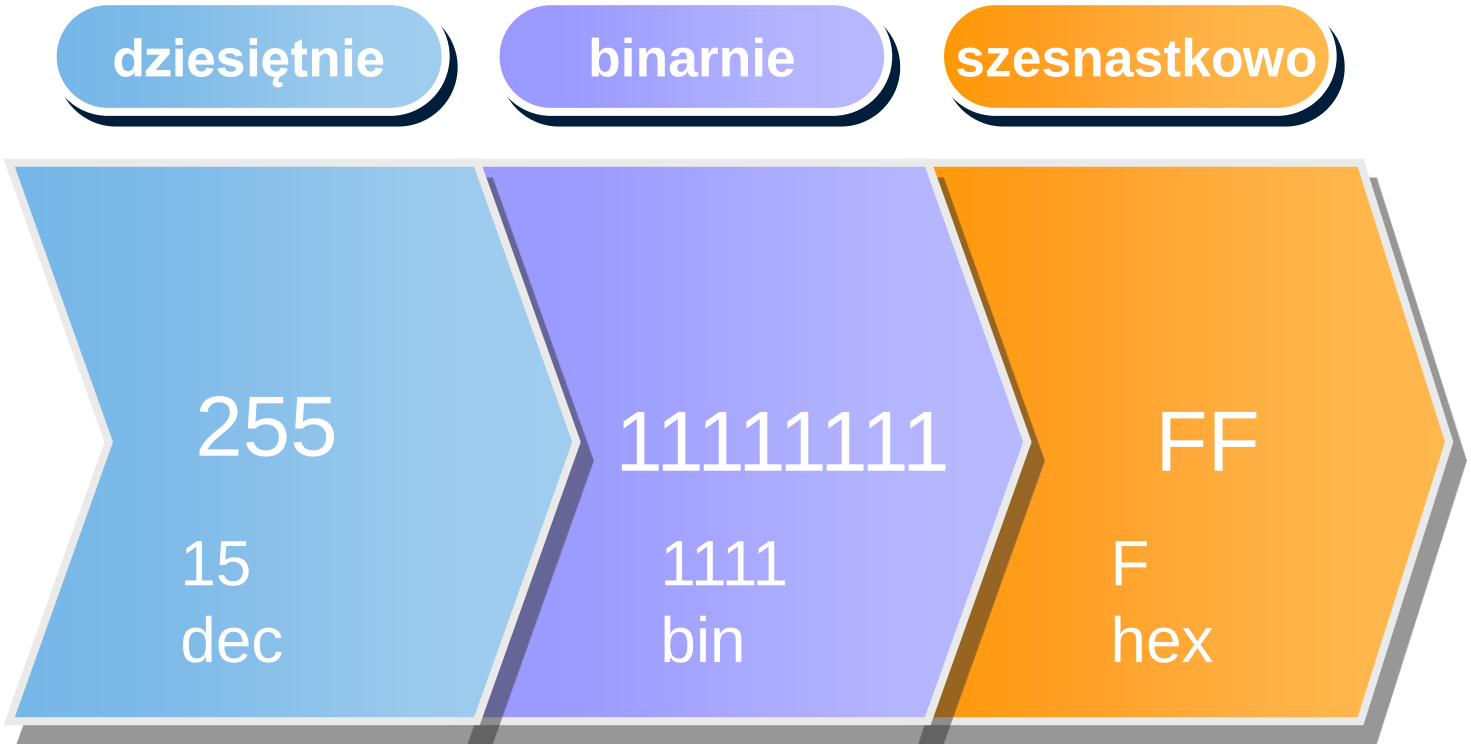
System szesnastkowy

Bajt podzielony na dwie równe części, 4-bitowe. Zatem bajt można reprezentować **dwoma znakami**, nie ośmioma.

Na każdy znak składa się 2^4 = **16** możliwości. Stąd wzięła się nazwa szesnastkowy

Powstaje problem: cyfr jest tylko 10, a trzeba mieć 16. Co zrobić? Postanowiono liczbom 10-15 przyporządkować odpowiednio znaki **A-F**.

porównanie systemów



Podsumowanie

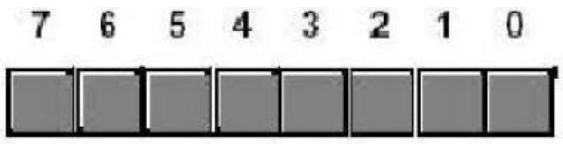
szybka powtórka

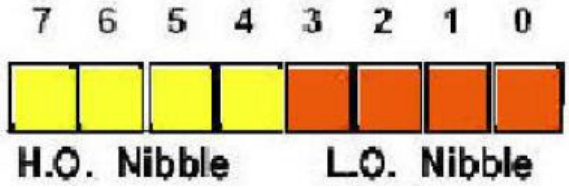
Organizacja danych

- **Z czysto matematycznego punktu widzenia liczba może zawierać dowolną ilość cyfr. W świecie komputerów nie ma takiej swobody – komputery manipulują porcjami cyfr dwójkowych (bitów) o określonej liczności. Odpowiednie instrukcje procesora manipulują pojedynczymi**
  - bitami,
  - półbajtami (czwórkami bitów),
  - bajtami (ósemkami bitów),
  - słowami (grupami 16 bitów),
  - słowami podwójnymi (grupami 32 bitów),
  - słowami poczwórnymi (grupami 64 bitów),
  - słowami „jeszcze dłuższymi” (po 128 bitów i więcej).
- **Rozmiary te nie zostały przyjęte dowolnie.**

Bajty

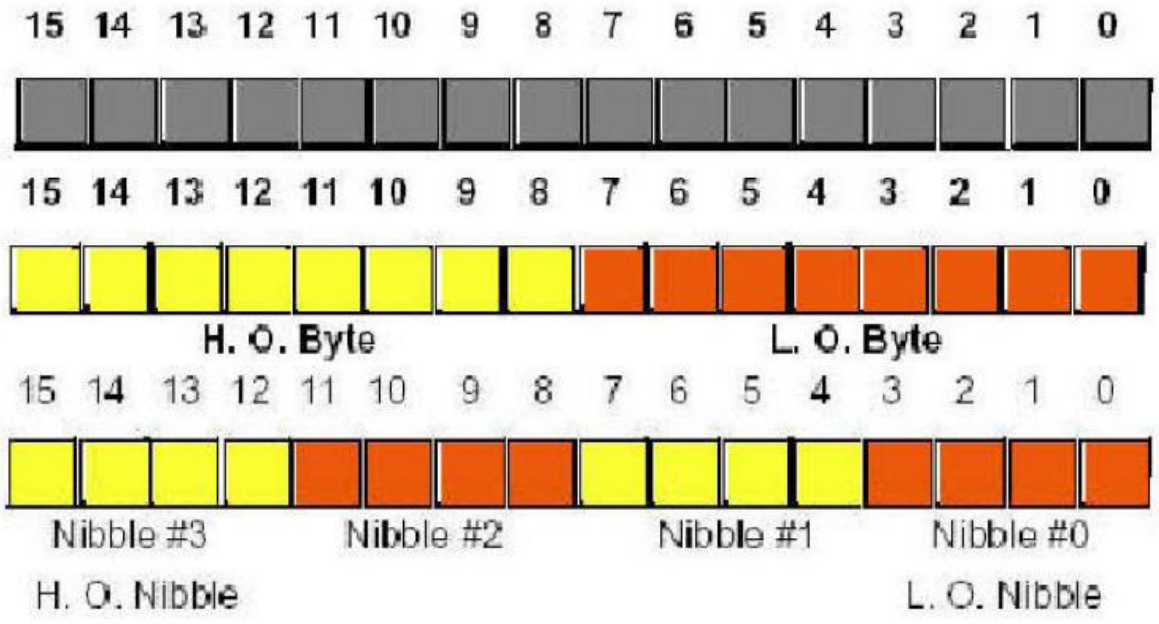
- Bez wątpienia najważniejsza struktura danych procesorów z rodziny 80x86.
- Składają się z ośmiu bitów
- Są najmniejszą adresowalną jednostką danych mikroprocesora 80x86.
- Zarówno pamięć operacyjna, jak i adresy wejścia-wyjścia są adresowane bajtowo.
- W języku asemblera procesora 80x86 są najmniejszą jednostką, do jakiej można się bezpośrednio odwołać.





Słowa

- jest zestawem 16 bitów,
- bity są numerowane od 0 do 15,
- bit zerowy jest bitem najmniej znaczącym,
- bit piętnasty jest bitem najbardziej znaczącym,



Operacje logiczne na bitach

- W algebrze Boole'a (podobnie jak w zwykłej algebrze) istnieją operacje elementarne które można wykonywać nad wartościami logicznymi. Są to:
  - negacja (zaprzeczenie logiczne - NOT)
  - alternatywa (suma logiczna - OR)
  - koniunkcja (iloczyn logiczny - AND)

- Bity są idealne do reprezentowania wartości logicznych. Wartość logiczną fałszu będziemy reprezentować stanem bitu 0, a wartość prawdy stanem 1.

Podsumowanie

szybka powtórka

Reprezentacja liczb ujemnych

- W procesorach z rodziny 80x86 przyjęto notację z uzupełnieniem do dwóch.
- Najstarszy bit jest określa znak
- 0 – liczba jest dodatnia
- 1 – liczba jest ujemna
  - jeśli bit najstarszy ma wartość zero, to liczba traktowana jest jako dodatnia i interpretuje się ją zgodnie z regułą systemu dwójkowego.
    W przypadku kiedy najstarszy bit ma wartość jeden, liczba jest traktowana jako ujemna a jej zapis odpowiada notacji U2.

U2

- Aby dokonać konwersji liczby dwójkowej do zapisu z uzupełnieniem do dwóch, należy skorzystać z następującego algorytmu:

1. Odwrócić wszystkie bity liczby – wykonać operację NOT.
2. Dodać do wyniku wartość jeden.

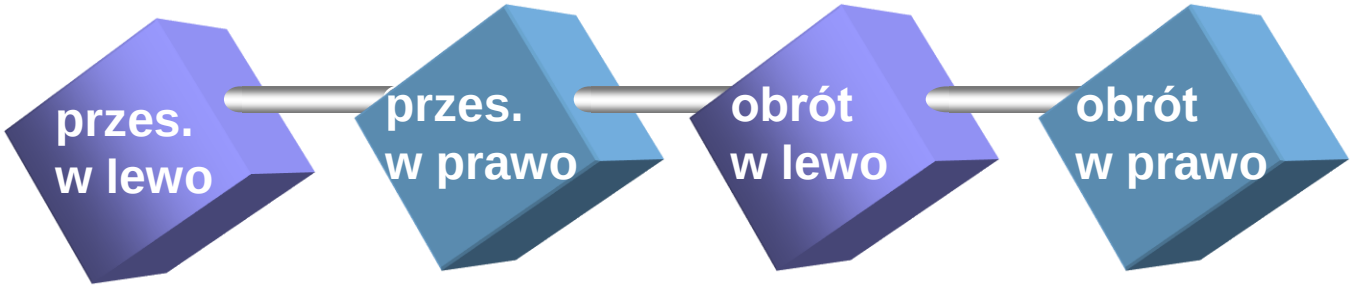
przykład:

zapis liczby - 5:	%1111_1011
po zanegowaniu bitów:	%0000_0100
po dodaniu jedności:	%0000_0101

Podsumowanie

szybka powtórka

Przesunięcia i obroty



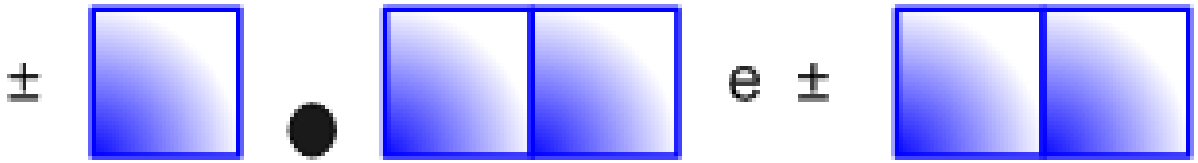
przes. w lewo	przes. w prawo	obrót w lewo	obrót w prawo
Każdy bit jest przesuwany na pozycję swojego lewego sąsiada	Każdy bit jest przesuwany na pozycję swojego prawego sąsiada	Bity wypychane z operandu z jednej strony są do niego wprowadzane z drugiej	Bity wypychane z operandu z jednej strony są do niego wprowadzane z drugiej

Podsumowanie

szybka powtórka

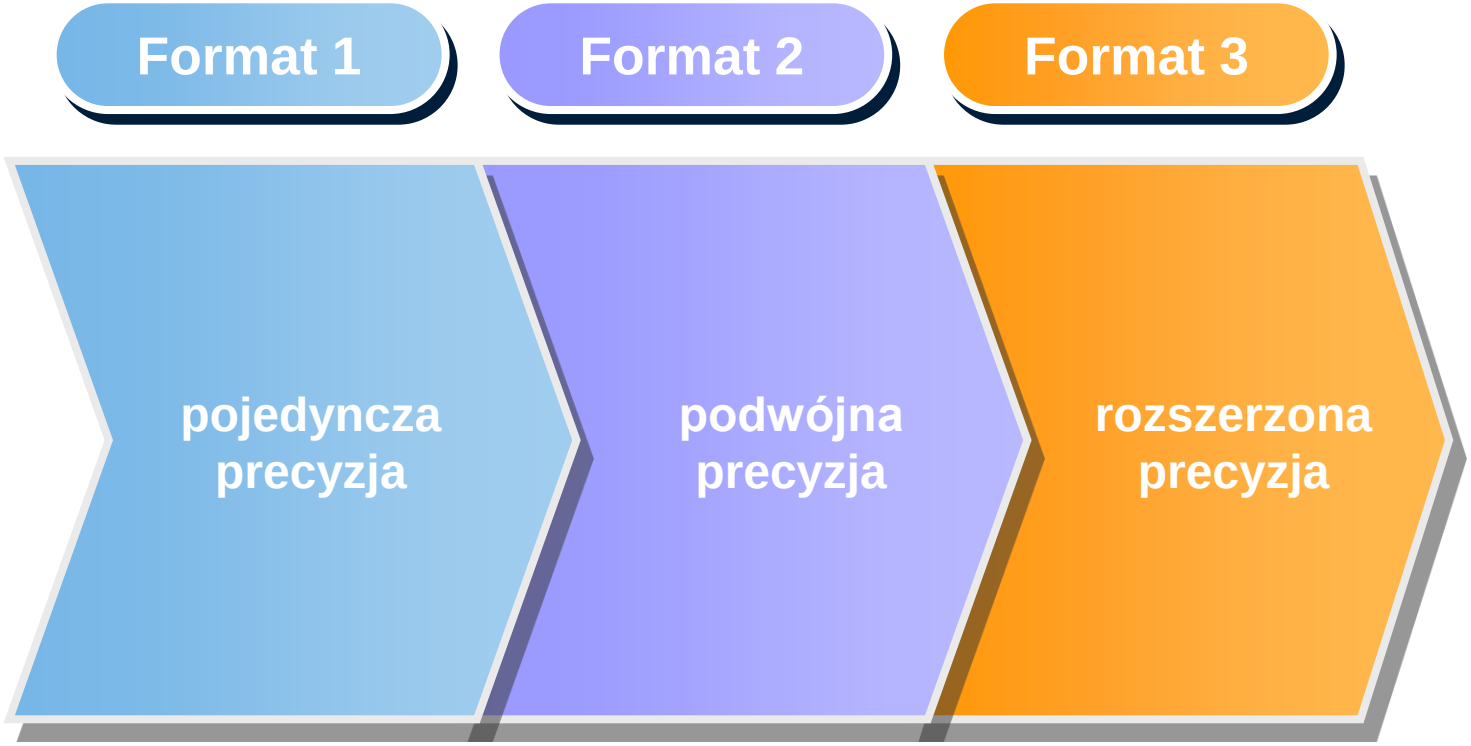
Liczby zmiennoprzecinkowe

- **Większość formatów zmiennoprzecinkowych przyjmuje reprezentowanie liczby rzeczywistej za pomocą zapisu wykładniczego przy zarezerwowaniu pewnej liczby bitów na mantysę liczby i pewnej – zwykle mniejszej - liczby bitów na wykładnik.**
- **W efekcie liczby zmiennoprzecinkowe reprezentują jedynie podzbiór liczb rzeczywistych o określonej liczbie cyfr znaczących.**
- **Przykład:**
  - nasz format liczby zmiennoprzecinkowej będzie dopuszczał mantysę o trzech cyfrach znaczących oraz dziesiętny dwucyfrowy wykładnik. Zarówno mantysa, jak i wykładnik, będą liczbami ze znakiem.



Formaty przyjęte przez IEEE

- Firma Intel wprowadziła trzy formaty zmiennoprzecinkowe:



IEEE 754 – c.d.

- Bity specjalne występują jedynie w wewnętrznej reprezentacji liczy w jednostce zmiennoprzecinkowej i wykorzystywane są do zniwelowania efektu utraty dokładności przy wykonywaniu działań, przez zaokrąglenie.
- Zarówno liczba wejściowa jak i wyjściowa takiego układu nie posiada tych bitów. Oznacza to, że liczby 32/64/128 bitowe, w jednostce zmiennoprzecinkowej, po uzupełnieniu bitami specjalnymi, maja długość odpowiednio 35/67/131 bitów.

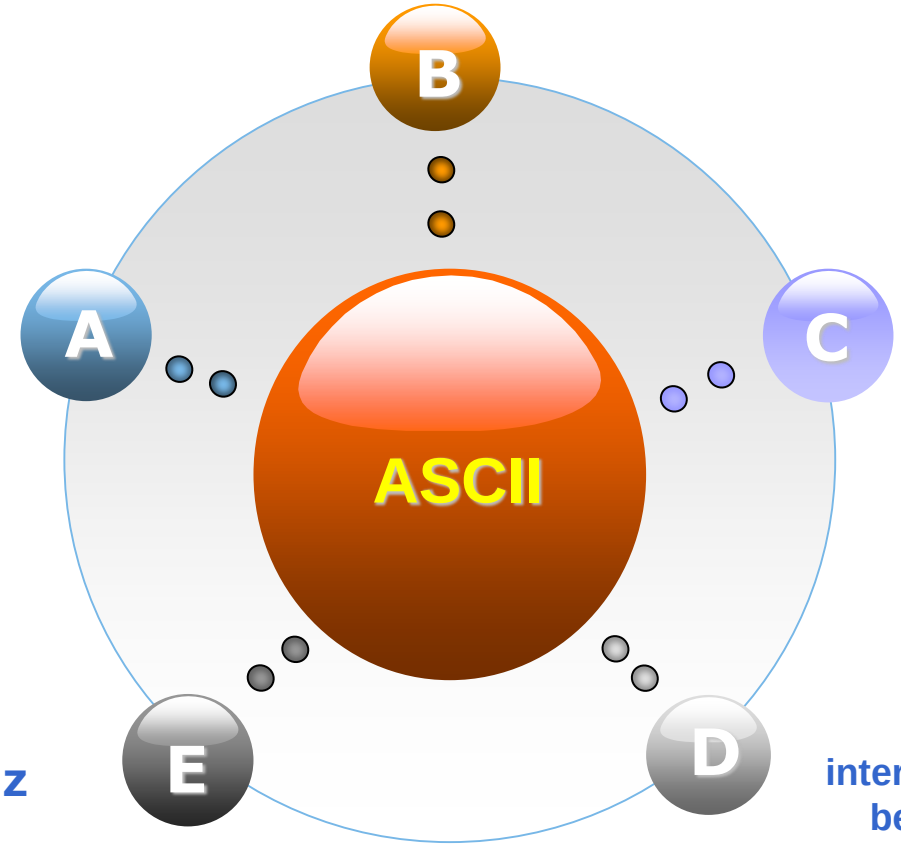
rozmiar liczby	wykładnik E	moduł ułamka M	nazwa
32	8	23	single
64	11	52	double
128	16	111	extended

Podsumowanie

szybka powtórka

Znaki – zestaw znaków ASCII

definiuje 128 znaków



American Standard Code for Information Interchange

standard przyjęty przez znaczną część przemysłu komputerowego

wartości z zakresu od 0 do 127

interpretowane bez znaku wartości całkowite

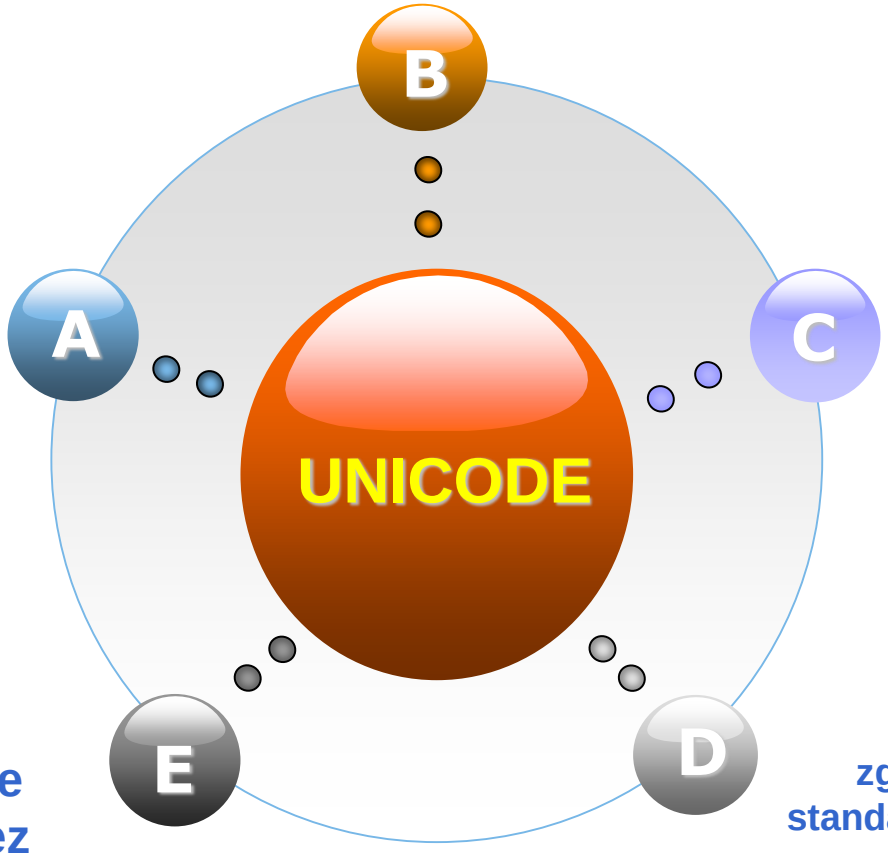
Zestaw znaków ASCII

Zestaw kodów ASCII (American Standard Code for Information Interchange) definiuje odwzorowanie 128 znaków do interpretowanych bez znaku wartości całkowitych z zakresu od 0 do 127.

Standard przyjęty przez znaczną część przemysłu. Jeżeli więc do reprezentowania znaku „A” wykorzystywana jest liczba 65, zgodnie z zestawem znaków ASCII, wtedy można mieć pewność że liczba ta zostanie zinterpretowana jako znak „A” również przez drukarkę czy terminal.

Zestaw znaków UNICODE

pozwala na zdefiniowanie 65536 znaków



A — każdy znak reprezentowany jest słowem szesnastobitowym

B — wypieranie ASCII przez UNICODE

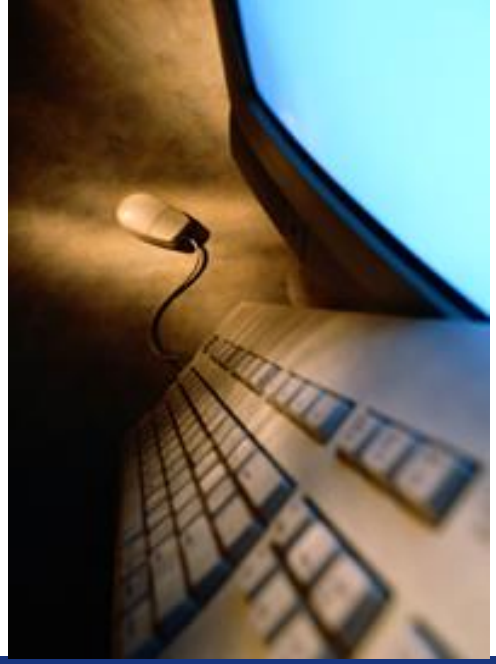
C — dużo większy zakres znaków niż w przypadku standardu ASCII

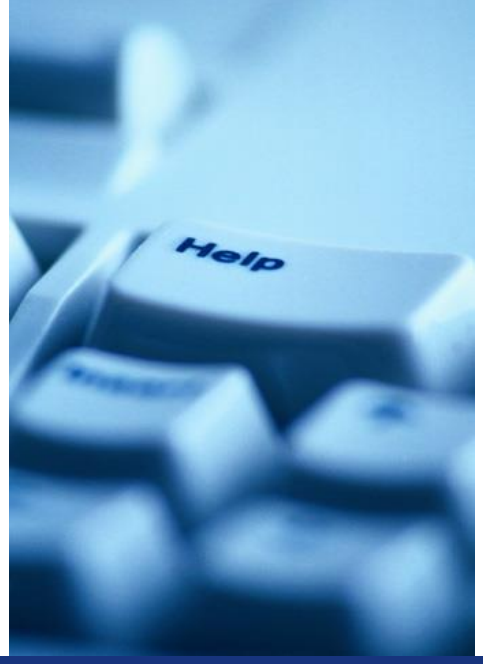
D — zgodny ze standardem ASCII

Źródła

Literatura do wykładu

- S. Kruk, „KURS PROGRAMOWANIA W JĘZYKU ASEMBLER” , wydawnictwo MIKOM 2001.
- B. Drozdowski, „Język asembler dla każdego”, 2008 r.
- R. Hyde, The Art. of Assembly Language”, 2004 r.
- S. Kruk, „ASEMBLER podręcznik użytkownika, 1999 r.
- G. Michałek, „Asembler MINIPRZEWODNIK”, wydawnictwo INFOLAND 2001.





Instytut Elektroniki i Technik Informacyjnych

Dziękuję za uwagę !

Piotr Kisała