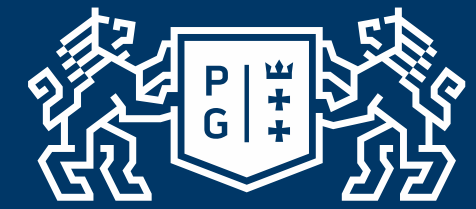


Architektura komputerów

sem. zimowy 2024/2025

cz. 3

Tomasz Dziubich

POLITECHNIKA GDAŃSKA

Przekazywanie parametrów do podprogramów (1)

A. Sposoby przekazywania parametrów:

- *przekazywanie przez wartość* (ang. call by value) – do podprogramu przekazywana jest bezpośrednio wartość parametru, którym może być liczba, tablica liczb, łańcuch znaków, itp;
- *przekazywanie przez adres* (ang. call by location) – do podprogramu przekazywany jest adres lokacji pamięci, w której znajduje się przekazywana wartość.

POLITECHNIKA GDAŃSKA

Przekazywanie parametrów do podprogramów (2)

B. Drogi przekazywania parametrów:

- przez rejestry (w procesorach 64-bitowych, w procesorach klasy RISC)
- przez stos (typowa, powszechnie stosowana metoda).

POLITECHNIKA GDAŃSKA

Przekazywanie parametrów przez stos (1)

- Przekazywanie parametrów do podprogramów (procedur, funkcji) przez stos jest powszechnie stosowane przez kompilatory opracowane dla procesorów 32-bitowych.
- W architekturze 64-bitowej pierwsze cztery parametry przekazywane są przez rejestry procesora, a dopiero ewentualny piąty parametr, szósty, itd. przekazywane są przez stos.
- Technikę przekazywania parametrów przez stos wyjaśnimy na przykładzie podprogramu **suma**, który wyznacza sumę trzech liczb 32-bitowych ze znakiem.

POLITECHNIKA GDAŃSKA

Przekazywanie parametrów przez stos (2)

Przyjmujemy następujące założenia:

- parametry, w postaci liczb 32-bitowych ze znakiem (kod U2), wpisywane są na stos przed wywołaniem podprogramu;
- obliczona wartość wyrażenia (zakładamy, że również 32-bitowa) powinna zostać załadowana do rejestru EAX;
- w trakcie wykonywania obliczeń nie wystąpi nadmiar (przepełnienie);
- obowiązek zdjęcia parametrów ze stosu po wykonaniu obliczeń należy do podprogramu.

POLITECHNIKA GDAŃSKA

Przekazywanie parametrów przez stos (3)

- Jeśli przykładowe wartości parametrów będą wynosiły 17, –5, 7, to wywołanie podprogramu **suma** może mieć postać:

```
push   dword PTR 17
push   dword PTR -5
push   dword PTR 7
call   suma                    ; wywołanie podprogramu
```

POLITECHNIKA GDAŃSKA

Przekazywanie parametrów przez stos (4)

- W wyniku wykonania podanych rozkazów zawartość stosu będzie następująca:

↑	
większe	
adresy	17
	–5
	7
mniejsze	ślad rozkazu CALL
adresy	
↓	

POLITECHNIKA GDAŃSKA

Odczytywanie parametrów przez rozkazy wewnątrz podprogramu (1)

- Odczytywanie wartości parametrów wewnątrz podprogramu za pomocą rozkazu POP byłoby kłopotliwe: wymagałoby uprzedniego odczytania wartości śladu, a po wykonaniu obliczeń należało by ponownie załadować ślad na stos.
- Odczytane parametry można by umieścić w rejestrach ogólnego przeznaczenia — rejestry te jednak używane są do wykonywania obliczeń i przechowywania wyników pośrednich, wskutek czego nadają się do przechowywania jedynie kilku parametrów.

POLITECHNIKA GDAŃSKA

Odczytywanie parametrów przez rozkazy wewnątrz podprogramu (2)

- W celu zorganizowania wygodnego dostępu do parametrów umieszczonych na stosie przyjęto, że obszar zajmowany przez parametry będzie traktowany jako zwykły obszar danych.
- W istocie stos jest bowiem umieszczony w pamięci głównej (operacyjnej) i nic nie stoi na przeszkodzie, by w pewnych sytuacjach traktować jego zawartość jako zwykłe dane.

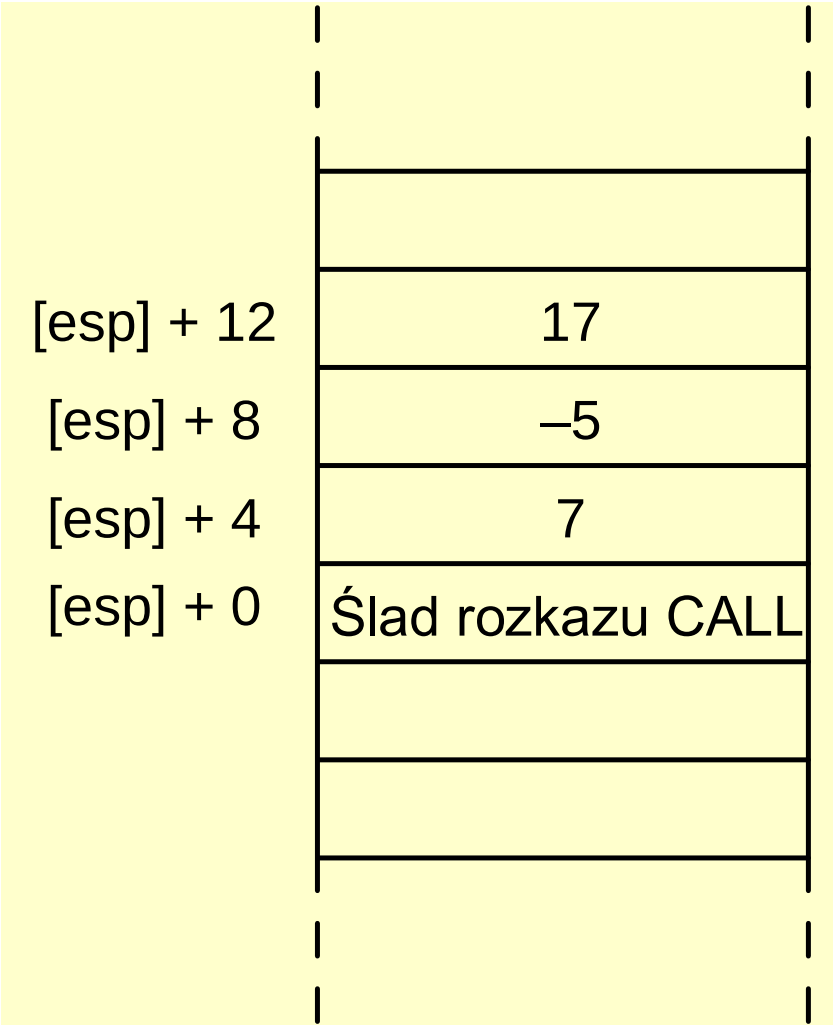
POLITECHNIKA GDAŃSKA

Odczytywanie parametrów przez rozkazy wewnątrz podprogramu (3)

- Taki niestandardowy dostęp do danych zapisanych na stosie wymaga znajomości ich adresów.
- Trzeba przy tym uwzględnić, że położenie danych zapisanych na stosie ustalane jest dopiero w trakcie wykonywania programu, i może się zmieniać przy kolejnych wywołaniach podprogramu — konieczne jest więc opracowanie schematu wyznaczania adresów parametrów umieszczonych na stosie, dostosowanego do aktualnej sytuacji na stosie (każdy parametr zajmuje w pamięci 4 bajty).

POLITECHNIKA GDAŃSKA

Odczytywanie parametrów przez rozkazy wewnątrz podprogramu (4)



POLITECHNIKA GDAŃSKA

Odczytywanie parametrów przez rozkazy wewnątrz podprogramu (5)

- Można zauważyć, że parametry podprogramu odległe są o ustaloną liczbę bajtów od wierzchołka stosu (tu: o 4 bajty, na których zapisany jest ślad).
- Ponieważ położenie wierzchołka stosu wskazuje rejestr ESP, więc na podstawie zawartości rejestru ESP można wyznaczyć adresy parametrów, np. liczba –5 znajduje się w komórce pamięci o adresie równym zawartości rejestru ESP powiększonej o 8.
- Z kolei znając adres komórki pamięci można odczytać jej zawartość posługując się mechanizmem adresowania indeksowego.

POLITECHNIKA GDAŃSKA

Pomocniczy wskaźnik stosu EBP (1)

- Zawartość rejestru ESP może się zmieniać w trakcie wykonywania podprogramu, konieczne jest więc użycie innego rejestru, którego zawartość, ustalona przez cały czas wykonywania podprogramu, będzie wskazywała obszar parametrów na stosie — rolę tę pełni, specjalnie do tego celu zaprojektowany rejestr EBP, często nazywany pomocniczym wskaźnikiem stosu.

POLITECHNIKA GDAŃSKA

Pomocniczy wskaźnik stosu EBP (2)

- Jeśli zawartość rejestru EBP jest równa zawartości ESP, to adres określony przez EBP wskazuje wierzchołek stosu, a EBP zwiększony o 4, 8, 12,... wskazuje kolejne, uprzednio zapisane, wartości 32-bitowe na stosie (należy pamiętać, że stos "rośnie" w kierunku malejących adresów, zatem lokacje bardziej odległe od wierzchołka stosu będą miały wyższe adresy).
- Ponieważ zawartość rejestru EBP jest zazwyczaj potrzebna w programie nadrzędnym, więc przed użyciem EBP w podprogramie trzeba zapamiętać jego zawartość na stosie.

POLITECHNIKA GDAŃSKA

Pomocniczy wskaźnik stosu EBP (3)

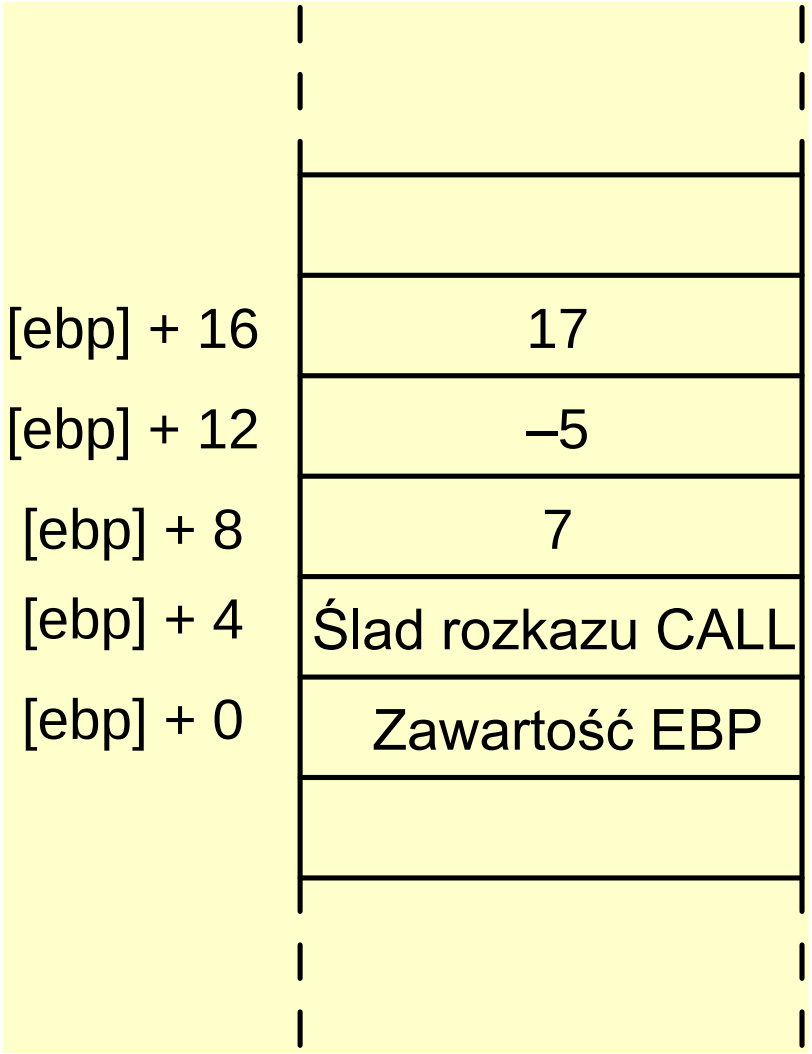
- Zaraz po tym trzeba wpisać do rejestru EBP aktualną zawartość rejestru ESP, tak by możliwy był dostęp do parametrów zapisanych na stosie — zatem dwa pierwsze rozkazy podprogramu suma będą miały postać:

```
suma PROC
        push   ebp      ; przechowanie EBP na stosie
        mov  ebp, esp   ; po wykonaniu tego
                        ; rozkazu, rejestr EBP
                        ; będzie wskazywał
                        ; wierzchołek stosu
```

POLITECHNIKA GDAŃSKA

Pomocniczy wskaźnik stosu EBP (4)

- Aktualna sytuacja na stosie (po wykonaniu dwóch omówionych rozkazów) pokazana jest rysunku.



POLITECHNIKA GDAŃSKA

Kod podprogramu suma

```
suma    PROC
        push ebp        ; przechowanie EBP na stosie
        mov  ebp, esp   ; po wykonaniu tego rozkazu rejestr EBP
                        ; będzie wskazywał wierzchołek stosu
        mov  eax, [ebp] + 8
        add  eax, [ebp] + 12
        add  eax, [ebp] + 16
; wynik obliczeń znajduje się w rejestrze EAX
        pop  ebp        ; odtworzenie rejestru EBP
        ret  12         ; powrót do programu wywołującego
                        ; i usunięcie parametrów ze stosu
suma    ENDP
```

POLITECHNIKA GDAŃSKA

Powrót z podprogramu (1)

Przy tworzeniu podprogramu suma przyjęliśmy założenie, że obowiązek zdjęcia parametrów ze stosu po wykonaniu obliczeń należy do podprogramu.

Wymaganie to można łatwo zrealizować przez zastosowanie rozkazu RET z parametrem.

POLITECHNIKA GDAŃSKA

Powrót z podprogramu (2)

Występujący na końcu przykład rozkaz

```
RET 12
```

powoduje przesłanie zawartości wierzchołka stosu do rejestru EIP (tak jak **`RET`** bez parametru), a następnie zwiększa zawartość wskaźnika stosu ESP o wartość parametru (tu: o 12).

Praktycznie, zwiększenie rejestru ESP o 12 oznacza usunięcie ze stosu trzech parametrów 32-bitowych (zajmują one 12 bajtów).

POLITECHNIKA GDAŃSKA

Zmienne statyczne i dynamiczne (1)

- W typowym programie w języku wysokiego poziomu (np. w C) potrzebne są m.in. zmienne, które muszą być dostępne dla różnych procedur i funkcji przez cały czas wykonywania programu – takie zmienne, zwane *statycznymi*, definiowane są w segmencie danych programu.
- Procedury i funkcje wykonują także działania na *zmiennych lokalnych* — zmienne te, nazywane *dynamicznymi*, definiowane wewnątrz funkcji lub procedury, potrzebne są tylko w trakcie ich wykonywania.

POLITECHNIKA GDAŃSKA

Zmienne statyczne i dynamiczne (2)

- Po zakończeniu wykonywania funkcji lub procedury, zmienne lokalne powinny być usunięte z pamięci.
- Zmienne lokalne powinny zajmować obszar pamięci przydzielany tylko na czas wykonywania funkcji (podprogramu) — wygodnie jest umieścić ten obszar na stosie.
- W celu rezerwacji potrzebnego obszaru pamięci na stosie wystarczy tylko odpowiednio zmniejszyć wskaźnik stosu ESP.

POLITECHNIKA GDAŃSKA

Zmienne statyczne i dynamiczne (3)

- Jeśli w programie zadeklarowane cztery zmienne typu int, np. `int x, y, z, wynik,` to konieczne jest zarezerwowanie 16 bajtów (każda taka zmienna typu `int` w trybie 32-bitowym zajmuje 4 bajty) — na poziomie kodu rozkazowego rezerwacja taka ma postać:

```
sub     esp, 16
```

- Zmniejszenie rej. ESP o 16 można interpretować jako zapisanie na stos 16 bajtów o nieokreślonej wartości początkowej — powstaje w ten sposób zarezerwowany obszar 16-bajtowy, w którym będą zapisywane wartości zmiennych lokalnych.

POLITECHNIKA GDAŃSKA

Zmienne statyczne i dynamiczne (4)

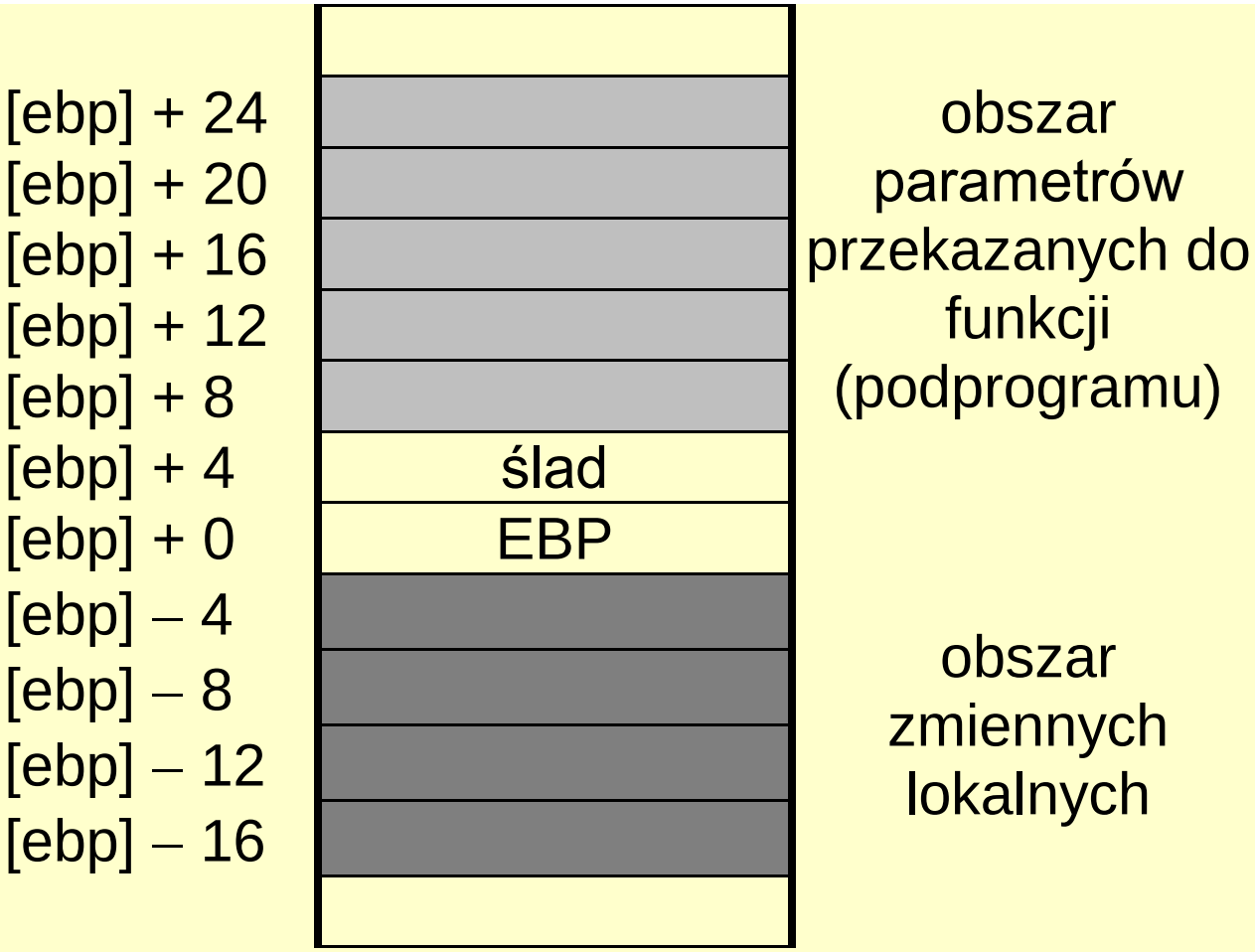
- Zazwyczaj omawiana rezerwacja wykonywana jest w początkowej części podprogramu

```
push  ebp     ; przechowanie EBP
mov   ebp, esp
sub   esp, 16 ;rezerwacja 16 bajtów
```

- W rezultacie na stosie, obok istniejącego już obszaru zawierającego parametry przekazywane do funkcji, pojawia się inny obszar, w którym przechowywane są zmienne lokalne (np. x, y, z, wynik).

POLITECHNIKA GDAŃSKA

Zmienne statyczne i dynamiczne (5)



POLITECHNIKA GDAŃSKA

Zmienne statyczne i dynamiczne (6)

- Dostęp do zmiennych lokalnych realizowany jest również za pomocą pomocniczego wskaźnika stosu EBP, przy czym w wyrażeniach adresowych występuje znak minus, np. [ebp – 12]

- Przydzielony obszar pamięci zwalnia się w końcowej części podprogramu poprzez odpowiednie zwiększenie wskaźnika stosu ESP lub przepisanie ESP ← EBP

POLITECHNIKA GDAŃSKA

Ramka stosu (1)

Tak zorganizowany podprogram wykonuje działania na danych dostępnych w:

- na stosie umieszczone są wartości parametrów, które zostały przekazane przez program wywołujący;
- na stosie przechowywane są także wartości zmiennych lokalnych;
- zmienne globalne (np. w języku C deklarowane na zewnątrz funkcji) przechowywane w odrębnym obszarze danych statycznych, dostępnym dla wszystkich podprogramów.

POLITECHNIKA GDAŃSKA

Ramka stosu (2)

- Obszar stosu, w którym zawarte są zmienne lokalne podprogramu, przekazane parametry i ślad rozkazu CALL nazywany jest **ramką stosu** .
- Część ramki stosu jest już umieszczona na stosie w chwili wywołania podprogramu — stanowi ona *rekord aktywacji* .
- W takim ujęciu zawartość rejestru EBP można uważać za wskaźnik ramki: rejestr EBP wskazuje dane zawarte w bieżącej ramce.
- W literaturze wskaźnik ramki oznaczany jest skrótem FP (ang. *frame pointer*).

POLITECHNIKA GDAŃSKA

Interfejs ABI (1)

- Współczesne systemy oprogramowania składają się wielu modułów, w tym bibliotecznych, które muszą współdziałać na poziomie kodu binarnego — nierzadko kody źródłowe tych modułów tworzone są w różnych językach programowania.
- W tej sytuacji konieczne jest wprowadzenie reguł i ustaleń opisujących zasady współpracy, na poziomie kodu binarnego, między programami, bibliotekami i systemem operacyjnym — taki zbiór reguł i ustaleń nazwano interfejsem ABI (ang. Application Binary Interface).

POLITECHNIKA GDAŃSKA

Interfejs ABI (2)

- ABI definiuje *standard wywoływania* (ang. calling convention), który określa sposób wywoływania funkcji, przekazywania jej argumentów, przejmowania obliczonej wartości, podaje wykaz rejestrów, których zawartości powinny być zachowane, itp.; m.in. standard ten określa czy parametry przekazywane są przez rejestry czy przez stos, a jeśli przez stos to w jakiej kolejności są ładowane, czy dopuszcza się zmienną liczbę argumentów, itd.

POLITECHNIKA GDAŃSKA

Standardy wywoływania funkcji (1)

- Kompilatory języków programowania stosują kilka typowych sposobów przekazywania parametrów do funkcji, znanych jako:

  standard Pascal,

  standard C

  standard StdCall

- Główne różnice między standardami dotyczą kolejności ładowania parametrów na stos i obowiązku usuwania parametrów.

POLITECHNIKA GDAŃSKA

Standardy wywoływania funkcji (2)

- W trybie 32-bitowym czasami stosuje się też standard *FastCall*, w którym parametry przekazywane są przez rejestry procesora.
- W tym przypadku standard nie jest ustalony, np. kompilatory Microsoft i Gnu przekazują dwa pierwsze parametry przez rejestry ECX i EDX, a dalsze przez stos wg reguł *StdCall* .

Standard	Kolejność ładowania na stos	Obowiązek zdjęcia parametrów
Pascal	od lewej do prawej	wywołany podprogram
C	od prawej do lewej	program wywołujący
StdCall	od prawej do lewej	wywołany podprogram

POLITECHNIKA GDAŃSKA

Standardy wywoływania funkcji (3)

	Aplikacje 16-bitowe DOS, Windows	Aplikacje 32-bitowe Windows, Linux
Rejestry używane bez ograniczeń	AX, BX, CX, DX, ES, ST(0) – ST(7)	EAX, ECX, EDX, ST(0) – ST(7), XMM0 – XMM7
Rejestry, które muszą być zapamiętywane i odtwarzane	SI, DI, BP, DS	EBX, ESI, EDI, EBP
Rejestry, które nie mogą być zmieniane		DS, ES, FS, GS, SS
Rejestry używane do przekazywania parametrów		(ECX)
Rejestry używane do zwracania wartości	AX, DX, ST(0)	EAX, EDX, ST(0)

POLITECHNIKA GDAŃSKA

Standardy wywoływania funkcji (4)

	Aplikacje 64-bitowe Windows	Aplikacje 64-bitowe Linux
Rejestry używane bez ograniczeń	RAX, RCX, RDX, R8 – R11, ST(0) – ST(7), XMM0 – XMM5	RAX, RCX, RDX, RSI, RDI, R8 – R11, ST(0) – ST(7), XMM0 – XMM15
Rejestry, które muszą być zapamiętywane i odtwarzane	RBX, RSI, RDI, RBP, R12 – R15, XMM6 – XMM15	RBX, RBP, R12 – R15
Rej., które nie mogą być zmieniane		
Rejestry używane do przekazywania parametrów	RCX, RDX, R8, R9, XMM0 – XMM3	RDI, RSI, RDX, RCX, R8, R9, XMM0 – XMM7
Rejestry używane do zwracania wartości	RAX, XMM0	RAX, RDX, XMM0, XMM1,ST(0), ST(1)

POLITECHNIKA GDAŃSKA

Standardy wywoływania funkcji (5)

- Znaczniki operacji arytmetycznych i logicznych (w rejestrze znaczników) mogą być używane bez ograniczeń.
- Znacznik DF powinien być zerowany zarówno przed wywołaniem podprogramu, jak i wewnątrz podprogramu przed rozkazem **`RET`**, jeśli używane były rozkazy operacji blokowych (np. **`MOVSB`**).

POLITECHNIKA GDAŃSKA

Standardy wywoływania funkcji (6)

Nazwa	System operacyjny/ kompilator	Paramtery	Kolejność paramterów	Strona czyszcząca stos
cdecl	Microsoft, Unix-like (GCC)	Stos	RTL	Caller
stdcall	Microsoft	stos	RTL	Callee
fastcall	Microsoft	ECX, EDX	RTL	Callee
register	Delphi, Pascal, Linux	EAX, EDX, ECX	LTR	Callee
thiscall	Microsoft VC++	ECX	RTL	Callee
vectorcall	Microsoft VC++	ECX, EDX, [XY]MM0–5	RTL	Callee

POLITECHNIKA GDAŃSKA

Program przykładowy C/ASM (1)

- Program podany na kolejnych slajdach wyznacza sumę wszystkich elementów podanej tablicy, a także wyznacza sumy liczb ujemnych i dodatnich. Obliczenia ww. sum realizowane jest za pomocą kodu w asemblerze.
- Obliczona suma wszystkich elementów przekazywana jest przez nazwę funkcji, a sumy liczb ujemnych i dodatnich – przez argumenty funkcji (w postaci wskaźników).

Program przykładowy C/ASM (2)

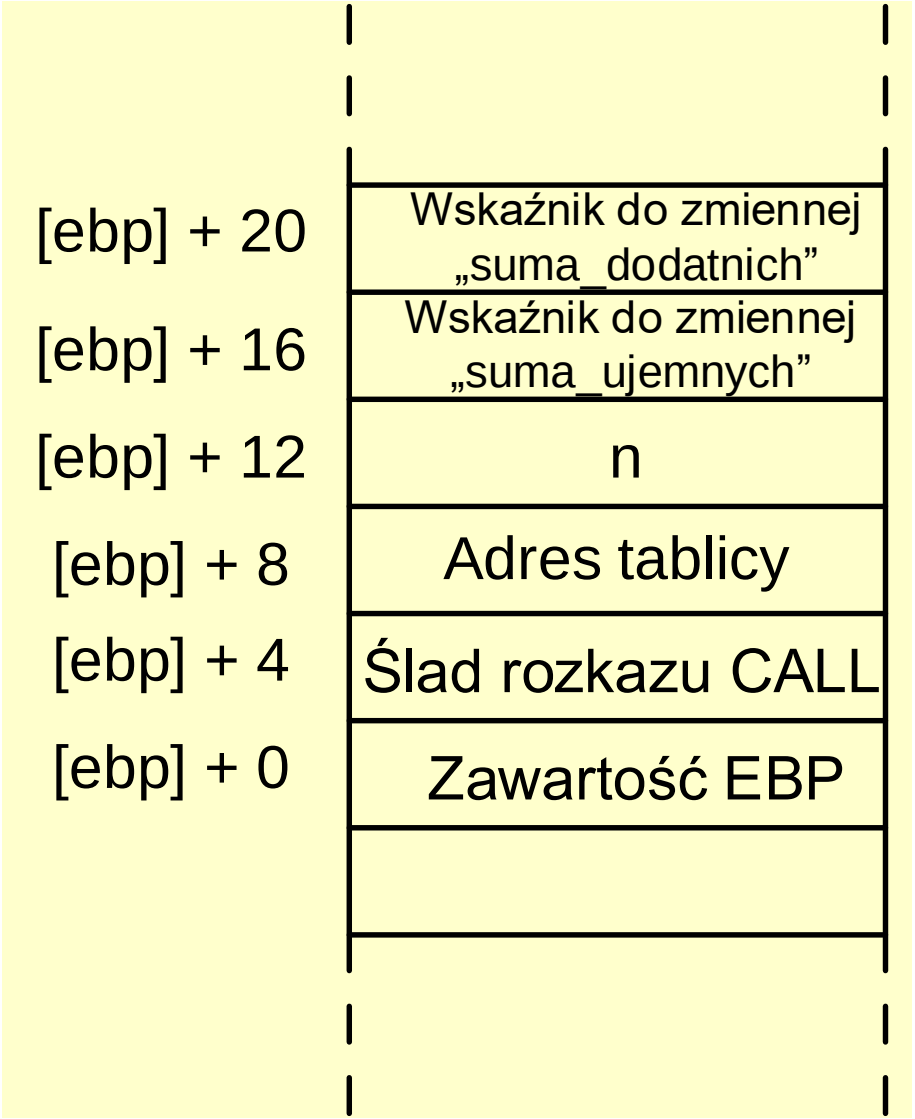
```
#include <stdio.h>
extern int sumy_liczb (int * tablica, int n,
      int * suma_ujemnych, int * suma_dodatnich);
int main ()
{ int punkty[5] = {-3, 14, 21, 0, -1};
  int ujemne, dodatnie, wszystkie;

  wszystkie = sumy_liczb (punkty, 5, &ujemne,
  &dodatnie);
  printf("\nSuma wszystkich liczb = %d",
  wszystkie);
  printf("\nSuma dodatnich = %d\nSuma ujemnych =\
  %d\n", dodatnie, ujemne);
  return 0;
}
```

POLITECHNIKA GDAŃSKA

Program przykładowy C/ASM (3)

- Sytuacja na stosie po wykonaniu dwóch pierwszych rozkazów podprogramu



Program przykładowy C/ASM (4)

```
; obliczanie sumy liczb dodatnich i sumy liczb ujemnych
; oraz sumy wszystkich liczb

.686
.model flat
public _sumy_liczb
.code

_sumy_liczb    PROC
; funkcja wywoływana jest z poziomu języka C,
; prototyp ma postać:
; extern int sumy_liczb (int * tablica, int n,
;     int * suma_ujemnych, int * suma_dodatnich);

  push   ebp
  mov          ebp, esp
```

Program przykładowy C/ASM (5)

```
; przechowanie rejestrów EBX, ESI, EDI
  push    ebx
  push    esi
  push    edi

  mov     ebx, [ebp+8]  ; adres tablicy liczb
  mov     ecx, [ebp+12] ; rozmiar tablicy

  mov     edx, 0    ; początkowa wartość
                    ; sumy wszystkich liczb
  mov     esi, 0    ; początkowa wartość sumy liczb ujem.
  mov     edi, 0    ; początkowa wartość sumy liczb dodatn.
```

POLITECHNIKA GDAŃSKA

Program przykładowy C/ASM (6)

```
ptl:
  add  edx, [ebx]              ; dodanie kolejnego elementu do
                               ; sumy wszystkich liczb
  cmp  dword PTR [ebx], 0      ; porównanie kolejnego
                               ; elementu tablicy
  jl   ujemna                  ; skok. gdy liczba ujemna
  add  edi, [ebx]              ; aktualizacja sumy liczb dodatnich
  jmp  dalej
ujemna:
  add  esi, [ebx]  ; aktualizacja sumy liczb ujemnych
dalej:
  add  ebx, 4      ; zwiększenie indeksu o 4, tak by
                   ; wskazywał następny element tablicy
  loop  ptl        ; sterowanie pętlą
```

Program przykładowy C/ASM (7)

```
; adresy zmiennych, do których zostaną wpisane sumy liczb:
; ujemnych [ebp+16], dodatnich [ebp+20]
  mov  ebx, [ebp+16]
  mov  [ebx], esi ; przesłanie sumy liczb ujemnych
  mov  ebx, [ebp+20]
  mov [ebx], edi  ; przesłanie sumy liczb dodatnich
  mov  eax, edx   ; przesłanie sumy wszystkich liczb
  pop  edi        ; odtworzenie zawartości rejestrów
  pop  esi
  pop  ebx
  pop  ebp
  ret             ; powrót z podprogramu
_sumy_liczb  ENDP
END
```

POLITECHNIKA GDAŃSKA

Funkcje systemowe (1)

- W początkowym okresie rozwoju informatyki każdy program był całkowicie samodzielny i w trakcie wykonywania nie korzystał z pomocy innych programów — tego rodzaju rozwiązania spotyka się współcześnie w mikrokomputerach (mikrokontrolerach) obsługujących nieskomplikowane urządzenia.

POLITECHNIKA GDAŃSKA

Funkcje systemowe (2)

- Zauważono, że wiele programów ma takie same lub bardzo podobne fragmenty, związane głównie z obsługą urządzeń wejścia/wyjścia (wówczas dalekopisu) — te fragmenty kodu zostały wyodrębnione i została im nadana postać podprogramów.
- Podprogramy te zostały udostępnione innym programom, które mogły je wywoływać w trakcie wykonywania — w ten sposób zaczęły powstawać systemy operacyjne.
- Później obudowano te podprogramy dodatkowymi modułami, których zadaniem było zarządzanie systemem komputerowym.

POLITECHNIKA GDAŃSKA

Funkcje systemowe (3)

- Programy wykonywane we współczesnych komputerach ogólnego przeznaczenia zazwyczaj nie mogą bezpośrednio sterować pracą sprzętu, ale jedynie za pośrednictwem funkcji systemowych udostępnianych przez system operacyjny.
- Dokumentacja systemu operacyjnego dostarczana użytkownikom (programistom) zawiera katalog funkcji systemowych, które mogą być wywoływane z poziomu programu.

POLITECHNIKA GDAŃSKA

Interfejs programowania aplikacji (API) (1)

- Dla każdej funkcji systemowej w katalogu podane są szczegółowe informacje o wymaganych parametrach i sposobach przekazywania ich do podprogramów systemowych.
- W ten zostaje określony pewien interfejs określający sposób porozumiewania się programu z systemem operacyjnym — interfejs ten oznaczany jest skrótem API (ang. application programming interface), co tłumaczy się jako interfejs programowania aplikacji.

POLITECHNIKA GDAŃSKA

Interfejs programowania aplikacji (API) (2)

- W aktualnie używanych systemach MS Windows dostępny jest interfejs Win32 API; w komputerach wyposażonych w system Linux używany jest interfejs zgodny ze standardem POSIX (ang. Portable Operating System Interface).
- Podstawowy zestaw Win32 API obejmuje około 1000 funkcji systemowych, które definiują zarządzanie procesami i wątkami, operacje na plikach, operacje graficzne, i wiele innych; zestaw jest w trakcie ciągłego rozwoju — pojawienie się nowych zastosowań powoduje rozszerzenie zestawu o kolejne grupy funkcjonalne, jak np. DirectX, OpenGL, WinSock, WNet i inne.

POLITECHNIKA GDAŃSKA

Funkcja GetComputerName (1)

- Wywołanie funkcji systemowej GetComputerName spowoduje wpisanie nazwy komputera do wskazanego obszaru pamięci.
- Poniżej podano fragmenty oryginalnego opisu katalogowego (Win32 API) funkcji GetComputerName, a następnie przykład wywołania tej funkcji na poziomie kodu w asemblerze.

POLITECHNIKA GDAŃSKA

Funkcja GetComputerName – fragment opisu oryginalnego (1)

GetComputerName

The **GetComputerName** function retrieves the computer name of the current system. This name is established at system startup, when it is initialized from the registry.

```
BOOL GetComputerName (
    LPTSTR  lpBuffer,  // address of name buffer
    LPDWORD   nSize // addr. of size of name buffer
);
```

POLITECHNIKA GDAŃSKA

Funkcja GetComputerName – fragment opisu oryginalnego (2)

Parameters

lpBuffer

Pointer to a buffer that receives a null-terminated string containing the computer name. The buffer size should be large enough to contain MAX_COMPUTERNAME_LENGTH + 1 characters.

nSize

Pointer to a **DWORD** variable. On input, the variable specifies the size, in bytes or characters, of the buffer. On output, the variable returns the number of bytes or characters copied to the destination buffer, not including the terminating null character.

If the buffer is too small, the function fails.

Return Values

If the function succeeds, the return value is nonzero. If the function fails, the return value is zero.

POLITECHNIKA GDAŃSKA

Funkcja GetComputerName – przykład wywołania w asemblerze (1)

```
.data
nazwa           db          80 dup ('?')
rozmiar         dd          80

.code
    push        dword PTR offset rozmiar
    push        dword PTR offset nazwa
    call        _GetComputerNameA@8

    cmp         eax, 0
    je          blad                ; skok, gdy wystąpił błąd
```

POLITECHNIKA GDAŃSKA

Funkcja GetComputerName – przykład wywołania w asemblerze (2)

; wyświetlenie nazwy komputera

```
push    rozmiar
push    dword PTR offset nazwa
push    dword PTR 1
call    __write
add     esp, 12
```

POLITECHNIKA GDAŃSKA

Funkcja GetComputerName – przykład wywołania w asemblerze (3)

- Warto zwrócić uwagę na rozkaz push rozmiar, który podaje dla funkcji write liczbę wyświetlanych znaków.
- Pierwotna zawartość zmiennej rozmiar wynosiła 80, jednak zgodnie z podanym opisem funkcji GetComputerName zawarta tam liczba, w wyniku wykonania funkcji zostaje zastąpiona przez liczbę znaków nazwy komputera (bez bajtu zerowego).
- Zatem mamy tu do czynienia z przekazywaniem obliczonej wartości przez argument funkcji, który został określony jako adres zmiennej rozmiar.

POLITECHNIKA GDAŃSKA

Inne interfejsy programowania w systemie Windows i Linux (1)

- Oprócz interfejsu Win32 API, system Windows (z wyjątkiem wersji 64-bitowych) udostępnia także inny interfejs przejęty z używanego dawniej systemu DOS — interfejs ten określony jest na poziomie asemblera i może być używany jedynie w 16-bitowych programach wykonywanych w trybie V86 (lub w trybie rzeczywistym).
- W tym przypadku funkcje systemowe wywołuje się za pomocą rozkazu INT 21H, który można uważać za odmianę rozkazu CALL.

POLITECHNIKA GDAŃSKA

Inne interfejsy programowania w systemie Windows i Linux (2)

- Adresy podprogramów realizujących funkcje systemowe nie są znane w trakcie translacji programu — zastąpienie nazw funkcji systemowych przez odpowiednie adresy wykonuje system operacyjny bezpośrednio po załadowaniu programu do pamięci.
- Ze względu na ograniczenia sprzętowe we wczesnych wersjach komputerów osobistych, w systemie DOS przyjęto, że funkcje systemowe identyfikowane są przez numery katalogowe, a nie przez nazwy funkcji.

POLITECHNIKA GDAŃSKA

Inne interfejsy programowania w systemie Windows i Linux (3)

- Podany numer funkcji jest traktowany jako indeks w specjalnej tablicy adresowej, której elementami są adresy funkcji systemowych.
- W przypadku wytworzenia nowej wersji systemu operacyjnego, adresy poszczególnych funkcji ulegają zmianie, ale sposób ich wywoływania za pomocą indeksu w tablicy pozostaje niezmieniony.
- W architekturze x86 koncepcja ta została zrealizowana za pomocą tablicy adresowej zwanej ***tablicą wektorów (deskryptorów) przerwań***.

Tablica wektorów przerwań (1)

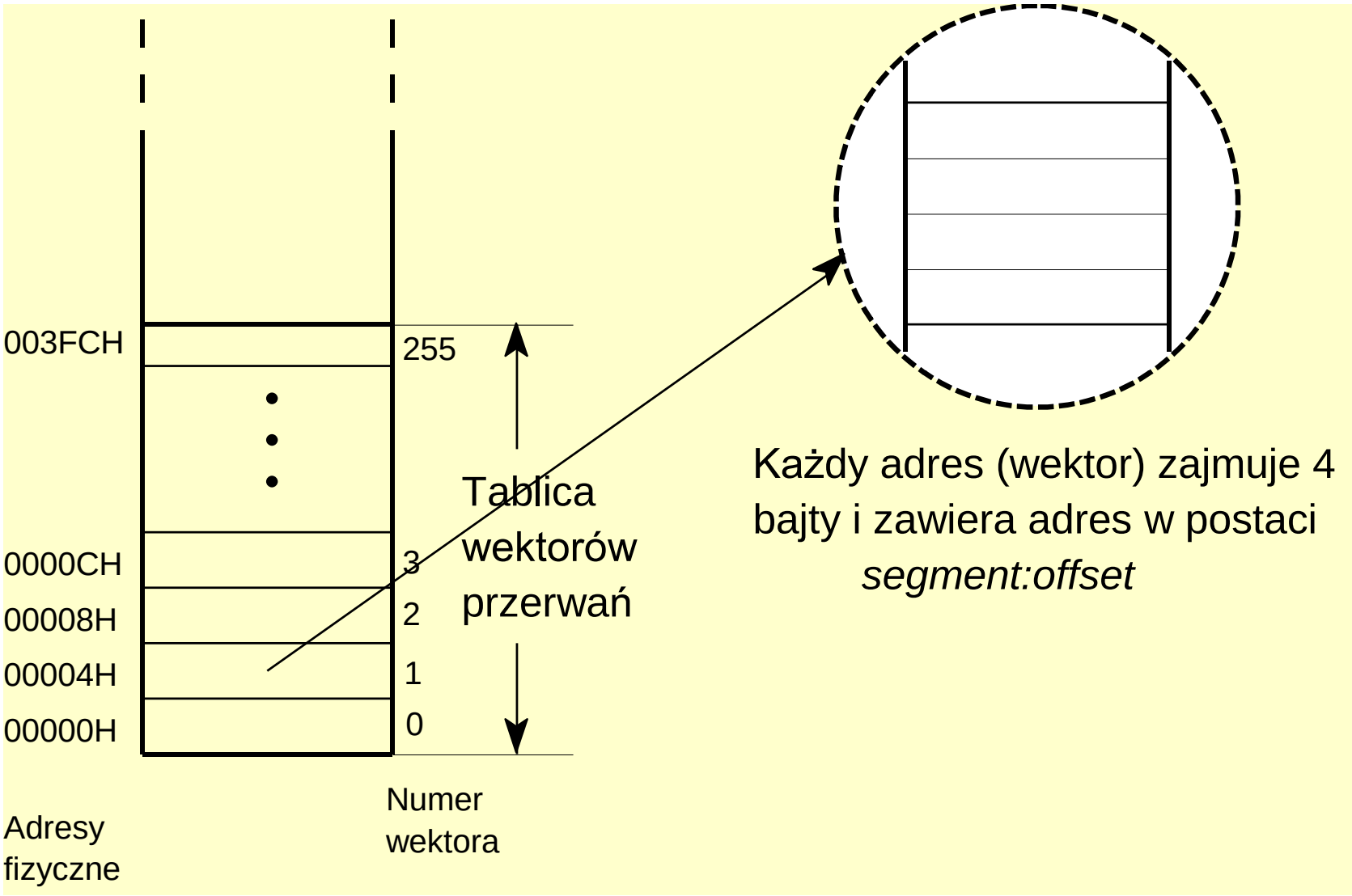
- Używane obecnie procesory zgodne z architekturą x86, bezpośrednio po uruchomieniu pracują w trybie rzeczywistym, prawie identycznym z trybem pracy stosowanym przez procesory 8086/8088 (będące pierwowzorem całej rodziny).
- Tryb rzeczywisty oferuje dość ubogie mechanizmy adresowania i słabą pomoc dla systemu operacyjnego, wobec czego po dokonaniu wstępnych testów (kilkadziesiąt sekund) system operacyjny przełącza procesor do trybu chronionego, w którym dostępne są rozbudowane mechanizmy adresowania i ochrony systemu.

POLITECHNIKA GDAŃSKA

Tablica wektorów przerwań (2)

- W odniesieniu do trybu rzeczywistego używany jest termin *tablica wektorów przerwań*, natomiast w trybie chronionym mówimy o *tablicy deskryptorów przerwań* — funkcjonalnie obie tablice pełnią identyczną rolę.
- Podany dalej opis dotyczy *tablicy wektorów przerwań* — ze względu na brak mechanizmów ochrony w trybie rzeczywistym, tablica wektorów pozwala na wykonywanie interesujących eksperymentów w zakresie sprzętu i oprogramowania.
- W trybie chronionym podobne eksperymenty są trudne lub niemożliwe do realizacji.

Tablica wektorów przerwań (3)



Tablica wektorów przerwań (4)

- Tablica wektorów przerwań zawiera 256 adresów 4-bajtowych i umieszczona jest w pamięci operacyjnej począwszy od adresu fizycznego 0.
- Adresy podprogramów w tej tablicy zapisywane są w układzie *segment:offset*; oba pola są 16-bitowe, a pole *offset* zajmuje dwa bajty o niższych adresach.
- Adres fizyczny w tym przypadku określa formuła

segment*16 + offset

POLITECHNIKA GDAŃSKA

Wywoływanie podprogramów systemowych za pomocą rozkazu INT (1)

- Do wywoływania podprogramów systemowych za pośrednictwem tablicy wektorów przerwań zdefiniowano rozkaz **`INT`**.
- Rozkaz **`INT`** należy klasyfikować jako rozkaz skoku do podprogramu typu pośredniego, podobny do rozkazu **`CALL`**.
- Jednocześnie rozkaz **`INT`** wykazuje zewnętrzne podobieństwo do (dalej omawianych) przerwań sprzętowych i tego powodu został nazwany *INTerrupt*, czyli przerwanie — podobieństwo to może utrudniać zrozumienie istoty mechanizmu przerwań sprzętowych.

POLITECHNIKA GDAŃSKA

Wywoływanie podprogramów systemowych za pomocą rozkazu INT (2)

- Rozkaz **`INT`** pozostawia ślad na stosie zawierający rejestry (E)IP, CS i rejestr znaczników (E)FLAGS; następnie wykonuje do podprogramu, którego adres określony jest przez zawartość wektora przerwania o numerze określonym przez argument rozkazu **`INT`**.
- Przykładowo, rozkaz **`INT 21H`** powoduje skok do podprogramu, którego adres zawarty jest w wektorze o numerze 33 (=21H).

POLITECHNIKA GDAŃSKA

Wywoływanie podprogramów systemowych za pomocą rozkazu INT (3)

- Ze względu na inną strukturę śladu, zdefiniowano oddzielny rozkaz powrotu z podprogramu **`IRET`** — rozkaz ten jest używany w przypadku gdy podprogram został wywołany za pomocą rozkazu **`INT`** albo wskutek wystąpienia przerwania sprzętowego lub wyjątku procesora.

POLITECHNIKA GDAŃSKA

Przykład wywołania za pomocą rozkazu INT w systemie Windows (1)

- W omawianym interfejsie systemu Windows, odziedziczonym po systemie DOS, poszczególnym funkcjom systemowym przypisano numery katalogowe.
- Przed wywołaniem funkcji (rozkaz **`INT 21H`**) do rejestru AH należy wpisać odpowiedni numer katalogowy, a dodatkowe parametry umieszcza się w innych rejestrach.

POLITECHNIKA GDAŃSKA

Przykład wywołania za pomocą rozkazu INT w systemie Windows (2)

- Poniższy fragment programu wyświetla na ekranie literę A za pomocą funkcji o numerze katalogowym 2

```
mov dl, 'A'  ; kod ASCII znaku wpisywany
             ; jest do rejestru dl
mov ah, 2    ; nr katalogowy funkcji
int 21H      ; wywołanie funkcji systemowej
```

- Ta sama technika wywoływania podprogramów systemowych używana jest także w omawianym dalej systemie BIOS.

POLITECHNIKA GDAŃSKA

Przykład wywołania za pomocą rozkazu INT w systemie Windows (3)

- W literaturze omawiany interfejs API określany często jest nieścisłym terminem przerwania DOSu.
- Zaletą tego interfejsu są m.in. proste operacje wprowadzania i wyprowadzania pojedynczych znaków (np. funkcja katalogowa nr 1 odczytuje znak z klawiatury), co koresponduje ze stylem programów pisanych w asemblerze.
- Jednak firma Microsoft stopniowo eliminuje omawiany interfejs — nie jest on już dostępny w 64-bitowych wersjach systemu Windows.

POLITECHNIKA GDAŃSKA

Przykład wywołania za pomocą rozkazu INT w systemie Linux (1)

- W podobny sposób mogą być wywoływane podprogramy systemowe w systemie Linux. Dla programów kodowanych w asemblerze dostępny jest zestaw funkcji systemowych, które wywołuje się za pomocą rozkazu **`INT 80H`**.
- W tym przypadku numer funkcji systemowej podaje się w rejestrze EAX, a pozostałe parametry wywołania w innych rejestrach.
- Katalog funkcji systemowych zawiera obszerny zestaw funkcji zgodnych ze specyfikacją POSIX. Opisy poszczególnych funkcji systemowych można znaleźć na wielu stronach internetowych.

POLITECHNIKA GDAŃSKA

Przykład wywołania za pomocą rozkazu INT w systemie Linux (2)

- Przykładowo, wywołanie **`INT 80H`** przy EAX = 3 powoduje odczytanie zawartości pliku, przy czym wcześniej do rejestru EBX należy wpisać deskryptor (dojście) pliku, do ECX adres obszaru pamięci, do którego zostanie wpisana zawartość pliku, a do rejestru EDX liczbę bajtów do przeczytania.

POLITECHNIKA GDAŃSKA

System BIOS (1)

- Bezpośrednio po uruchomieniu (lub zresetowaniu) komputera procesory x86 pobierają pierwszy rozkaz z komórki pamięci o adresie FFFF0H — zazwyczaj znajduje się tam rozkaz skoku bezwarunkowego prowadzący do programów rozruchowych, dokonujących wstępnej inicjalizacji sprzętu w komputerze.
- Omawiane programy rozruchowe przechowywane są w pamięci ROM (ang. read-only memory — pamięć tylko do odczytu), która jest nagrywana w trakcie produkcji komputerów; obecnie używa się zwykle pamięci EEPROM, co pozwala na późniejszą aktualizację oprogramowania.

POLITECHNIKA GDAŃSKA

System BIOS (2)

- W pamięci ROM obok ww. inicjalizujących umieszcza się także programy testujące i konfigurujące zainstalowany sprzęt — wstępne testowanie sprzętu przed uruchomieniem systemu operacyjnego zapobiega przed dalszymi powikłaniami, które mogą doprowadzić do uszkodzenia oprogramowania i danych.
- W pamięci ROM przechowywany jest także pakiet podprogramów wykonujących podstawowe czynności sterowania i obsługi urządzeń (zwykle na poziomie portów).

POLITECHNIKA GDAŃSKA

System BIOS (3)

- Można więc powiedzieć, że w pamięci ROM znajduje się mini-system operacyjny — w komputerach PC system ten nosi nazwę BIOS — Basic Input Output System.
- System BIOS zawiera m.in. zestaw programów testujących POST (ang. Power On Self Test), uruchamianych po włączeniu komputera.
- Funkcje udostępniane przez BIOS dla innych programów tworzą BIOS API, który oferuje elementarny zestaw operacji, niezależny od konstrukcji konkretnego komputera (zatem oprogramowanie BIOSu może "wyrównywać" różnice w konstrukcji sprzętu).

POLITECHNIKA GDAŃSKA

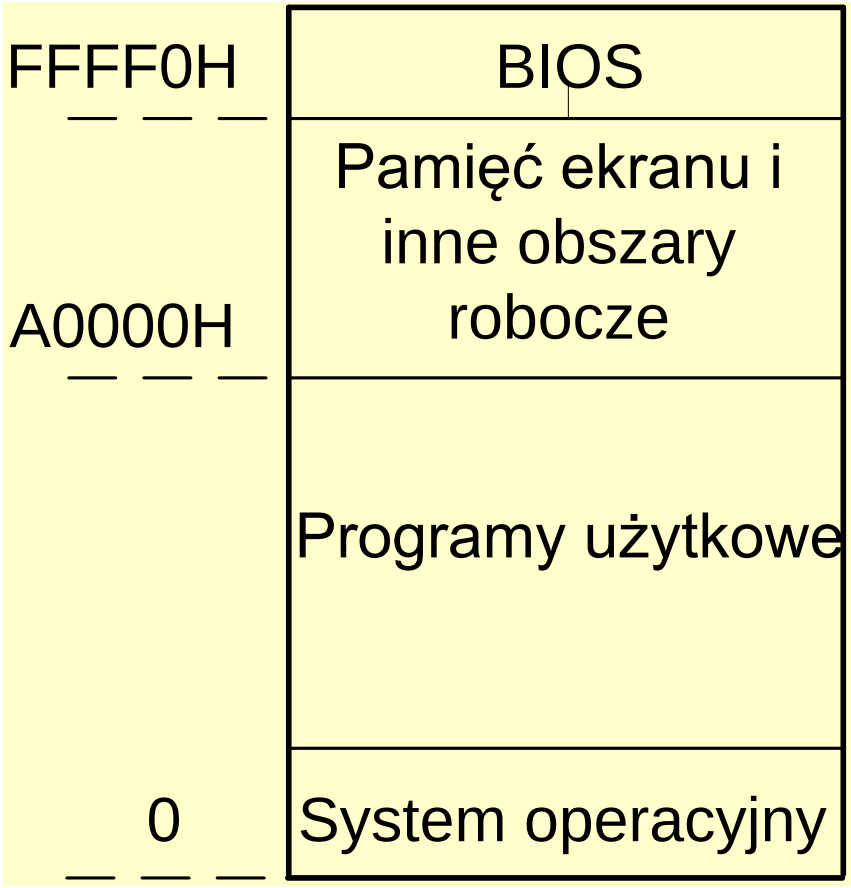
System BIOS (4)

- BIOS posiada także mechanizmy pozwalające na dynamiczną konfigurację systemu, w zależności od wyposażenia komputera.
- BIOS został tak skonstruowany, że niektóre jego fragmenty mogą być umieszczone są na kartach rozszerzeniowych komputera (np. na karcie sterownika graficznego).

POLITECHNIKA GDAŃSKA

System BIOS (5)

- BIOS zapisany jest w pamięci ROM i w przestrzeni adresowej 1 MB zajmuje obszar pamięci począwszy od adresu fizycznego F0000H (lub F000H:0000H w formacie *segment:offset*).

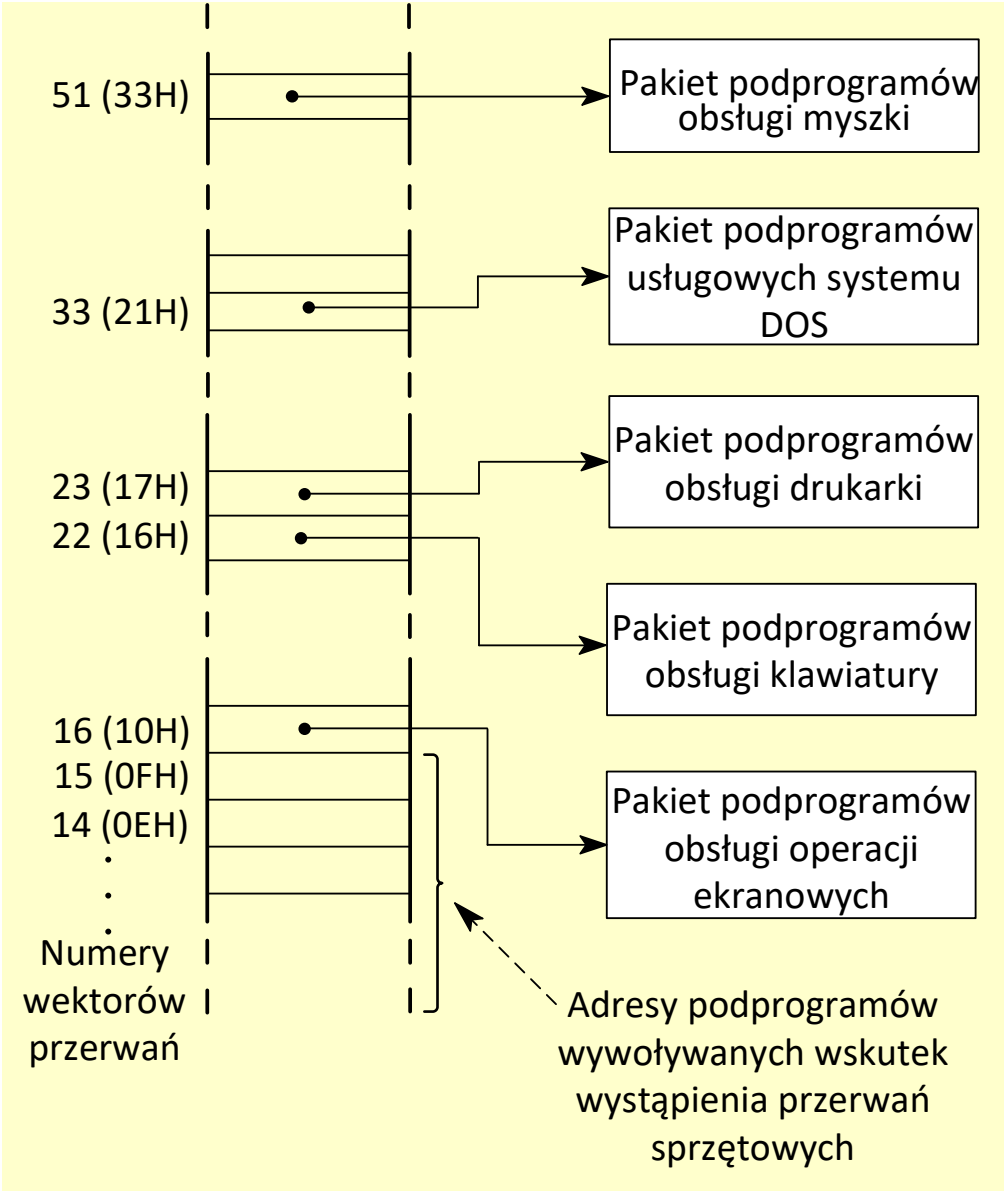


Funkcje systemowe BIOS (1)

- BIOS udostępnia funkcje systemowe potrzebne do uruchomienia właściwego systemu operacyjnego, a także do komunikacji z użytkownikiem (np. w celu konfiguracji komputera) — funkcje te wywoływane są za pomocą rozkazu **`INT`**, który można traktować jako odmianę rozkazu **`CALL`**.
- Operand rozkazu **`INT`** (np. **`INT 10H`** — operacje sterownika graficznego) stanowi indeks do tablicy wektorów przerwań, w której podane są adresy podprogramów realizujących funkcje systemowe.

POLITECHNIKA GDAŃSKA

Funkcje systemowe BIOS (2)



POLITECHNIKA GDAŃSKA

System UEFI

- Następcą systemu BIOS w komputerach jest stopniowo rozwijany interfejs UEFI (ang. Unified Extensible Firmware Interface).
- UEFI rozszerza możliwości konfiguracji sprzętu, a w przyszłości ma oferować wiele nowych usług, np. możliwość komunikacji przez internet w przypadku uszkodzenia głównego systemu operacyjnego. Niektóre planowane jego właściwości wzbudzają pewne kontrowersje, np. ograniczenia dostępu do nośników multimedialnych.

Pamięć fizyczna i wirtualna (1)

- Rozkazy (instrukcje) programu odczytujące dane z pamięci operacyjnej (czy też zapisujące wyniki) zawierają informacje o położeniu danej w pamięci, czyli zawierają adres danej.
- W wielu procesorach adres ten ma postać *adresu fizycznego*, czyli wskazuje jednoznacznie komórkę pamięci, gdzie znajduje się potrzebna dana.
- W trakcie operacji odczytu adres fizyczny kierowany do układów pamięci poprzez linie adresowe, a ślad za tym układy pamięci odczytują i odsyłają potrzebną daną.

Pamięć fizyczna i wirtualna (2)

- Takie proste adresowanie jest niepraktyczne w systemach wielozadaniowych.
- W rezultacie wieloletniego rozwoju architektury procesorów i systemów operacyjnych wyłoniła się koncepcja ***pamięci wirtualnej***, będącej pewną iluzją pamięci rzeczywistej (fizycznej).
- Pamięć operacyjna komputera w kształcie widzianym przez programistę nosi nazwę *pamięci liniowej (wirtualnej)*, a zbiór wszystkich możliwych adresów w pamięci wirtualnej nosi nazwę *liniowej* (*wirtualnej) przestrzeni adresowej*.

Pamięć fizyczna i wirtualna (3)

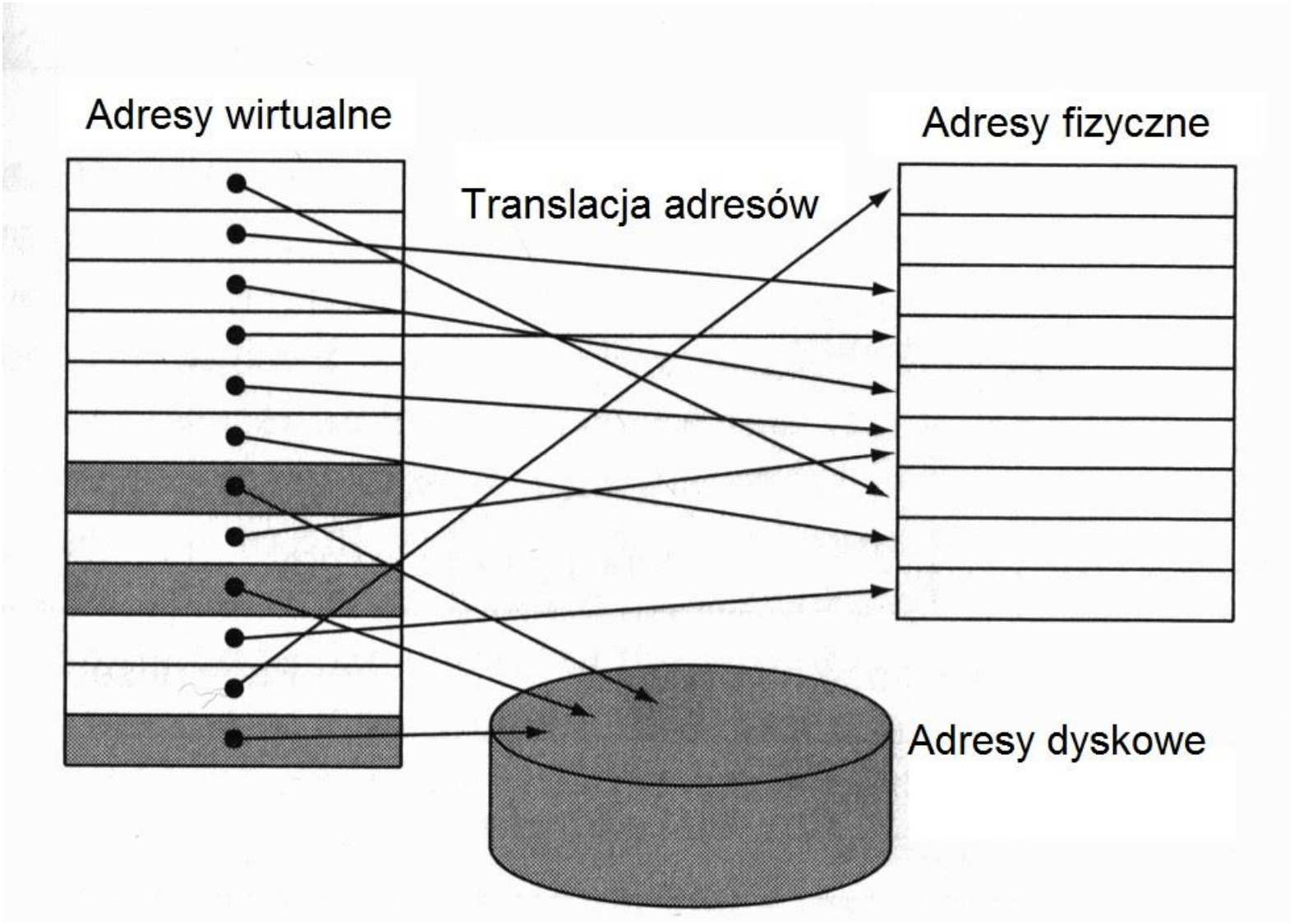
- Pamięć wirtualna jest implementowana za pomocą pamięci głównej (operacyjnej) i pamięci dyskowej.
- Zazwyczaj w pamięci operacyjnej przechowuje się tylko aktualnie używane fragmenty obszarów danych i rozkazów — aktualnie nieużywane obszary danych i rozkazów przechowywane są na dysku i miarę potrzeby przesyłane do pamięci głównej (operacyjnej).

Pamięć fizyczna i wirtualna (4)

- Programista, w trakcie tworzenia programu, może przyjąć, że w komputerze dostępna cała przestrzeń adresowa, np. 4 GB w postaci ciągłego obszaru pamięci, podczas gdy w rzeczywistości obszar ten może podzielony na fragmenty przechowywane częściowo w kilku obszarach pamięci fizycznej i częściowo na dysku.
- Programista nie musi też brać pod uwagę rozmiaru rzeczywiście zainstalowanej pamięci fizycznej, aczkolwiek zbyt mały rozmiar pamięci fizycznej zwiększa liczbę transmisji dyskowych, co spowalnia pracę programu.

POLITECHNIKA GDAŃSKA

Pamięć fizyczna i wirtualna (5)



Pamięć fizyczna i wirtualna (6)

- Transformacja adresów z liniowej przestrzeni wirtualnej na adresy fizyczne (rzeczywiście istniejących komórek pamięci) jest technicznie dość skomplikowana.
- Problemy te zostały jednak skutecznie rozwiązane, a związane z tym wydłużenie czasu wykonywania programu zwykle nie przekracza kilku procent.

Pamięć fizyczna i wirtualna (7)

- Przesyłanie danych i rozkazów z dysku do pamięci głównej i odwrotnie jest sterowane przez system operacyjny, silnie wspomagany przez specjalne funkcje procesora.
- Ww. przesyłanie jest niewidoczne z punktu widzenia zwykłego programu, w szczególności jeśli program żąda odczytania danych, które aktualnie znajdują się na dysku, to wykonywanie programu zostaje wstrzymane (program zostaje „uśpiony”) na czas transmisji z dysku do pamięci operacyjnej.

Pamięć fizyczna i wirtualna (8)

- Wprowadzenie pamięci wirtualnej przynosi szereg korzyści, m.in. w komputerze istnieje możliwość pseudo-jednoczesnego wykonywania wielu programów, który łączny rozmiar przekracza rozmiar zainstalowanej pamięci głównej (operacyjnej).

POLITECHNIKA GDAŃSKA

Funkcje systemowe a maszyny wirtualne

- Koncepcje wirtualizacji mogą się także odnosić do listy rozkazów procesora. Programista w niektórych kontekstach może nie zdawać sobie sprawy czy instrukcja występująca w programie stanowi pojedynczy rozkaz procesora, czy też wywołuje podprogram wchodzący w skład systemu operacyjnego (np. omawiany wcześniej rozkaz **`INT 21H`**).
- Funkcje systemu operacyjnego wraz z rozkazami procesora tworzą pewne środowisko, które częściowo jest realizowane sprzętowo, a częściowo przez funkcje systemu operacyjnego — powstaje w ten sposób pewna maszyna wirtualna, w której działa programista.

Dodawanie liczb całkowitych (1)

- Liczby całkowite kodowane są zazwyczaj jako liczby bez znaku (w naturalnym kodzie binarnym) albo jako liczby ze znakiem w kodzie U2
- Przypomnienie: wartość liczby binarnej kodowanej w systemie U2 określa wyrażenie (*m* oznacza liczbę bitów rejestru lub komórki pamięci):

$$w = -x_{m-1} \cdot 2^{m-1} + \sum_{i=0}^{m-2} x_i \cdot 2^i$$

Dodawanie liczb całkowitych (2)

- Przykłady kodowania liczb w kodzie U2:

–128	64	32	16	8	4	2	1
0	**0**	**1**	**1**	**1**	**0**	**0**	**1**

+32 +16 +8 +1 = 57

–128	64	32	16	8	4	2	1
1	**1**	**1**	**1**	**1**	**1**	**1**	**1**

–128 +64 +32 +16 +8 +4 +2 +1 = –1

Dodawanie liczb całkowitych (3)

- Dodawanie liczb w naturalnym kodzie binarnym jest wykonywane przez rozkaz ADD, np.

```
add   si, wynik
```

- W przypadku dodawania liczb ze znakiem w kodzie U2 można zauważyć, że jeśli pominąć skrajny lewy bit (bit znaku), to dodawanie może być wykonane tak samo jak dodawanie liczb bez znaku — niezależnie od tego czy liczba jest dodatnia czy ujemna, wagi przypisane poszczególnym bitom (z wyjątkiem skrajnego z lewej strony) mają wartości dodatnie.

Dodawanie liczb całkowitych (4)

- W trakcie dodawania skrajnych bitów trzeba uwzględnić, że mają one przypisane wagi będące liczbami ujemnymi (np. –128 w formacie 8-bitowym) oraz uwzględnić ewentualne przeniesienie powstałe w wyniku dodawania zawartości pozostałych bitów obu liczb
- Można tu rozpatrzyć kilka przypadków, których analiza wskazuje, że obliczenie sumy dwóch liczb w kodzie U2 wymaga dodania tych liczb traktowanych jako wartości w naturalnym kodzie binarnym, bez konieczności sprawdzania znaku.

Dodawanie liczb całkowitych (5)

- Tak więc algorytmy dodawania liczb kodowanych w systemie U2 są takie same jak dla liczb bez znaku (tj. liczb kodowanych w naturalnym kodzie binarnym).
- Z tego powodu rozkaz dodawania **`ADD`** może służyć zarówno do dodawania liczb bez znaku, jak i do dodawania liczb ze znakiem w kodzie U2; również rozkaz odejmowania SUB może wykonywać działania na obu typach liczb.

POLITECHNIKA GDAŃSKA

Przykład dodawania liczb całkowitych

- Za pomocą rozkazu **ADD** zostały dodane dwie analizowane wcześniej liczby binarne (+57 i –1) — liczba podana w nawiasie wpisywana jest do znacznika CF i stanowi wartość przeniesienia, które wystąpiło w trakcie dodawania najstarszych bitów obu liczb

```
        0 0 1 1 1 0 0 1
        1 1 1 1 1 1 1 1
        ---------------
(1)     0 0 1 1 1 0 0 0
```

POLITECHNIKA GDAŃSKA

Identyfikacja nadmiaru w trakcie dodawania (1)

- Czy uzyskany wynik dodawania jest poprawny?
- Udzielenie odpowiedzi na to pytanie jest możliwe tylko wówczas, gdy znamy typy dodawanych liczb.
- Jeśli przyjąć, że sumowanie wykonywane na liczbach w kodzie U2, tzn. wykonano obliczenie 57+(–1)=56, to wynik jest poprawny.
- Jeśli jednak sumowane liczby interpretowano jako liczby bez znaku, tzn. wykonano dodawanie 57+255=312, to wynik (=56) jest niepoprawny — w tym przypadku uzyskany wynik (312) nie mógł być zapisany na 8 bitach, wskutek czego wystąpiło przepełnienie.

POLITECHNIKA GDAŃSKA

Identyfikacja nadmiaru w trakcie dodawania (2)

- W przypadku dodawania liczb bez znaku ustawienie znacznika CF w stan 1 oznacza wystąpienie przeniesienia, co w podanym przykładzie interpretujemy jako nadmiar, a co z tym idzie niepoprawny wynik dodawania.
- Przeniesienie to nie ma jednak wpływu na poprawność wyniku, jeśli liczby interpretowane są jako liczby ze znakiem w kodzie U2.
- Na podstawie analizy podanego przykładu można stwierdzić, że identyfikacja nadmiaru powinna być prowadzona oddzielnie dla liczb ze znakiem (w kodzie U2) i dla liczb bez znaku.

Znaczniki CF i OF (1)

- W trakcie wykonywania rozkazu dodawania **`ADD`** procesor nie posiada żadnych informacji czy dodawane liczby zakodowane są jako liczby bez znaku czy też jako liczby w kodzie U2 — informacje takie posiada jedynie autor programu (czy kompilatora, który wytworzył te rozkazy).
- W tej sytuacji procesor udziela informacji o nadmiarze za pomocą dwóch znaczników:
  - CF (ang. carry flag) — dla przypadku, gdy dodawano liczby bez znaku;
  - OF (ang. overflow flag) — dla przypadku, gdy dodawano liczby w kodzie U2.

Znaczniki CF i OF (2)

- W dalszej kolejności, w zależności od typu dodawanych liczb programista testuje znacznik CF albo OF (do testowania można zastosować rozkazy **`jc`**, **`jnc`** albo **`jo`**, **`jno`**).
- Procesor ustawia znacznik CF na podstawie wartości przeniesienia, które powstaje przy dodawaniu najstarszych bitów obu liczb.
- Procesor ustawia znacznik OF na podstawie wartości wyrażenia $p_n \oplus p_{n-1}$, gdzie p_{n-1} i p_n oznaczają przeniesienia występujące podczas dodawania dwóch najbardziej znaczących bitów.

POLITECHNIKA GDAŃSKA

Odejmowanie liczb całkowitych (1)

- Zazwyczaj procesor wykonuje operację odejmowania poprzez zmianę znaku odjemnika, a następnie wykonuje dodawanie:

$$a - b = a + (-b)$$

- Jeden ze sposobów zmiany znaku liczby w kodzie U2 polega na zanegowaniu wszystkich bitów, a następnie dodaniu 1 do otrzymanej wartości.

POLITECHNIKA GDAŃSKA

Odejmowanie liczb całkowitych (2)

- Operacja odejmowania liczb ze znakiem (w kodzie U2) i liczb bez znaku, podobnie jak w przypadku dodawania, wykonywana jest przez ten sam rozkaz **`SUB`** — taką samą rolę jak przy dodawaniu pełnią znaczniki CF i OF, przy czym w operacji odejmowania znacznik CF reprezentuje pożyczkę.

POLITECHNIKA GDAŃSKA

Odejmowanie liczb całkowitych (3)

- Jednoargumentowy rozkaz **`NEG`** używany jest do obliczenia *liczby przeciwnej* (tj. zmiany znaku liczby w kodzie U2), np.

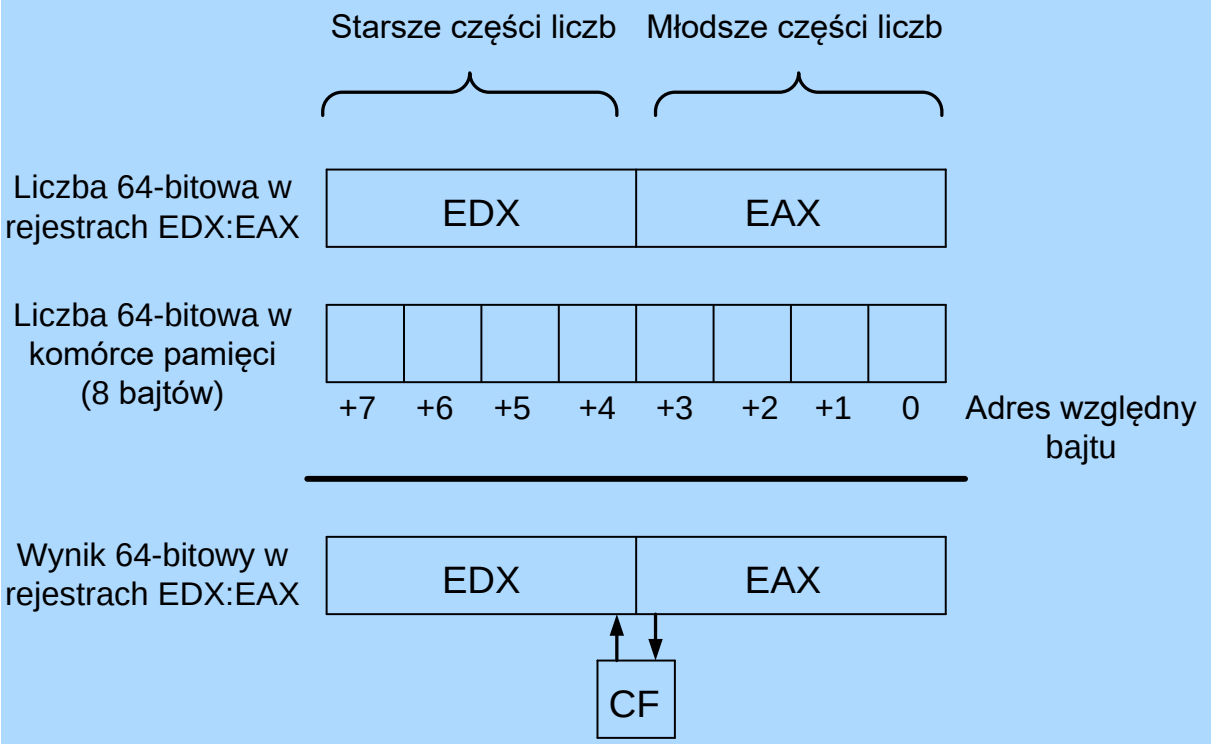
```
neg ecx
```

- Realizacja rozkazu **`NEG`** polega na odjęciu zawartości wskazanego obiektu od zera.
- Uwaga: należy odróżniać ten rozkaz od rozkazu negacji bitowej **`NOT`**.

POLITECHNIKA GDAŃSKA

Przykład dodawania w podwójnej precyzji (1)

- Dostępne są rozkazy, które wspomagają dodawanie i odejmowanie podwójnej precyzji



POLITECHNIKA GDAŃSKA

Przykład dodawania w podwójnej precyzji (2)

- W trakcie dodawania młodszych części obu liczb (rozkaz **add**) może wystąpić przeniesienie, które sygnalizowane jest przez ustawienie znacznika CF. Kolejny rozkaz (**adc**) uwzględnia to przeniesienie, dodając do wyniku drugiego dodawania liczbę zawartą w CF (0 lub 1).

```
; dodawanie młodszych części liczb
    add     eax, dword PTR liczba
; dodawanie starszych części liczb
    adc     edx, dword PTR liczba + 4
```

POLITECHNIKA GDAŃSKA

Mnożenie i dzielenie liczb całkowitych (1)

- Dla operacji mnożenia i dzielenia zdefiniowano oddzielne rozkazy dla liczb bez znaku i dla liczb ze znakiem w kodzie U2.
- Mnożenie liczb bez znaku wykonuje rozkaz **`MUL`**, a mnożenie liczb ze znakiem — rozkaz **`IMUL`**.
- Analogicznie w przypadku dzielenia: rozkaz **`DIV`** wykonuje dzielenie liczb bez znaku, rozkaz **`IDIV`** dzielenie liczb ze znakiem.

POLITECHNIKA GDAŃSKA

Mnożenie i dzielenie liczb całkowitych (2)

- Rozkazy dzielenia liczb całkowitych na podstawie dzielnej i dzielnika wyznaczają iloraz i resztę w postaci liczb całkowitych, przy czym spełniony jest związek:

$$\text{dzielna} = \text{iloraz} * \text{dzielnik} + \text{reszta}$$

przy czym wartość bezwzględna reszty jest zawsze mniejsza od wartości bezwzględnej dzielnika, a znak reszty jest taki sam jak znak dzielnej.

POLITECHNIKA GDAŃSKA

Nadmiar przy operacjach mnożenia i dzielenia (1)

- W typowych operacjach mnożenia (np. **`MUL`**, **`IMUL`**), mnożna i mnożnik zajmują jednakową liczbę bitów, a iloczyn umieszczany jest w rejestrze dwukrotnie dłuższym (zazwyczaj stanowiącym złożenie dwóch rejestrów, np. EDX:EAX) — w tej sytuacji nadmiar nigdy nie wystąpi.
- W typowych operacjach dzielenia (np. **`DIV`**, **`IDIV`**) dzielna jest dwukrotnie dłuższa od dzielnika, a iloraz i reszta umieszczane w rejestrach tych samych długości co dzielnik.

POLITECHNIKA GDAŃSKA

Nadmiar przy operacjach mnożenia i dzielenia (2)

- Błędne argumenty dzielenia mogą łatwo doprowadzić do powstania nadmiaru, który w przypadku rozkazu dzielenia prowadzi do wygenerowania *wyjątku procesora*, co powoduje przekazanie sterowania do systemu operacyjnego i zakończenie wykonywania programu.
- Wyjątki procesora omawiane będą w dalszej części wykładu.
- Podany niżej fragment programu wykonuje dzielenie 256/1, co przy nieodpowiednio dobranych długościach rejestrów prowadzi do wyjątku procesora (i zakończenia programu).

POLITECHNIKA GDAŃSKA

Nadmiar przy operacjach mnożenia i dzielenia (3)

```
mov     ax, 256
mov     bh, 1
div     bh          ; wyjątek procesora !!!
```

- Podany rozkaz dzielenia **DIV** dzieli liczbę zawartą w 16-bitowym rejestrze AX przez liczbę umieszczoną w 8-bitowym rejestrze BH.
- Reszta z dzielenia wpisywana jest do rejestru AH, a iloraz (tu: 256) do 8-bitowego rejestru AL — ponieważ w rejestrze AL może być zapisana co najwyżej liczba 255, więc powstaje nadmiar, a ślad za tym generowany jest wyjątek procesora.

Operacje na danych logicznych

- Lista rozkazów procesora zawiera zazwyczaj obszerną grupę rozkazów wykonujących działania na danych logicznych. Ponieważ dane logiczne mogą przyjmować tylko wartości *prawda* (true) albo *fałsz* (false), więc do zapisu danej logicznej wystarcza jeden bit.
- Omawiana grupa rozkazów jest szczególnie rozbudowana w procesorach przeznaczonych do zastosowań w systemach sterowania.

Operacje bitowe (1)

- W grupie rozkazów przeznaczonych do wykonywania działań na danych logicznych dostępne są rozkazy wykonujące działania na pojedynczych bitach, jak też na zespołach bitów. W szczególności możliwe jest wykonanie różnych operacji logicznych: negacji, sumy, iloczynu, sumy modulo dwa.

Operacje bitowe (2)

- W architekturze x86 dostępne są rozkazy wykonujące działania na wybranym bicie, przy czym przed wykonaniem operacji zawartość bitu jest kopiowana do znacznika CF
  - **`BT`** bit nie ulega zmianie
  - **`BTS`** wpisanie 1 do bitu
  - **`BTR`** wpisanie 0 do bitu
  - **`BTC`** zanegowanie zawartości bitu

Operacje bitowe (3)

- Każdy ww. rozkaz ma dwa operandy:

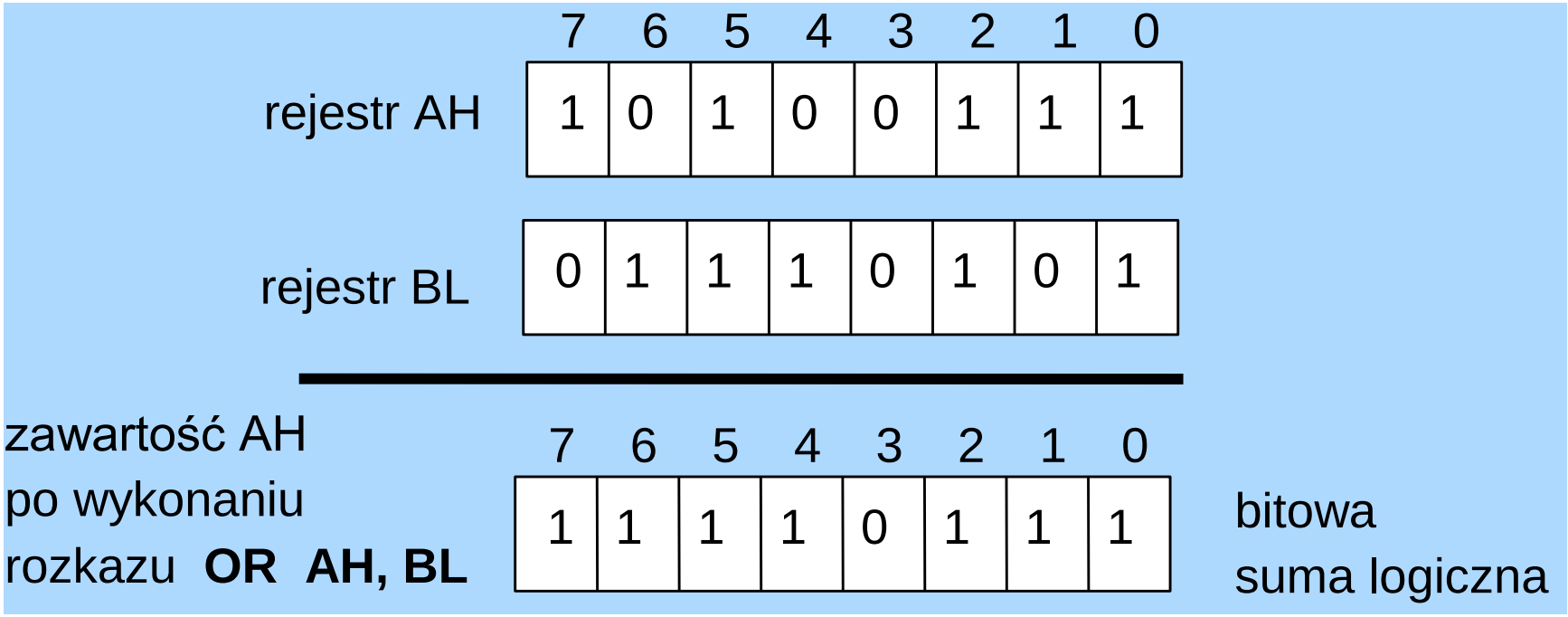
  pierwszy operand określa rejestr lub komórkę pamięci zawierającą modyfikowany bit;

  drugi operand, w postaci liczby lub rejestru, wskazuje numer bitu, na którym ma być wykonana operacja.
- Przykład: rozkaz `btc edi,29` powoduje zanegowanie bitu nr 29 w rejestrze EDI.
- Uwaga: operandami omawianej grupy rozkazów nie mogą być rejestry 8-bitowe.

Operacje bitowe (4)

- Bitowe operacje logiczne mogą być też wykonywane jednocześnie na zespołach bitów umieszczonych w rejestrach lub w komórkach pamięci.
- Rozkaz **`NOT`** jest jednoargumentowy — następuje zanegowanie wszystkich bitów w rejestrze.
- Rozkazy **`AND`** (iloczyn logiczny), **`OR`** (suma logiczna), **`XOR`** (suma modulo dwa) są dwuargumentowe i wykonują operacje logiczne na odpowiadających sobie bitach obu operandów (rejestru lub komórki pamięci) — rezultat wpisywany jest do operandu docelowego.

Operacje bitowe (5)

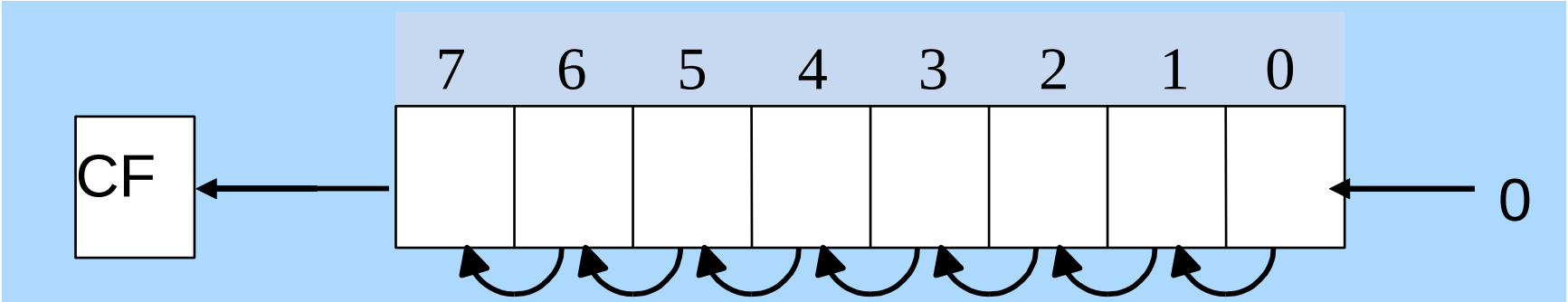


Operacje bitowe (6)

- Do testowania zawartości jednego lub kilku bitów stosowany jest rozkaz **`TEST`**, który wyznacza iloczyn logiczny, ale nigdzie nie wpisuje uzyskanego wyniku — ustawia natomiast znaczniki (rozkaz **`TEST`** działa analogicznie do rozkazu **`CMP`**).

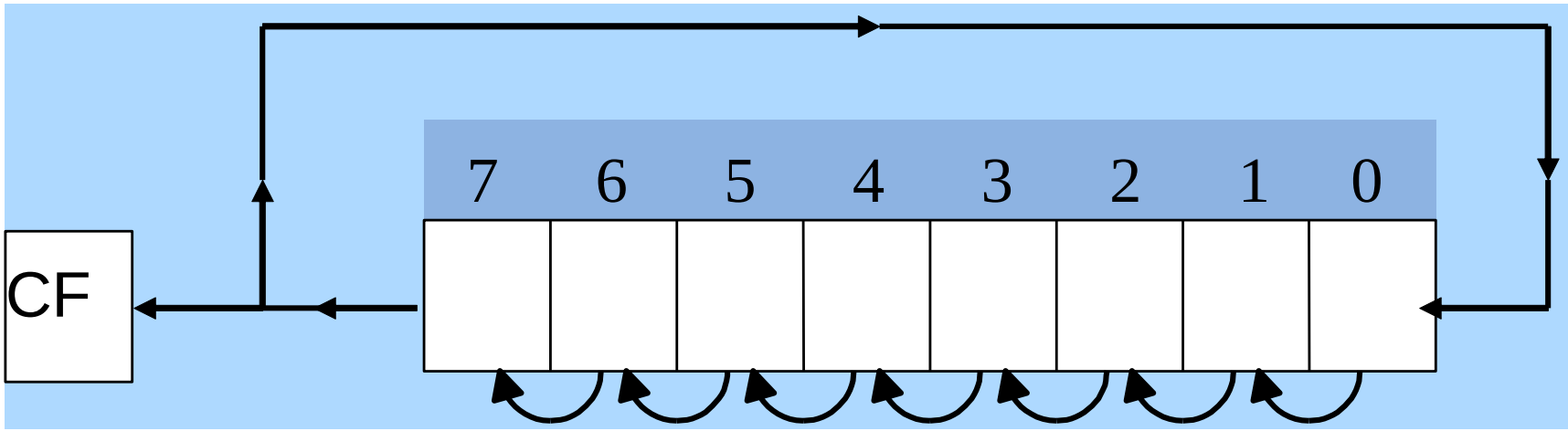
Przesunięcia logiczne

- Zawartość rejestru lub komórki pamięci traktowana jako ciąg bitów może być przesunięta w lewo lub prawo o podaną liczbę pozycji.
- Jeśli w trakcie przesuwania bity wychodzące z rejestru (lub z komórki pamięci) są tracone, to mówimy wówczas o *przesunięciu logicznym* (np. w x86 rozkazy **`SHL`**, **`SHR`**).



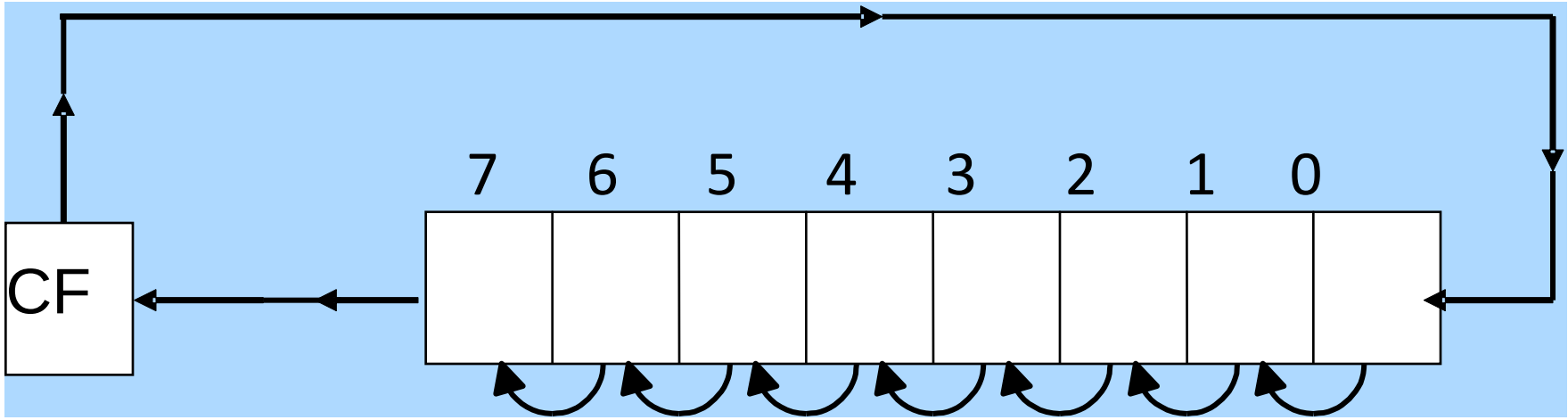
Przesunięcia cykliczne (1)

- Jeśli bity wychodzące są zawracane i wprowadzane z drugiej strony rejestru, to mówimy o *przesunięciu cyklicznym*, nazywanym także obrotem (np. w x86 rozkazy **`ROL`**, **`ROR`**).



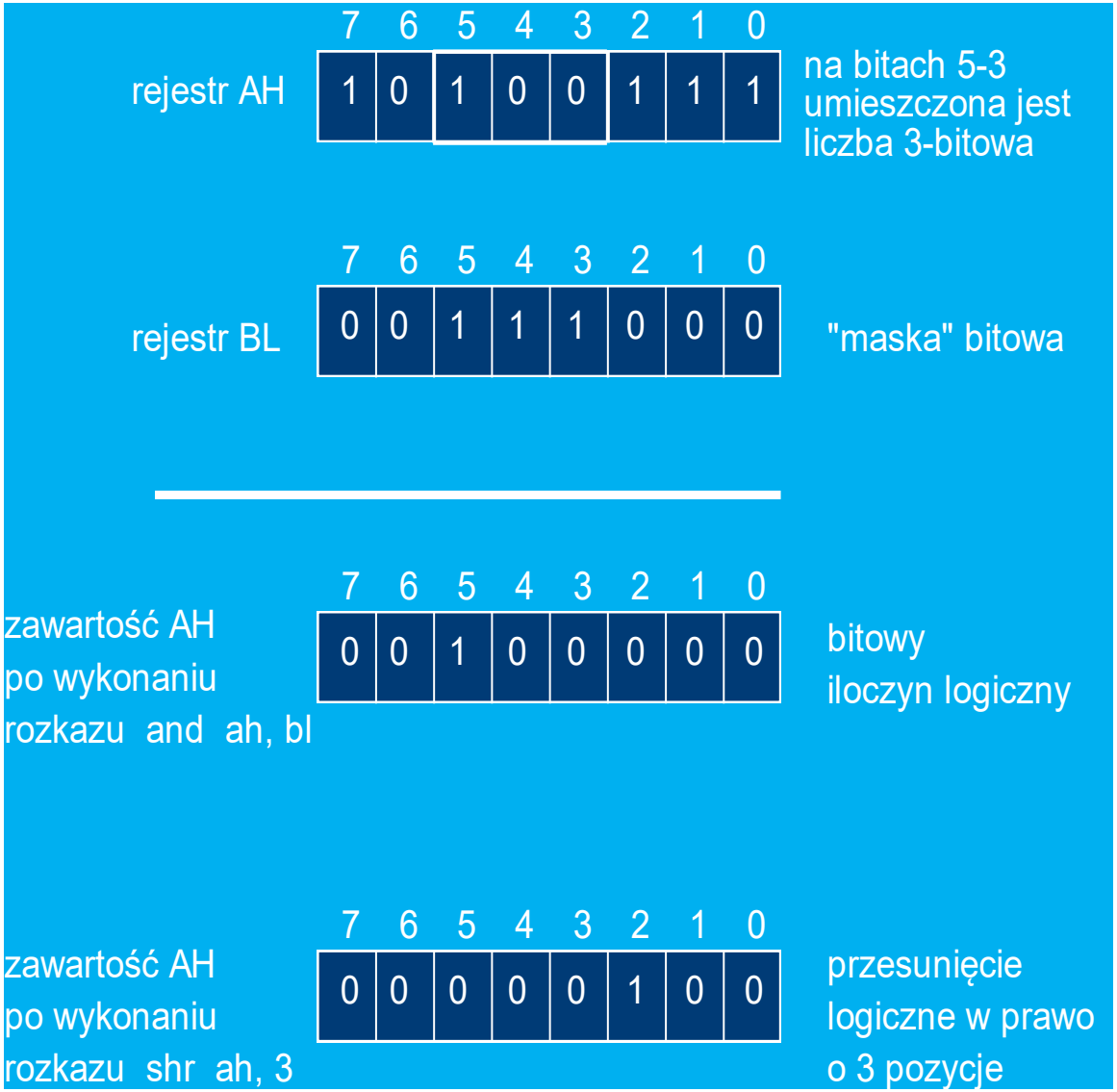
Przesunięcia cykliczne (2)

- Dostępne są także odmiany przesunięć cyklicznych: **`RCL`** i **`RCR`** — w rozkazach tych przyjmuje się, że na czas ich wykonywania znacznik CF staje się 9-, 17- lub 33-bitem rejestru — poniższy rysunek ilustruje przesunięcie w lewo (**`RCL`**).



POLITECHNIKA GDAŃSKA

Wyodrębnianie pól bitowych



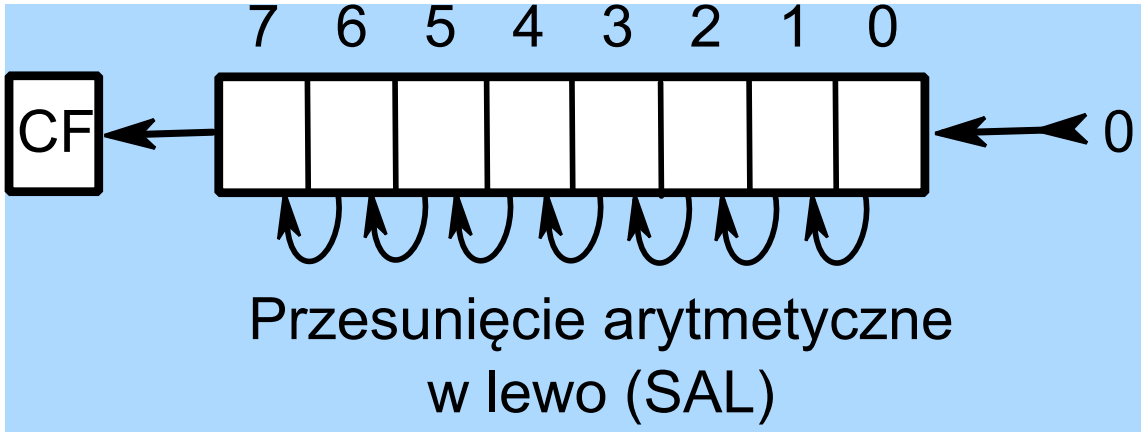
Przesunięcia arytmetyczne

- Rozkazy przesunięć mogą być zastosowane do mnożenia i dzielenia przez 2, 4, 8, ... (ogólnie: przez 2^k) — w przypadku dzielenia pozwala to znaczne skrócenie czasu operacji (np. 10 razy).
- Ze względu na specyfikę kodowania liczb ze znakiem w systemie U2 w architekturze x86 wprowadzono rozkazy przesunięć arytmetycznych, (np. **`SAL`**, **`SAR`**), które są bardzo podobne lub identyczne do przesunięć bitowych.

POLITECHNIKA GDAŃSKA

SAL – przesunięcie arytmetyczne w lewo (1)

- Przesunięcie arytmetyczne w lewo (**SAL**) działa dokładnie tak samo jak przesunięcie logiczne w lewo (**SHL**). Bity wychodzące z lewej strony wpisywane są znacznika CF, natomiast z prawej strony wprowadzane bity zerowe.



POLITECHNIKA GDAŃSKA

SAL – przesunięcie arytmetyczne w lewo (2)

- W przypadku przesuwania w lewo liczby bez znaku ewentualny nadmiar sygnalizowany jest przez ustawienie znacznika CF, natomiast nadmiar w przypadku liczb w kodzie U2 sygnalizuje znacznik OF.

POLITECHNIKA GDAŃSKA

SAL – Przesunięcie arytmetyczne w lewo (3)

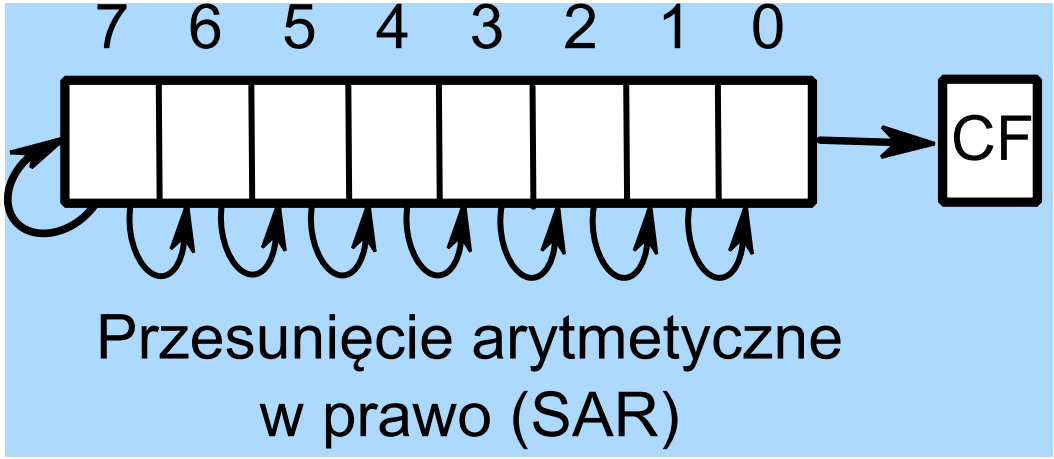
- Poniższa tabela ilustruje mnożenie przez 2 za pomocą rozkazu przesunięcia arytmetycznego w lewo o jedną pozycję (**`SAL BH, 1`**).

Rozkaz	Zawartość rejestru BH		
	przed wykonaniem rozkazu	po wykonaniu rozkazu	
SAL BH, 1	11111111 (–1)	11111110 (–2)	OF=0, CF=1
	11000000 (–64)	10000000 (–128)	OF=0, CF=1
	00111111 (+63)	01111110 (+126)	OF=0, CF=0
	10111111 (–65)	01111110 (+126)	OF=1,CF=1

POLITECHNIKA GDAŃSKA

SAR – przesunięcie arytmetyczne w prawo (1)

- Przesunięcie arytmetyczne w prawo (**SAR**) działa inaczej niż przesunięcie logiczne w prawo (**SHR**). Bity wychodzące z prawej strony wpisywane są znacznika CF, natomiast zawartość skrajnego bitu z lewej strony jest powielana.



POLITECHNIKA GDAŃSKA

SAR – przesunięcie arytmetyczne w prawo (2)

- Poniższa tabela ilustruje dzielenie przez 2 za pomocą rozkazu przesunięcia arytmetycznego w prawo o jedną pozycję (**`SAR BH, 1`**).

Rozkaz	Zawartość rejestru BH		
	przed wykonaniem rozkazu	po wykonaniu rozkazu	
SAR BH, 1	11111111 (–1)	11111111 (–1)	OF=0, CF=1
	11000000 (–64)	11100000 (–32)	OF=0, CF=0
	00111111 (+63)	00011111 (+31)	OF=0, CF=1
	10111111 (–65)	11011111 (–33)	OF=0, CF=1

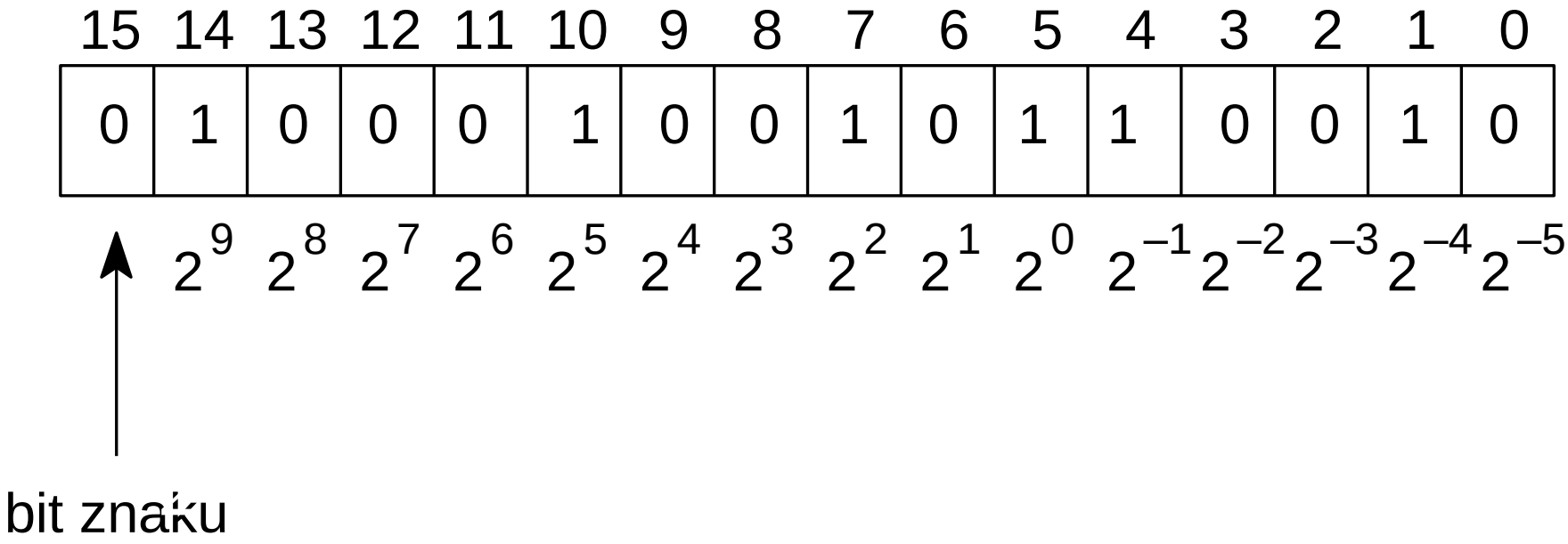
POLITECHNIKA GDAŃSKA

Kodowanie liczb mieszanych

- Opisy rozkazów arytmetycznych wykonywanych przez procesor podawane są zazwyczaj przy założeniu, że operacje wykonywane są na liczbach całkowitych.
- Działania mogą być wykonywane także na liczbach ułamkowych i mieszanych — wymaga to przekształcenia algorytmu w taki sposób, by operacje na ułamkach zostały zastąpione przez operacje na liczbach całkowitych.

POLITECHNIKA GDAŃSKA

Przykład kodowania liczby mieszanej



POLITECHNIKA GDAŃSKA

Formaty liczb mieszanych

- Nie ma standardowych formatów liczb mieszanych — przypisanie wag poszczególnym bitom, a więc ustalenie położenia kropki rozdzielającej część całkowitą i ułamkową liczby zależy od decyzji programisty.
- Ustalenie formatu wynika z zakresu zmienności danych i wyników pośrednich, jak również z wymagań dotyczących dokładności obliczeń.
- Dość często spotyka się format *śródprzecinkowy*, w którym połowa bitów ma przypisane wagi o wartościach całkowitych, a pozostałe – ułamkowe

POLITECHNIKA GDAŃSKA

Ułamki dziesiętne a ułamki binarne (1)

- Zazwyczaj dane przekazywane są do programu w postaci liczb dziesiętnych.
- Reprezentacja binarna części całkowitej liczby jest zawsze dokładna.
- Reprezentacja ułamka dziesiętnego w postaci binarnej stanowi na ogół przybliżenie — tylko niektóre liczby, jak np. 0.625 (tj. 0.5 + 0.125) mają dokładną reprezentację binarną.

POLITECHNIKA GDAŃSKA

Ułamki dziesiętne a ułamki binarne (2)

- Rozwinięcie binarne liczby 0.3 ma postać 0.010 011 001 100 110 011 001 100 110 011 001 100 ... = 0.0(1001)
- W poniższym programie

```
int  i;  float  a = 0, b = 0;
for (i=0; i < 100; i++)
          a = a + 0.3; b = b + 0.25;
printf("\na = %f  b = %f", a, b);
```

zostały wyświetlone wyniki

a = 29.999971 b = 25.000000

POLITECHNIKA GDAŃSKA

Obliczenia na liczbach bardzo dużych i bardzo małych (1)

- Przykład: obliczenie stałej czasowej obwodu RC

$$R = 4.7\ \mathrm{M\Omega}, C = 68\ \mathrm{pF}$$

$$RC = 4.7 \cdot 10^{6} \cdot 68 \cdot 10^{-12} = 319.6 \cdot 10^{-6}$$

- Wartość R (= 4 700 000) w postaci 24-bitowej liczby binarnej:

R=01000111 10110111 01100000

- Wartość pojemności C (= 0.000 000 000 068) w postaci binarnej ma rozwinięcie nieskończone okresowe.

POLITECHNIKA GDAŃSKA

Obliczenia na liczbach bardzo dużych i bardzo małych (2)

- Przyjmujemy, że część ułamkowa liczby będzie zajmowała 40 bitów.
- Trzy najbliższe 40-bitowe liczby binarne do wartości 0.000 000 000 068 mają postać:

0.00000000 00000000 00000000 00000000 01001010

(= 0.000 000 000 067 302 607 931 196 689 605 712 890 625)

0.00000000 00000000 00000000 00000000 01001011

(= 0.000 000 000 068 212 102 632 969 617 843 627 929 687 5)

0.00000000 00000000 00000000 00000000 01001100

(= 0.000 000 000 069 121 597 334 742 546 081 542 968 75

- liczba 0.00000000 00000000 00000000 00000000 01001011 stanowi więc najlepsze przybliżenie 40-bitowe

POLITECHNIKA GDAŃSKA

Obliczenia na liczbach bardzo dużych i bardzo małych (3)

- Zatem obliczenie wartości RC wymaga przyjęcia formatu, w którym część całkowita liczby zajmować będzie 24 bity, a część ułamkowa 40 bitów – łącznie 64 bity, czyli 8 bajtów.
- Przy podanym formacie liczby R i C będą zajmować w pamięci po 8 bajtów, przy czym w reprezentacji binarnej liczby R część ułamkowa będzie zawierała same zera (5 bajtów), a w reprezentacji liczby C część całkowita liczby będzie zawierała same zera (3 bajty), a także początkowe 4 bajty części ułamkowej będą wypełnione zerami.

POLITECHNIKA GDAŃSKA

Liczby zmiennoprzecinkowe (1)

- Podany poprzednio sposób kodowania liczb jest szczególnie nieefektywny jeśli w obliczeniach występują liczby bardzo duże i bardzo małe, co jest charakterystyczne dla obliczeń naukowo-technicznych.
- Radykalną poprawę w tym zakresie przynosi kodowanie zmiennoprzecinkowe (zmienno-pozycyjne), w którym skupia się uwagę na cyfrach znaczących liczby, rejestrując jednocześnie położenie kropki rozdzielającej część całkowitą i ułamkową.

Liczby zmiennoprzecinkowe (2)

- Weźmy pod uwagę omawianą poprzednio liczbę binarną

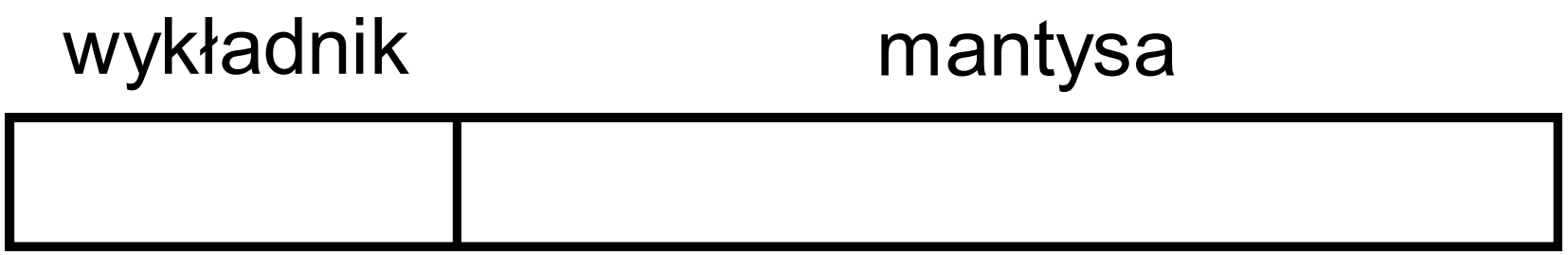
  0.00000000 00000000 00000000 00000000 01001011

- Spróbujmy przesunąć kropkę w prawo, tak by kropka znalazła się po prawej stronie najbardziej znaczącej cyfry 1 — otrzymamy: 1.001011
- W tym przypadku kropkę przesunęliśmy o 34 pozycje w prawo.
- Późniejsze odtworzenie oryginalnej liczby będzie możliwe, jeśli obok wartości liczby (z przesuniętą kropką) zapiszemy także liczbę przesunięć (tu: 34).

Liczby zmiennoprzecinkowe (3)

Omawiany sposób kodowania wymaga więc przechowywania liczby w postaci dwóch elementów:

- **mantysy**, stanowiącej wartość liczby znormalizowaną do przedziału (–2,–1> lub <1,2)
- **wykładnika**, opisującego liczbę przesunięć kropki w prawo lub w lewo



Liczby zmiennoprzecinkowe (4)

$$\text{mantysa} \cdot 2^{\text{wykladnik}}$$

- Powyższe wyrażenie określa wartość liczby zmiennoprzecinkowej różnej od 0.
- Liczba 0 kodowana jest jako wartość specjalna (pola mantysy i wykładnika wypełnione są zerami).
- Dla liczb różnych od zera wymaga się, by mantysa spełniała *warunek normalizacji*:

$$1 \leq |\ \text{mantysa}\ | < 2$$

Liczby zmiennoprzecinkowe (5)

- Tak więc liczba omawiana liczba przykładowa

0.00000000 00000000 00000000 00000000 01001011

w postaci zmiennoprzecinkowej może być zakodowana jako:

wykładnik	mantysa
–34 (binarnie)	1.001011

POLITECHNIKA GDAŃSKA

Standardy kodowania liczb zmiennoprzecinkowych

- Podane zasady kodowania zostały sformalizowane w postaci standardu znanego jako norma IEEE 754.
- Pierwsza wersja standardu została opublikowana w roku 1985, a najnowsza aktualizacja pochodzi z roku 2008.
- W standardzie zdefiniowano formaty liczb zmiennoprzecinkowych, określono także reguły zaokrąglania, postępowanie w przypadków błędów w obliczeniach, itd.

POLITECHNIKA GDAŃSKA

Formaty pojedynczej i podwójnej precyzji (1)

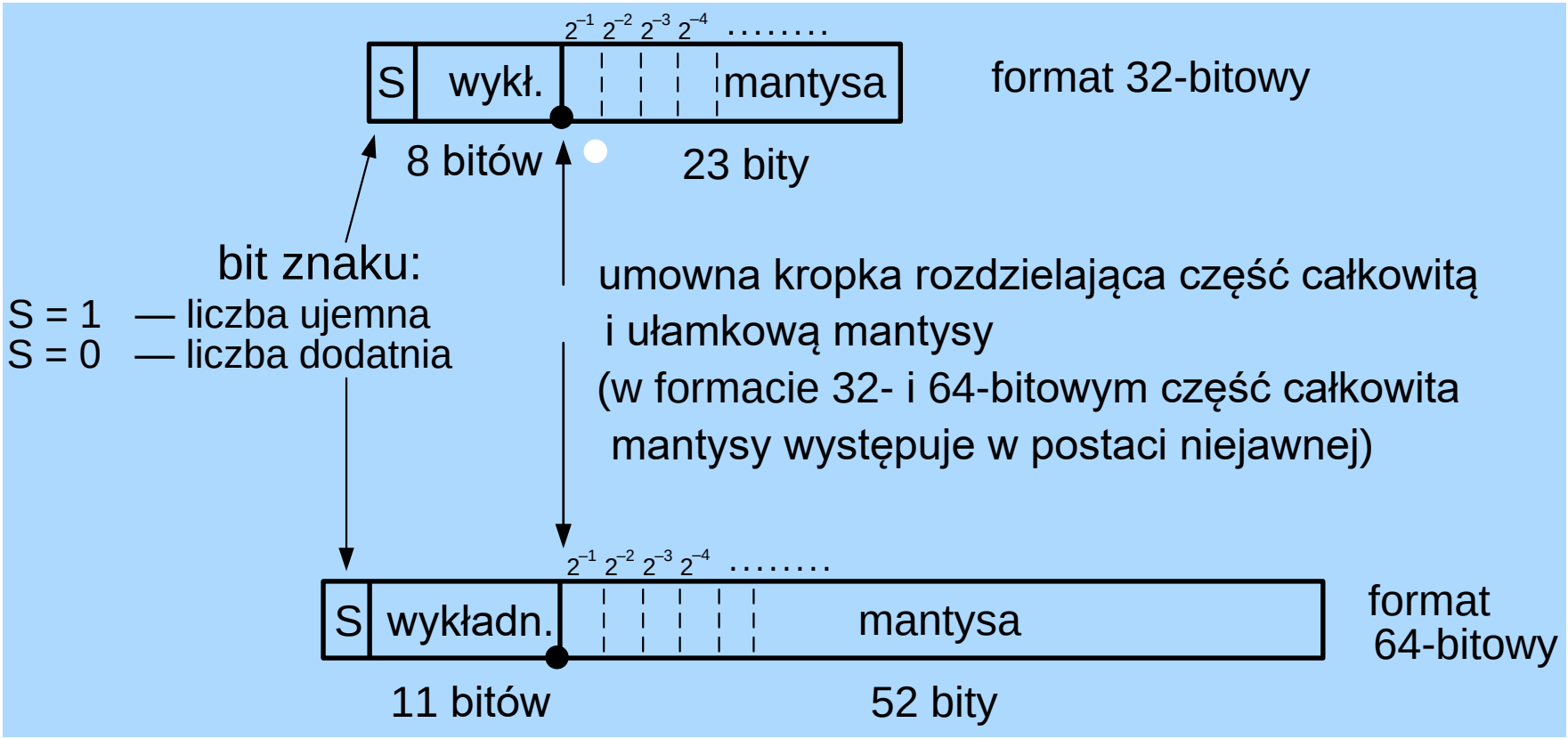
- Spośród formatów zdefiniowanych w standardzie IEEE 754 najczęściej używane są:

  Format 32-bitowy pojedynczej precyzji oznaczany jako *binary32*, w języku C formatowi temu odpowiada typ **float**

  Format 64-bitowy podwójnej precyzji oznaczany jako *binary64*, w języku C formatowi temu odpowiada typ **double**

POLITECHNIKA GDAŃSKA

Formaty pojedynczej i podwójnej precyzji (2)



POLITECHNIKA GDAŃSKA

Formaty pojedynczej i podwójnej precyzji (3)

- Znak mantysy określa skrajny bit z lewej strony.
- W formatach 32- i 64-bitowych pomija się kodowanie części całkowitej mantysy (ale uwzględnia w obliczeniach!) – z warunku normalizacji wynika bowiem, że bit ten jest zawsze równy 1 (dla liczb różnych od 0)

$$1 \leq |\text{mantysa}| < 2$$

POLITECHNIKA GDAŃSKA

Formaty pojedynczej i podwójnej precyzji (4)

- W standardzie nie przewidziano bitu znaku wykładnika — zamiast tego w *polu wykładnika* zapisuje się wartość wykładnika powiększoną o stałą wartość (127 dla formatu 32-bitowego, 1023 dla formatu 64-bitowego.

- W rezultacie liczba zapisywana w polu wykładnika jest zawsze dodatnia i bit znaku nie jest potrzebny.

POLITECHNIKA GDAŃSKA

Przykład kodowania liczby 12.25 w formacie 32-bitowym (1)

- Liczbę 12.25 przedstawiamy w postaci iloczynu

$$m \cdot 2^k$$

- Wykładnik potęgi k musi być tak dobrany, by spełniony był warunek normalizacji mantysy

$$1 \leq | m | < 2$$

- czyli

$$1 \leq \frac{12.25}{2^k} < 2$$

Przykład kodowania liczby 12.25 w formacie 32-bitowym (2)

- Łatwo zauważyć, że warunek normalizacji jest spełniony, gdy k = 3. Zatem

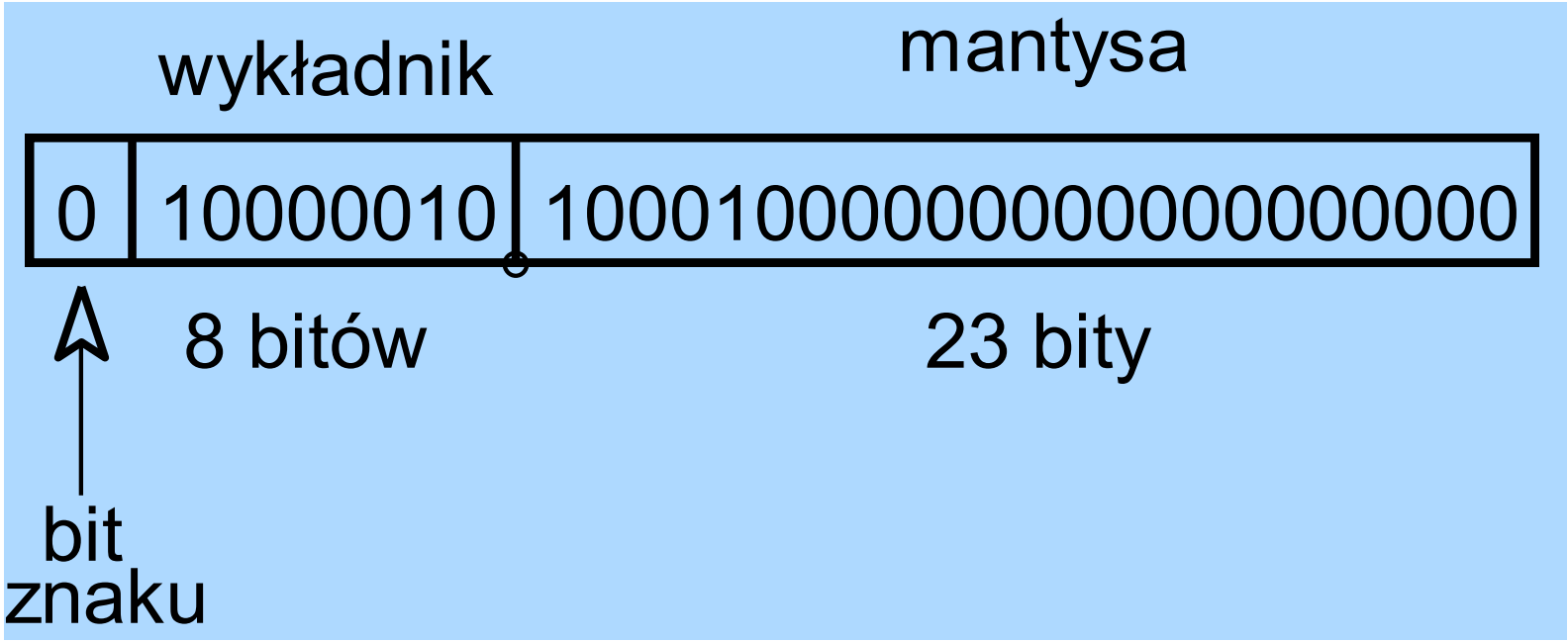
$$12.25 = \frac{12.25}{2^3} \cdot 2^3 = 1.53125 \cdot 2^3 = (1.10001)_2 \cdot 2^{130-127} =$$

$$= (1.10001)_2 \cdot 2^{(10000010)_2 - 127}$$

- Ponieważ część całkowita mantysy nie jest kodowana, więc w polu mantysy zostanie wpisana liczba $(0.10001)_2$, zaś w polu wykładnika (po przesunięciu o 127) liczba $(10000010)_2$. Ostatecznie otrzymamy

POLITECHNIKA GDAŃSKA

Przykład kodowania liczby 12.25 w formacie 32-bitowym (3)

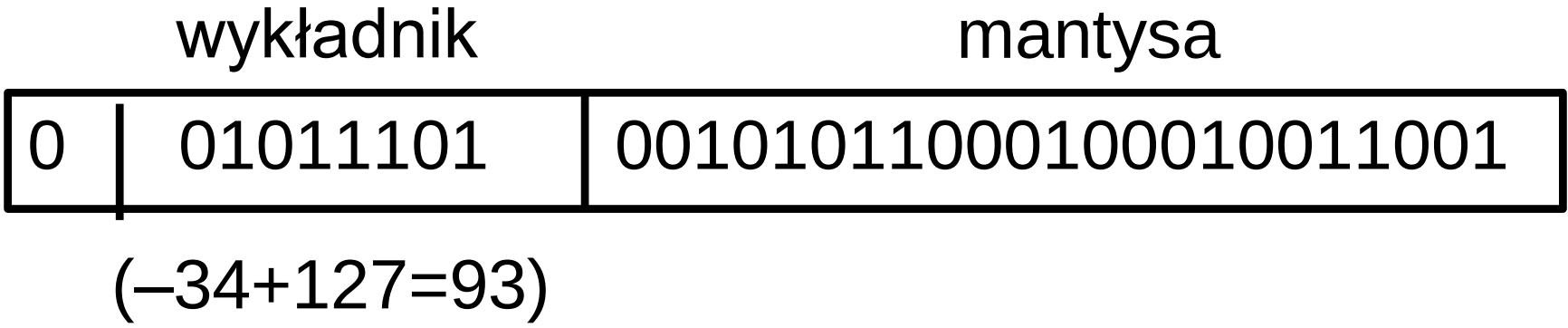


POLITECHNIKA GDAŃSKA

Przykład kodowania liczby $68*10^{-12}$

- Poniżej podano liczbę $68*10^{-12}$ zakodowaną w formacie 32-bitowym (float).
- Dokładna wartość zakodowanej liczby w zapisie dziesiętnym wynosi:

0.000000000068000001463008885593808372505009174346 92



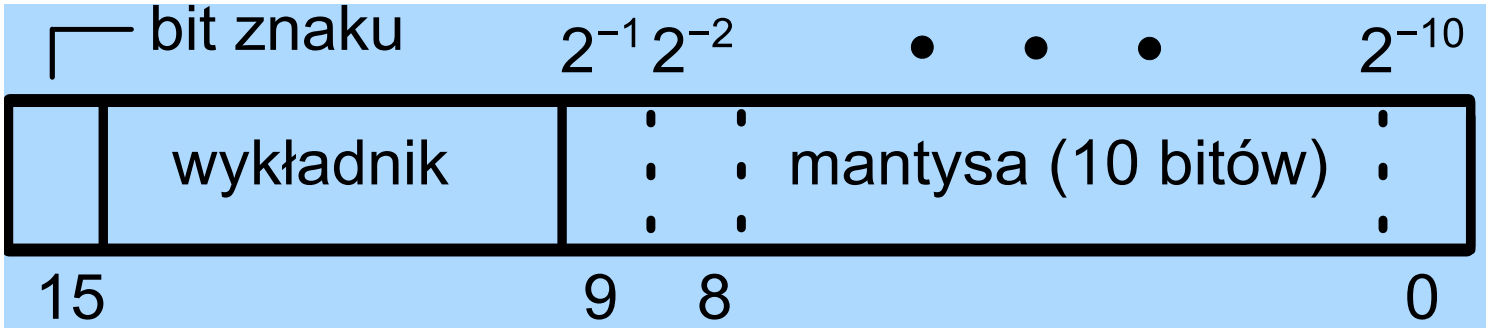
(–34+127=93)

POLITECHNIKA GDAŃSKA

Inne formaty w standardzie IEEE 754 (1)

- Oprócz omówionych, standard IEEE 754 definiuje również:

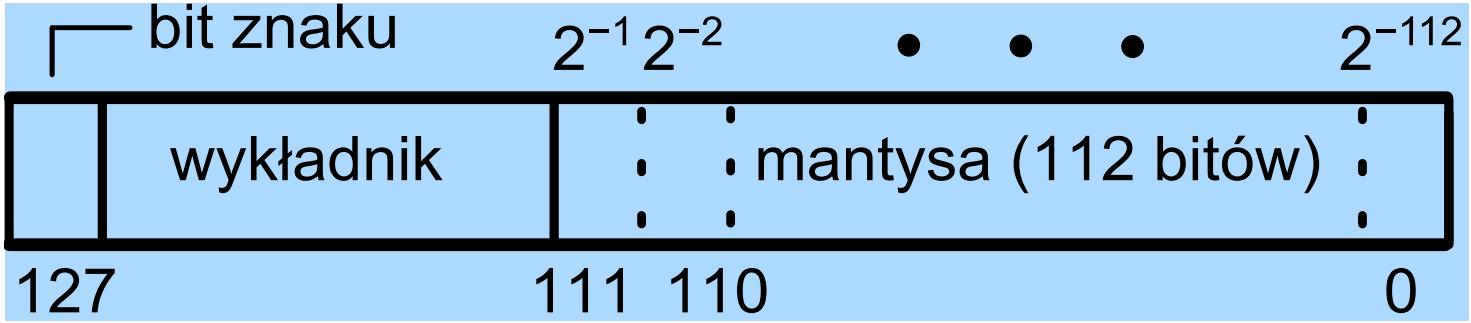
  Format 16-bitowy (ang. half precision) oznaczany jako *binary16* — mantysa zapisywana jest na 10 (11) bitach, a wykładnik na 5 bitach; format ten stosowany w grafice komputerowej.



POLITECHNIKA GDAŃSKA

Inne formaty w standardzie IEEE 754 (2)

Format 128-bitowy (ang. quadruple precision) oznaczany jako *binary128* — mantysa zapisywana jest na 112 (113) bitach, a wykładnik na 15 bitach; format ten używany jest w obliczeniach wymagających bardzo dużej dokładności.



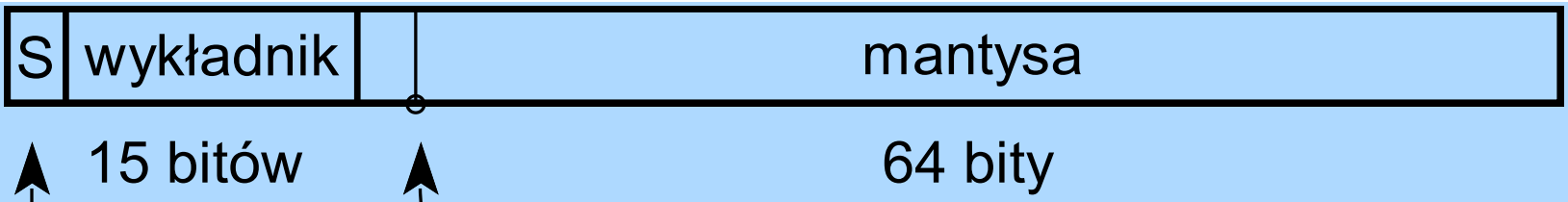
POLITECHNIKA GDAŃSKA

Inne formaty w standardzie IEEE 754 (3)

- Formaty dziesiętne (decimal32, decimal64, decimal128) przeznaczone są do obliczeń finansowych, w których zaokrąglenia muszą być wykonywane wg tych samych reguł co w obliczeniach wykonywanych na papierze.

Format 80-bitowy

- Koprocesor arytmetyczny stowarzyszony z głównym procesorem x86 wykonuje obliczenia na liczbach w formacie 80-bitowym pośrednim (ang. temporary real)



umowna kropka rozdzielająca część całkowitą i ułamkową mantysy (w formacie 80-bitowym część całkowita mantysy występuje w postaci jawnej)

bit znaku:
S = 0 — liczba dodatnia
S = 1 — liczba ujemna

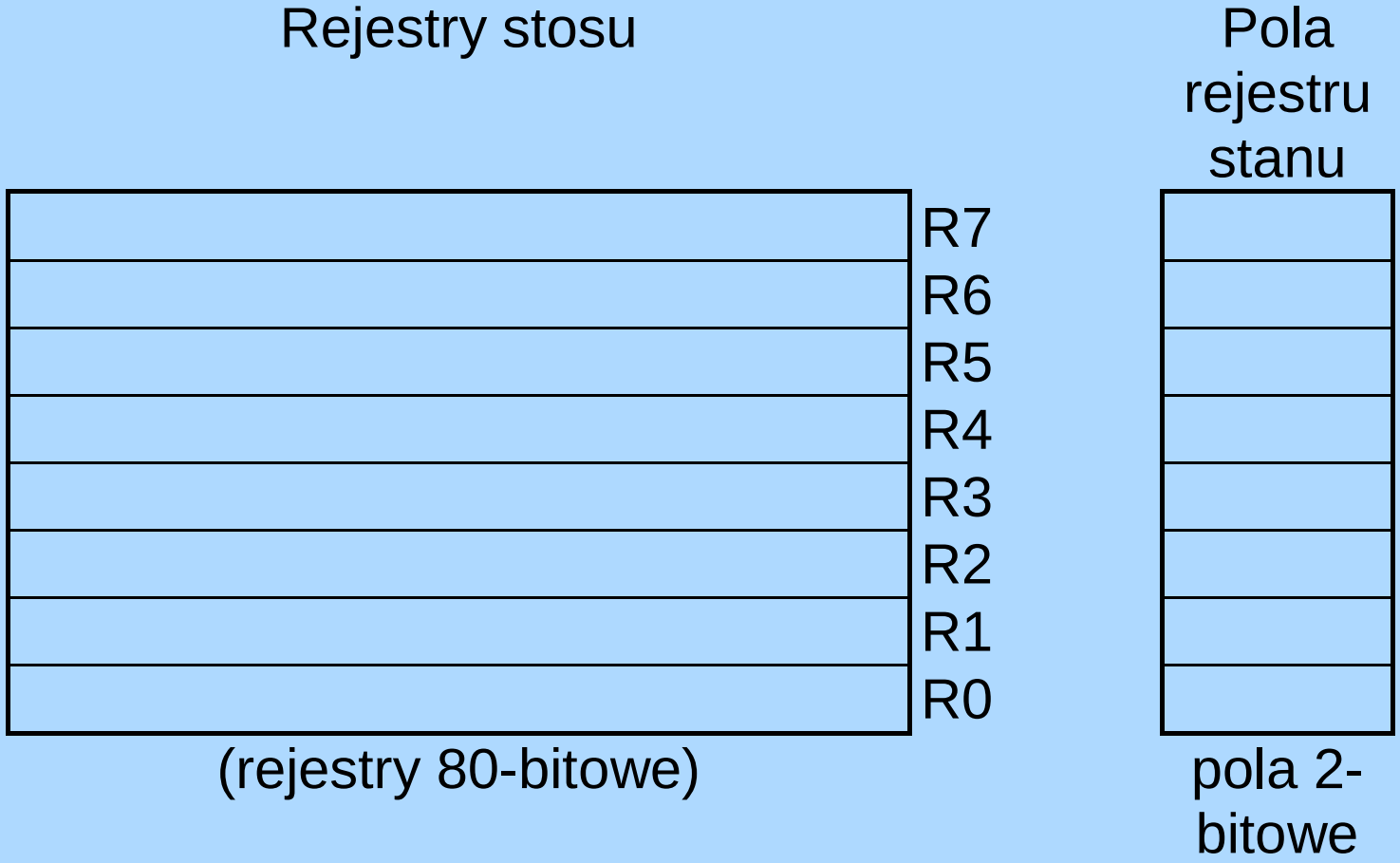
POLITECHNIKA GDAŃSKA

Zasady wykonywania obliczeń przez koprocesor arytmetyczny (1)

- Koprocesor arytmetyczny stanowi odrębny procesor, współdziałający z procesorem głównym i znajdujący się w tej samej obudowie.
- W przeszłości koprocesor stanowił oddzielny układ scalony instalowany na płycie głównej komputera.
- Liczby, na których wykonywane są obliczenia, składowane są w 8 rejestrach 80-bitowych tworzących *stos rejestrów koprocesora arytmetycznego* .

POLITECHNIKA GDAŃSKA

Zasady wykonywania obliczeń przez koprocesor arytmetyczny (2)



POLITECHNIKA GDAŃSKA

Zasady wykonywania obliczeń przez koprocesor arytmetyczny (3)

- Z każdym rejestrem związane jest 2-bitowe *pole stanu rejestru* (nazywane także *polem znaczeń*); wszystkie pola stanu tworzą 16-bitowy rejestr zwany *rejestrem stanu stosu koprocesora*; interpretacja pola stanu jest następująca:

  0 – rejestr zawiera liczbę różną od zera,

  1 – rejestr zawiera zero,

  2 – rejestr zawiera błędny rezultat,

  3 – rejestr jest pusty.

- Zawartość rejestru stanu stosu koprocesora można skopiować do pamięci za pomocą rozkazu **`FSTENV`** lub **`FSAVE`**.

POLITECHNIKA GDAŃSKA

Zasady wykonywania obliczeń przez koprocesor arytmetyczny (4)

- Rozkazy koprocesora adresują rejestry stosu nie bezpośrednio, ale względem wierzchołka stosu.
- W kodzie asemblerowym rejestr znajdujący się na wierzchołku stosu oznaczany jest ST(0) lub ST, a dalsze ST(1), ST(2),..., ST(7)
- Ze względu na specyficzny sposób adresowania koprocesor arytmetyczny zaliczany jest do procesorów o *architekturze stosowej*.
- Mechanizmy stosu rejestrów koprocesora są analogiczne do mechanizmów stosu w procesorze (m.in. stos rośnie w kierunku malejących numerów rejestrów).

POLITECHNIKA GDAŃSKA

Lista rozkazów koprocesora arytmetycznego (1)

- Lista rozkazów koprocesora arytmetycznego zawiera rozkazy wykonujące działania na liczbach zmiennoprzecinkowych, w tym rozkazy przesłania, działania arytmetyczne, obliczanie pierwiastka kwadratowego, funkcji trygonometrycznych (sin, cos, tg, arc tg), wykładniczych i logarytmicznych.
- Wszystkie mnemoniki rozkazów koprocesora zaczynają się od litery F (ang. float).

POLITECHNIKA GDAŃSKA

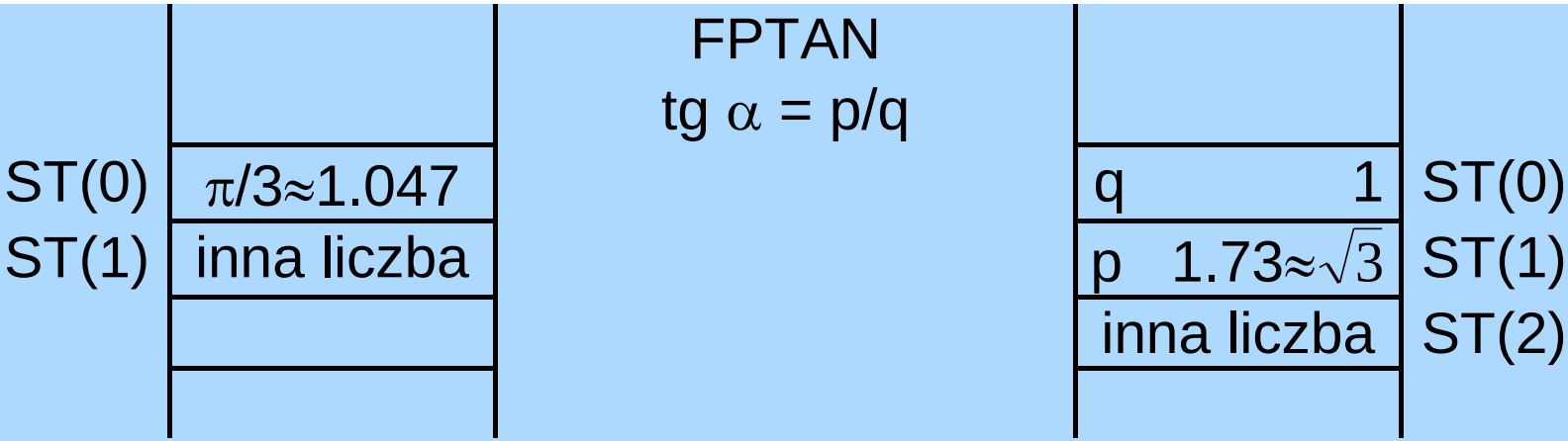
Lista rozkazów koprocesora arytmetycznego (2)

- Prawie zawsze jeden z argumentów wykonywanej operacji znajduje się na wierzchołku stosu ST(0), nawet jeśli nie jest określony w postaci jawnej.
- Przykładowo, rozkaz mnożenia **`fmul vc`** powoduje pomnożenie liczby znajdującej się na wierzchołku stosu przez wartość zmiennej vc, a wynik wpisywany w miejsce dotychczasowej wartości na wierzchołku stosu (wskaźnik stosu koprocesora nie zmienia się).

POLITECHNIKA GDAŃSKA

Przykład: obliczanie wartości tg α

- Przykładowo, do obliczenia wartości funkcji tangens używa się rozkazu FPTAN, który oblicza wartość funkcji dla argumentu (w radianach) podanego na wierzchołku stosu koprocesora.
- W wyniku podaje dwie liczby — ich iloraz stanowi wartość funkcji (bezpośrednio po rozkazie FPTAN należy wykonać rozkaz FDIV (bez operandów)).



POLITECHNIKA GDAŃSKA

Przykład: obliczanie wartości wyróżnika trójmianu kwadratowego (1)

- Zakładamy, że wartości współczynników trójmianu *a*, *b*, *c* przechowywane są w zmiennych, odpowiednio **va**, **vb**, **vc**

```
va          dq      1.0
vb          dq      5.5
vc          dq      -15.0
liczba4     dq      4.0
```

POLITECHNIKA GDAŃSKA

Przykład: obliczanie wartości wyróżnika trójmianu kwadratowego (2)

```
finit    ; inicjalizacja koprocesora

; załadowanie wartości 'b' na wierzchołek stosu
; koprocesora
fld      vb
```

b	st(0)

```
fmul     st(0), st(0) ; obliczenie b^2
```

b^2	st(0)

POLITECHNIKA GDAŃSKA

Przykład: obliczanie wartości wyróżnika trójmianu kwadratowego (3)

; załadowanie wartości 'a' na wierzchołek stosu

```
fld     va
```

b^2	st(1)
a	st(0)

```
fmul    vc          ; mnożenie a * c
```

b^2	st(1)
ac	st(0)

POLITECHNIKA GDAŃSKA

Przykład: obliczanie wartości wyróżnika trójmianu kwadratowego (4)

```
fmul     liczba4        ; obliczenie 4 * (a * c)
```

b^2	st(1)
4ac	st(0)

```
fsubp    st(1),st(0)    ; obliczenie b2 – 4ac
```

b^2-4ac	st(1)
4ac	st(0)

; litera **p** (pop) w mnemoniku fsub**p** oznacza
; polecenie usunięcia liczby z wierzchołka stosu
; koprocesora

b^2-4ac	st(0)

POLITECHNIKA GDAŃSKA

Przykład: obliczanie wartości wyróżnika trójmianu kwadratowego (5)

; obliczenie pierwiastka z liczby na wierzchołku

; stosu koprocesora

```
fsqrt
```

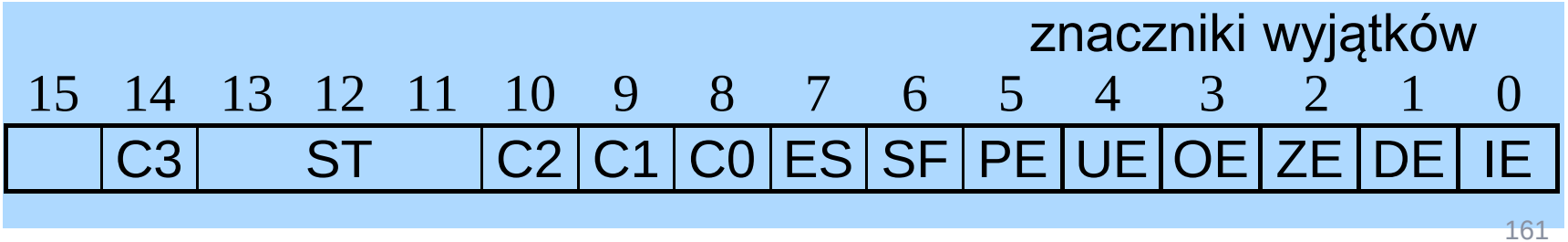
$\sqrt{b^2 - 4ac}$ **st(0)**

Rejestr stanu koprocesora

- *Rejestr stanu koprocesora* zawiera różne pola informujące o przebiegu obliczeń, wystąpieniu specjalnych warunków, określa położenie wierzchołka stosu i inne informacje.
- Zawartość rejestru stanu koprocesora może być przesłana do rejestru AX za pomocą rozkazu

```
fstsw  ax
```

- W języku C rejestr ten można odczytać za pomocą funkcji **_status87**



POLITECHNIKA GDAŃSKA

Rejestr sterujący koprocesora (1)

- *Rejestr sterujący koprocesora* zawiera pola, które określają sposób działanie koprocesora.
- Zawartość tego rejestru zmienia się wyłącznie wskutek jawnego załadowania go przez program, np. za pomocą rozkazu **fldcw**

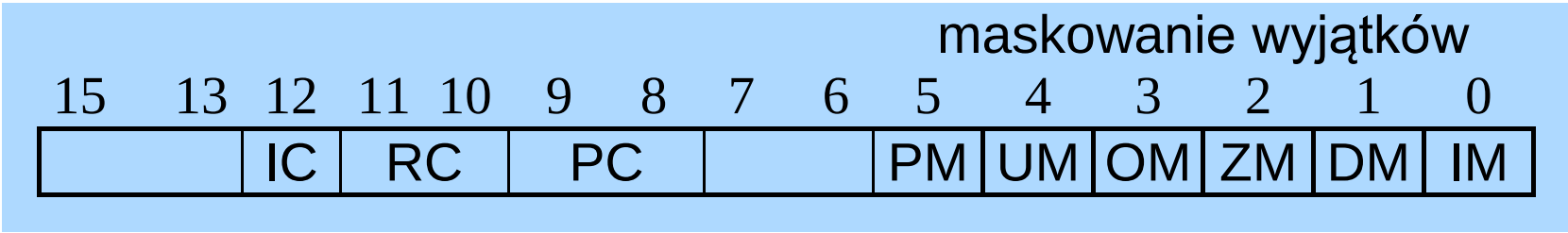
```
tryb_pracy    dw   037FH ; typowa wartość
- - - - - - - - - - -

              fldcw     tryb_pracy
```

POLITECHNIKA GDAŃSKA

Rejestr sterujący koprocesora (2)

- Poprzez ustawienie odpowiednich bitów w tym rejestrze można określić reguły zaokrąglania, sposób reagowania koprocesora na pewne zdarzenia w trakcie obliczeń (np. dzielenie przez zero) i inne.
- Zawartość rejestru sterującego koprocesora można także odczytać i zapisać na poziomie języka C za pomocą funkcji **_control87**



Rodzaje zaokrąglenia

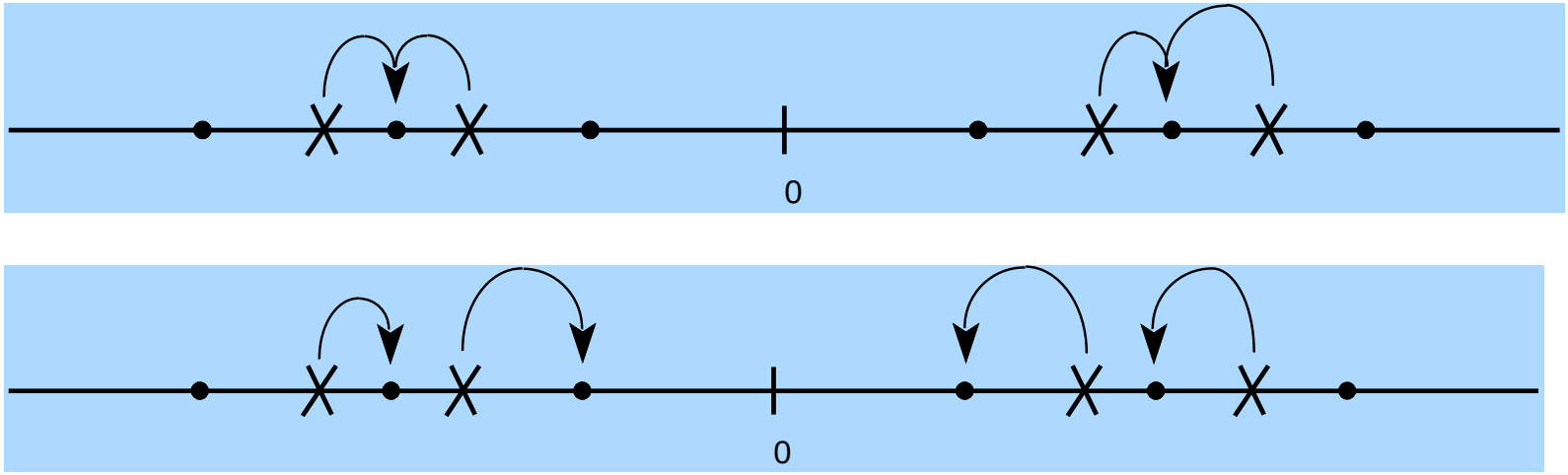
- Bity RC określają rodzaj stosowanego zaokrąglenia (do liczby reprezentowalnej w formacie zmiennoprzecinkowym, na rysunku w postaci kropki):

  00 — zaokrąglenie do liczby najbliższej (zob. rys.)

  11 — zaokrąglenie w kierunku zera (zob. rys.)

  01 — zaokrąglenie w dół (w kierunku $-\infty$)

  10 — zaokrąglenie w górę (w kierunku $+\infty$)



Rozkazy przesyłania danych (1)

- W wielu obliczeniach wykonywanych przez koprocesor jednym z argumentów jest rejestr będący wierzchołkiem stosu rejestrów koprocesora.
- W operacjach przesyłania danych podstawowe znaczenia mają dwa rozkazy:

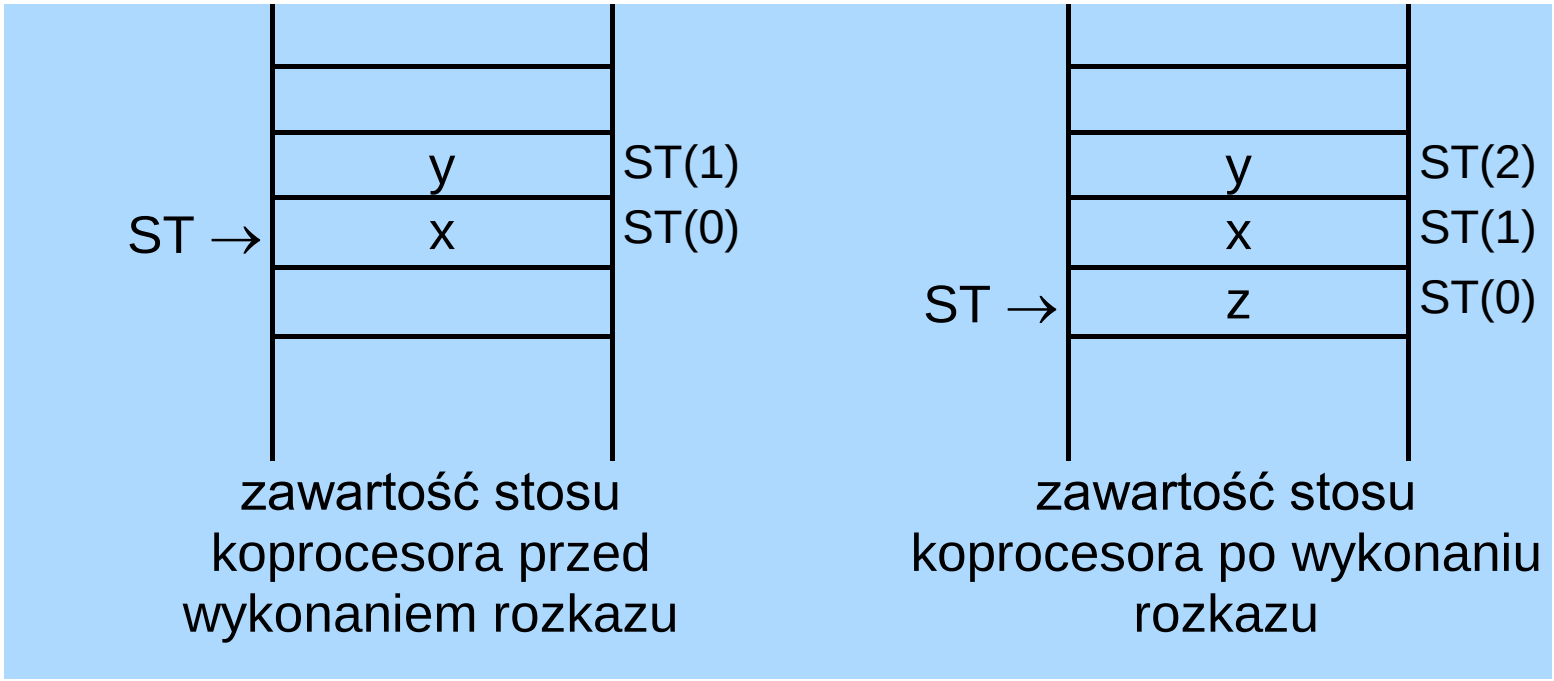
  `FLD` — ładowanie na wierzchołek stosu koprocesora liczby zmiennoprzecinkowej pobranej z lokacji pamięci lub ze stosu koprocesora;

  `FST` — przesłanie zawartości wierzchołka stosu do lokacji pamięci lub do innego rejestru stosu koprocesora.

Rozkazy przesyłania danych (2)

```
z   dq      -27.6082
_   _   _   _   _   _

    fld     z
```



Rozkazy przesyłania danych (3)

- Obok podstawowych rozkazów **`FLD`** i **`FST`** istnieje kilka odmian tych rozkazów przeznaczonych do różnych zastosowań.
- Rozkaz **`FILD`** pobiera z pamięci liczbę całkowitą w kodzie U2, następnie zamienia ją na liczbę zmiennoprzecinkową w formacie 80-bitowym i zapisuje na stosie rejestrów koprocesora.
- Rozkaz **`FIST`** przesyła liczbę z wierzchołka stosu do lokacji pamięci z jednoczesnym przekształceniem jej na format całkowity w kodzie U2.

Rozkazy przesyłania danych (4)

- Argumentami rozkazów koprocesora mogą być więc zawartości rejestrów stosu koprocesora st(0), st(1), . . . , st(7) lub zawartości lokacji pamięci. Koprocesor może odczytywać lub zapisywać do pamięci liczby zmiennoprzecinkowe 32-, 64- i 80-bitowe, jak również liczby całkowite (w kodzie U2) 16-, 32- i 64-bitowe. Akceptowane są również liczby w kodzie BCD.
- Zawartości rejestrów procesora (np. ECX) nie mogą być argumentami rozkazów koprocesora – z tego względu poniższy rozkaz jest błędny:

```
fld   ecx ; błąd !
```

Rozkazy przesyłania danych (5)

- Dostępnych jest kilka rozkazów, które wpisują stałe matematyczne na wierzchołek stosu:

FLDZ – ładowanie 0

FLD1 – ładowanie 1

FLDPI – ładowanie π

FLDL2E – ładowanie $\log_2 e$

FLDL2T – ładowanie $\log_2 10$

FLDLG2 – ładowanie $\log_{10} 2$

FLDLN2 – ładowanie $\log_e 2$

Działania arytmetyczne (1)

- Działania arytmetyczne wykonywane są przez rozkazy:

  `FADD` dodawanie

  `FSUB` odejmowanie

  `FMUL` mnożenie

  `FDIV` dzielenie

  na operandach oznaczonych jako <źródło> i <cel> — rezultat wpisywany jest do <cel>

- Podstawowy format jest następujący:

  <cel> ← <cel> { +, –,* , / } źródło

  pierwszy operand **drugi operand**

Działania arytmetyczne (2)

- We wszystkich przypadkach jednym z operandów musi być wierzchołek stosu.
- Przykładowo rozkaz

```
FDIV        ST(5), ST(0)
```

wykonuje obliczenie

$$ST(5) \leftarrow \frac{ST(5)}{ST(0)}$$

Działania arytmetyczne (3)

- Jeden z operandów może znajdować się w pamięci — w zapisie asemblerowym takich rozkazów pomija się operand ST(0), np. rozkaz

```
FSUB        qword  PTR  wymiar
```

wykonuje obliczenie

$$ST(0) \leftarrow ST(0) - wymiar$$

- Symbol **qword** oznacza, że zmienna **wymiar** jest 64-bitowa (double), **dword** oznacza zmienną 32-bitową, a **word** oznacza zmienną 16-bitową.
- Jeśli operand w pamięci jest liczbą całkowitą w kodzie U2, to stosuje się mnemonik rozkazu z dodatkową literą **I**, np.

```
FIADD   word  PTR  wspolczynnik
```

Działania arytmetyczne (4)

- Dalsze informacje o rozkazach koprocesora i ich operandach podane są w instrukcji laboratoryjnej do ćw. 5.

POLITECHNIKA GDAŃSKA

Wartości specjalne w koprocesorze arytmetycznym (1)

- Złożone obliczenia numeryczne trwają czasami wiele godzin czy nawet dni.
- Wystąpienie nadmiaru lub niedomiaru nie powinno powodować załamania programu (praktyka wskazuje, że w złożonych obliczeniach wyniki pośrednie z nadmiarem czy niedomiarem często mają niewielki wpływ na wynik końcowy).

POLITECHNIKA GDAŃSKA

Wartości specjalne w koprocesorze arytmetycznym (2)

- Spośród dopuszczalnych wartości liczb wyłączono niektóre i nadano im znaczenie specjalne — takie liczby określane są terminem wartości specjalne.
- Wartości specjalne mogą być argumentami obliczeń tak jak zwykłe liczby — jeśli jeden z argumentów jest wartością specjalną, to wynik jest niekiedy też wartością specjalną.
- W koprocesorze arytmetycznym przyjęto, że wszystkie liczby, których pole wykładnika zawiera same zera lub same jedynki traktowane są jako wartości specjalne.

POLITECHNIKA GDAŃSKA

Wartości specjalne w koprocesorze arytmetycznym (3)

- W szczególności wyróżnia się następujące wartości specjalne:
  - liczba 0 (pole mantysy i wykładnika zawiera wyłącznie bity o wartości 0);
  - nieskończoność (pole mantysy zawiera same zera, pole wykładnika zawiera same jedynki);
  - liczby z niedomiarem (pole wykładnika zawiera same zera, pole mantysy jest różne od zera);
  - nieliczby (NaN) (pole wykładnika zawiera same jedynki, pole mantysy jest różne od zera).

POLITECHNIKA GDAŃSKA

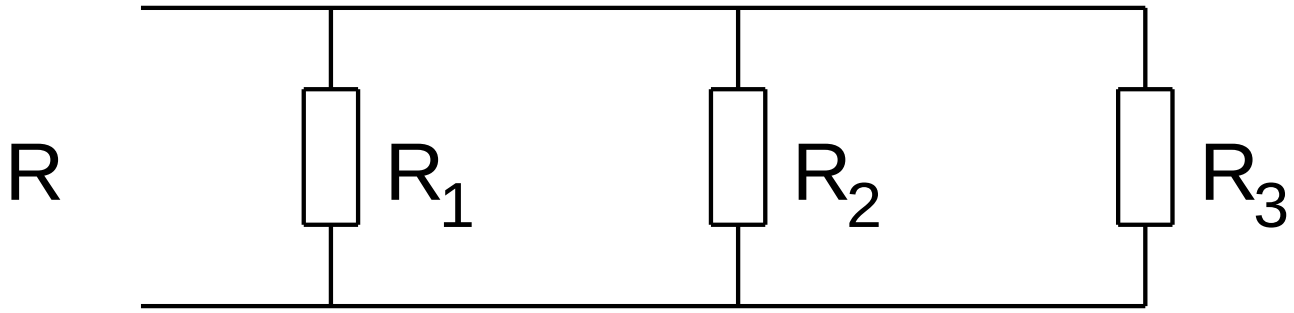
Wartości specjalne w koprocesorze arytmetycznym (4)

- W zależności od ustawienia bitów w rejestrze sterującym koprocesora, wystąpienie wartości specjalnej może powodować generowanie wyjątku koprocesora, albo też obliczenia mogą być kontynuowane.

POLITECHNIKA GDAŃSKA

Przykład — obliczanie rezystancji wypadkowej (1)

- Wartość rezystancji R dla podanego układu można wyznaczyć z zależności



$$R = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}}$$

POLITECHNIKA GDAŃSKA

Przykład — obliczanie rezystancji wypadkowej (2)

- Jeśli jedna z rezystancji ma wartość 0, to wystąpi dzielenie przez 0, a w ślad za tym koprocesor wygeneruje wyjątek.
- Jeśli jednak przez rozpoczęciem obliczeń do bitu **ZM** w rejestrze sterującym koprocesora zostanie wpisane 1, to wyjątek nie wystąpi, a wynikiem dzielenia będzie wartość specjalna „nieskończoność” (same jedynki w polu wykładnika, same zera w polu mantysy).
- Zatem, zamaskowanie wyjątku "dzielenie przez zero" pozwala na poprawne obliczenie rezystancji R podanego układu, także w przypadku, gdy wartość rezystancji R_1 lub R_2 lub R_3 wynosi 0.

Niedokładny wynik

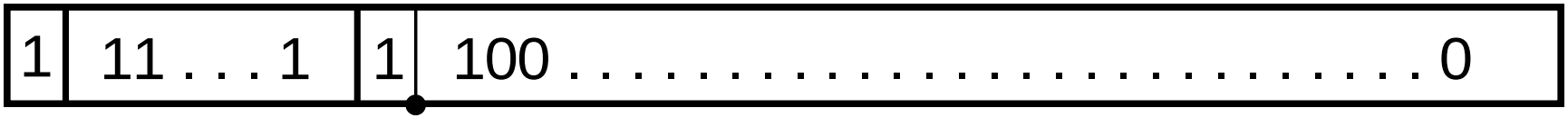
- Charakterystycznym przykładem wyjątku, który prawie zawsze jest maskowany jest *niedokładny wynik*, który sygnalizowany jest przez ustawienie bitu PE (ang. precision exception).
- Wyjątek *niedokładny wynik* powstaje, gdy wynik operacji nie może być dokładnie przedstawiony w żądanym formacie, np. wynik obliczenia

$$1/3 = (0.010101...)_2$$

ma okresowe nieskończone rozwinięcie binarne i nie może być przedstawiony dokładnie w postaci liczby zmiennoprzecinkowej, w której mantysa zajmuje ustaloną liczbę bitów.

Niedozwolona operacja

- Wyjątek *niedozwolona operacja* powstaje, gdy niemożliwe jest żadne inne działanie, np. próba obliczenia pierwiastka z liczby ujemnej, próba użycia pustego rejestru stosu.
- Jeśli bit IE jest zamaskowany (tj. IM = 1), to wynikiem operacji jest *nieliczba* – NaN (ang. Not a Number).
- Przykładowo, jeśli bit IE jest zamaskowany, to rozkaz FSQRT dla operandu –2 daje wynik:



POLITECHNIKA GDAŃSKA

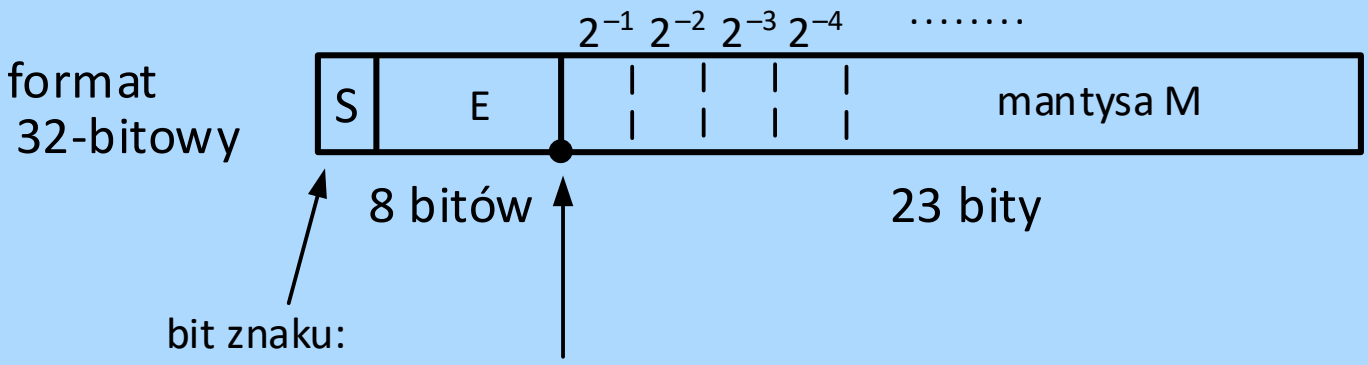
Zakresy liczb zmiennoprzecinkowych

- W formacie 32-bitowym można kodować liczby o wartościach do **$3.40*10^{38}$**, a w formacie 64-bitowym nawet do **$1.80*10^{308}$**.
- Możliwe jest także kodowanie liczb bardzo małych, np. w formacie 64-bitowym można zakodować z zadowalającą dokładnością (ok. 14 cyfr dziesiętnych) liczby bliskie **$2.23*10^{-308}$**.
- Takie same zakresy dotyczą również liczb ujemnych.

POLITECHNIKA GDAŃSKA

Uwagi o obliczeniach w arytmetyce zmiennoprzecinkowej (1)

- W przeciwieństwie do stałoprzecinkowych reprezentacji liczb całkowitych, reprezentacja zmiennoprzecinkowa liczb jest reprezentacją przybliżoną.
- Poniżej rozpatrzymy ten problem na przykładzie kodowania w formacie 32-bitowym (float).



POLITECHNIKA GDAŃSKA

Uwagi o obliczeniach w arytmetyce zmiennoprzecinkowej (2)

- Dokładna reprezentacja liczba całkowitych w formacie 32-bitowym możliwa jest dla liczb o wartości bezwzględnej mniejszej od 2^{24} , tj. takich, których rozwinięcie binarne mantysy zawiera nie więcej niż 24 bity.
- W tabeli podanej na następnym slajdzie można zauważyć, że w formacie 32-bitowym niektóre liczby całkowite większe od 2^{24} (16 777 216) są zastępowane przez ich przybliżenia. W tym przypadku konieczne jest usunięcie najmniej znaczących bitów, ponieważ pole mantysy jest 24 (23) bitowe.

POLITECHNIKA GDAŃSKA

Uwagi o obliczeniach w arytmetyce zmiennoprzecinkowej (3)

Liczba	Reprezentacja zmiennoprzecinkowa w formacie 32-bitowym (float)		
	wykładnik $_{10}$	mantysa $_{16}$	wartość $_{10}$
$2^{24} - 4 = 16777212$	150	FFFFFC	16777212
$2^{24} - 3 = 16777213$	150	FFFFFD	16777213
$2^{24} - 2 = 16777214$	150	FFFFFE	16777214
$2^{24} - 1 = 16777215$	150	FFFFFF	16777215
$2^{24} = 16777216$	151	800000	16777216
$2^{24} + 1 = 16777217$	151	800000	16777216
$2^{24} + 2 = 16777218$	151	800001	16777218
$2^{24} + 3 = 16777219$	151	800002	16777220
$2^{24} + 4 = 16777220$	151	800002	16777220

POLITECHNIKA GDAŃSKA

Uwagi o obliczeniach w arytmetyce zmiennoprzecinkowej (4)

- W arytmetyce zmiennoprzecinkowej wyniki obliczeń są przybliżone.
- W praktyce można przyjąć, że reprezentacja dziesiętna liczb typu float nie powinna zajmować więcej niż 6-7 pozycji, a liczb typu double – 14 pozycji. W takich przypadkach do wyświetlania liczb na ekranie może być wygodny format wykładniczy e, np. 3.3589e-12 (dostępny w językach wysokiego poziomu).

POLITECHNIKA GDAŃSKA

Uwagi o obliczeniach w arytmetyce zmiennoprzecinkowej (5)

- W szczególności, w przypadku operacji dodawania dwóch liczb, z których jedna jest bardzo duża, a druga bardzo mała, wartość wyniku jest określona wyłącznie przez wartość liczby bardzo dużej.
- W przypadku sumowania większej ilości liczb należy dodawać liczby w kolejności od najmniejszej do największej.

POLITECHNIKA GDAŃSKA

Uwagi o obliczeniach w arytmetyce zmiennoprzecinkowej (6)

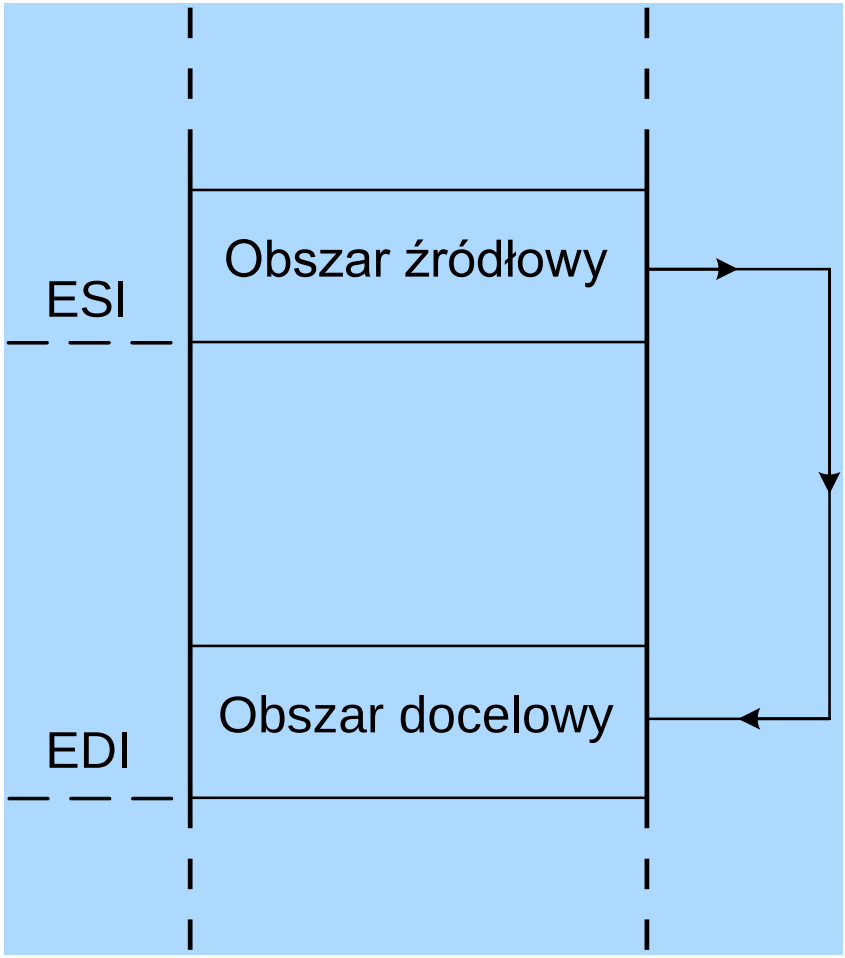
- Poruszone problemy ilustruje poniższy fragment programu w języku C.

```
float a = 17000000.0, b = 16999999.0;
printf("\n%e\n", a-b );
// na ekranie zostanie wyświetlona wartość 0
```

POLITECHNIKA GDAŃSKA

Rozkazy wykonywane na blokach danych (1)

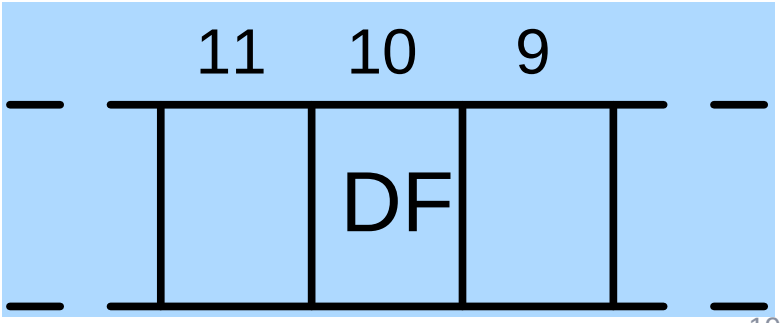
- W wielu programach występuje przepisywanie dużych obszarów danych a także ich przeszukiwanie — operacje te można zrealizować w konwencjonalny sposób, stosując zwykłą pętlę rozkazową.
- W architekturze x86 dostępne są jednak specjalnie zaprojektowane rozkazy realizujące takie operacje jako wewnętrzne działania procesora.



POLITECHNIKA GDAŃSKA

Rozkazy wykonywane na blokach danych (2)

- Rozkaz **`MOVSB`** powoduje przesłanie bajtu wskazanego przez adres zawarty w rejestrze ESI do lokacji pamięci wskazanej przez rejestr EDI, i następnie zawartości obu tych rejestrów są:
  - zwiększane o 1, gdy bit DF w rejestrze stanu procesora zawiera 0;
  - zmniejszane o 1, gdy bit DF w rejestrze stanu procesora zawiera 1.



POLITECHNIKA GDAŃSKA

Rozkazy wykonywane na blokach danych (3)

- Rozkaz **`MOVSW`** powoduje przesłanie słowa (16 bitów), a rozkaz **`MOVSD`** powoduje przesłanie podwójnego słowa (32 bity) — odpowiednio zmieniane są też (o 2 lub 4) zawartości rejestrów indeksowych ESI i EDI.
- Rozkaz **`CLD`** wpisuje 0 do znacznika DF, a rozkaz **`STD`** wpisuje 1 do znacznika DF.

Przedrostek powtarzania REP

- Przedrostek powtarzania **`REP`** (kod F3H) umieszczony przed rozkazem **`MOVSB`**, **`MOVSW`** lub **`MOVSD`**, powoduje powtarzanie wykonywania rozkazu — po każdym powtórzeniu zawartość rejestru ECX jest zmniejszana o 1, a gdy ECX = 0, to powtarzanie zostaje zakończone.

POLITECHNIKA GDAŃSKA

Przykład kopiowania obszaru pamięci

```
liczby_pierwsze         db   2, 3, 5, 7, 11, 13
                        db   17, 19, 23, 29, 31
                   db   37
tablica            db   12  dup  (?)
_   _   _   _   _   _   _   _   _   _   _

     mov       ecx, 12
     cld
     mov       esi,  OFFSET  liczby_pierwsze
     mov       edi,  OFFSET  tablica
     REP       movsb
```

Inne operacje blokowe

- Rozkazy **LODS**… (np. **LODSW**) i **STOS**… można uważać za uproszczone wersje rozkazu **MOVS**…
- Rozkaz **LODS**… przesyła zawartość 1-, 2- lub 4 bajtowego obszaru wskazanego przez rejestr ESI do rejestru AL, AX lub EAX.
- Rozkaz **STOS**… przesyła zawartość rejestru AL, AX lub EAX do lokacji pamięci wskazanej przez rejestr EDI; rozkaz **STOS**… poprzedzony przedrostkiem REP jest przydatny do inicjalizacji dużych obszarów pamięci.
- Do porównywania zawartości obszarów pamięci używane są rozkazy **CMPS** i **SCAS**

POLITECHNIKA GDAŃSKA

Zasady kodowania instrukcji (rozkazów) (1)

- Wszelkie informacje w komputerze kodowane są w postaci ciągów zerojedynkowych — taka postać informacji wynika ze stosowania elementów elektronicznych, które pracują pewnie i stabilnie jako elementy dwustanowe.
- Zatem również poszczególne instrukcje (rozkazy) programu muszą być przedstawiane w postaci ciągów zero-jedynkowych; producent procesora (np. firma Intel) ustala szczegółowe zasady kodowania rozkazów, przyporządkowując każdemu z nich ustalony ciąg zer i jedynek; przykładowo rozkazom mnożenia i dzielenia przypisano następujące kody:

POLITECHNIKA GDAŃSKA

Zasady kodowania instrukcji (rozkazów) (2)

mnożenie liczb 8-bitowych bez znaku 11110110..100

mnożenie liczb 32 (lub 16)-bitowych bez znaku 11110111..100

mnożenie liczb 8-bitowych ze znakiem 11110110..101

mnożenie liczb 32 (lub 16)-bitowych ze znakiem 11110111..101

dzielenie liczby 16-bitowej bez znaku przez liczbę 8-bitową

11110110..110

dzielenie liczby 64 (lub 32)-bitowej bez znaku przez liczbę 32 (16)-bitową 11110111..110

dzielenie liczby 16-bitowej ze znakiem przez liczbę 8-bitową

11110110..111

dzielenie liczby 64 (lub 32)-bitowej ze znakiem przez liczbę 32 (16)-bitową 11110111..111

POLITECHNIKA GDAŃSKA

Kodowanie operacji dwuargumentowych (1)

- Ciąg zerojedynkowy opisujący instrukcję, która wykonuje operację dwuargumentową, np. dodawania W ← A + B powinien zawierać następujące elementy (pola):

1. kod operacji (zakodowany opis wykonywanych czynności);
2. położenie pierwszego operandu (A);
3. położenie drugiego operandu (B);
4. informacje dokąd przesłać wynik (W);
5. gdzie znajduje się kolejna instrukcja do wykonania ? (tylko wyjątkowo, jeśli w procesorze nie jest używany wskaźnik instrukcji).

POLITECHNIKA GDAŃSKA

Kodowanie operacji dwuargumentowych (2)

- Konstruktorzy procesorów, wybierając sposób kodowania rozkazów, zwracają szczególną uwagę na możliwość kodowania programów za pomocą możliwie krótkiego kodu.
- W celu zmniejszenia liczby bitów rozkazu w wielu procesorach, w tym także w architekturze x86, przyjęto różne ograniczenia:

  1. wynik operacji przesyłany do lokacji, w której dotychczas znajdował się pierwszy operand (A);

POLITECHNIKA GDAŃSKA

Kodowanie operacji dwuargumentowych (3)

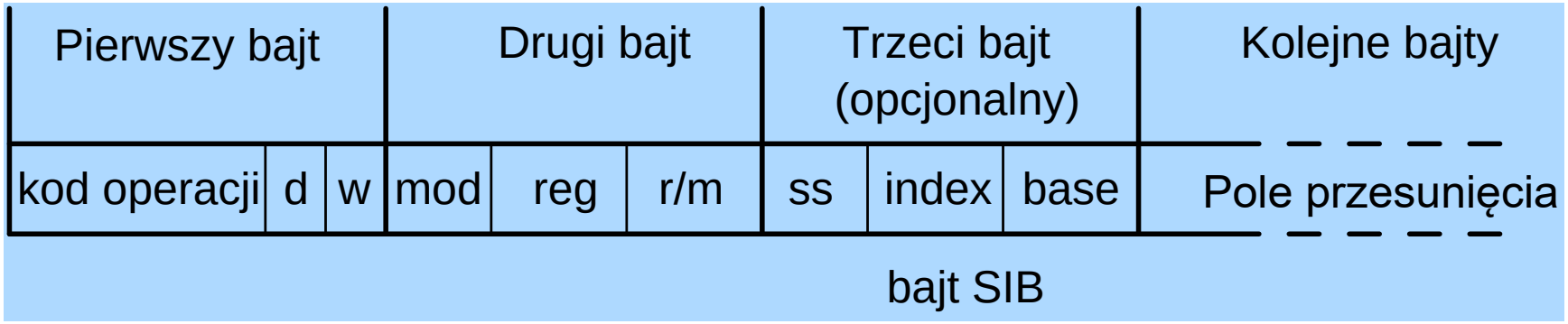
2. co najwyżej jeden operand (A albo B) może wskazywać na lokację pamięci, drugi operand lub oba operandy muszą wskazywać na rejestry;

3. położenie lokacji pamięci może być określone przez zawartość rejestru indeksowego.

POLITECHNIKA GDAŃSKA

Podstawowy format rozkazu w architekturze x86 (1)

- W architekturze x86 zdefiniowano kilkanaście formatów rozkazów sterujących (skoków) i niesterujących.
- Formaty rozkazów niesterujących wywodzą się z tzw. formatu podstawowego.



POLITECHNIKA GDAŃSKA

Podstawowy format rozkazu w architekturze x86 (2)

- Trzybitowe pole **reg** identyfikuje rejestr procesora, w którym znajduje się jeden z operandów.
- Jeśli przed kodem rozkazu zostanie umieszczony dodatkowy bajt 66H (*przedrostek rozmiaru operandu*), to działanie zostanie wykonane na operandzie 16-bitowym (a nie na 32-bitowym)

reg	000	001	010	011	100	101	110	111
w = 0	AL	CL	DL	BL	AH	CH	DH	BH
w = 1	AX	CX	DX	BX	SP	BP	SI	DI
w = 1	EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI

POLITECHNIKA GDAŃSKA

Podstawowy format rozkazu w architekturze x86 (3)

- Przyjęty sposób kodowania, w zależności od stanu bitu **d** umożliwia przesłanie wyniku do:
  - obiektu wskazanego przez pole r/m, gdy **d = 0**;
  - obiektu wskazanego przez pole reg, gdy **d = 1**;

POLITECHNIKA GDAŃSKA

Podstawowy format rozkazu w architekturze x86 (4)

- Drugi operand może znajdować się także w rejestrze (gdy dwubitowe pole **mod=11**) — w takim przypadku kod rejestru podany jest w 3-bitowym polu **r/m**
- Jeśli drugi operand znajduje się w lokacji pamięci (gdy pole mod=00,01,10), to pole **r/m** określa sposób obliczania adresu efektywnego wg podanej tablicy.

POLITECHNIKA GDAŃSKA

Podstawowy format rozkazu w architekturze x86 (5)

r/m	
000	EAX + przesunięcie
001	ECX + przesunięcie
010	EDX + przesunięcie
011	EBX + przesunięcie
100	sposób oblicz. określa bajt SIB
101	EBP + przesunięcie
110	ESI + przesunięcie
111	EDI + przesunięcie

POLITECHNIKA GDAŃSKA

Podstawowy format rozkazu w architekturze x86 (6)

- Pole **mod** określa długość (liczbę bajtów) pola przesunięcia.

- W przypadku **mod**=00 pole przesunięcie nie występuje, gdy **mod** = 01, to pole przesunięcia jest 1-bajtowe, a gdy **mod** = 10, to pole przesunięcia jest 4-bajtowe.

POLITECHNIKA GDAŃSKA

Kodowanie złożonych schematów adresowania (1)

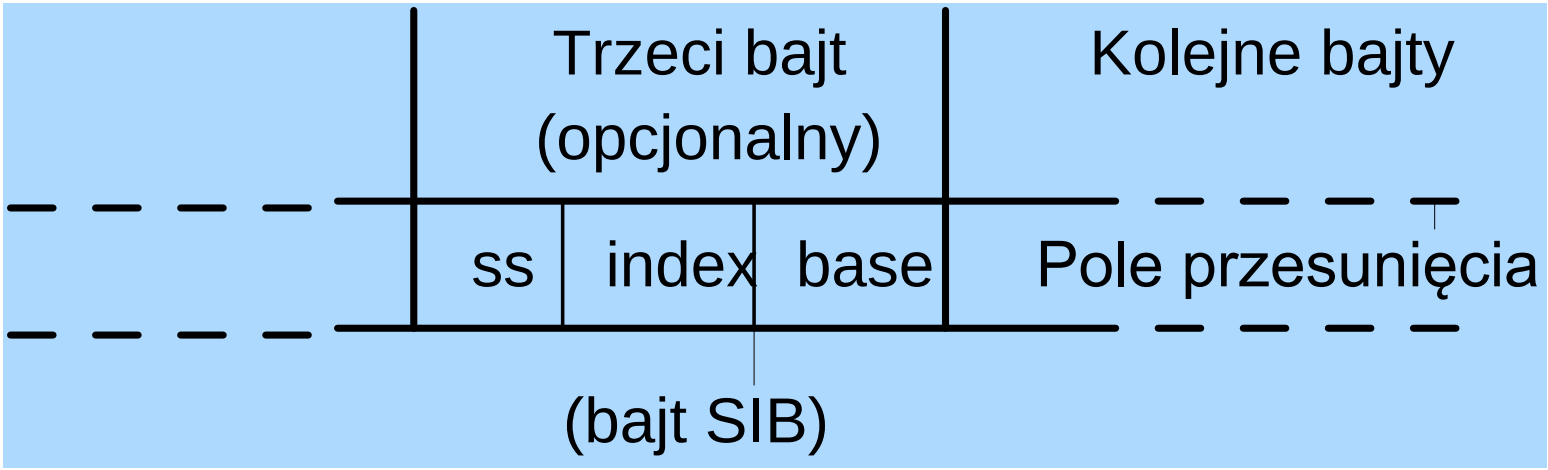
- W architekturze x86 wartość wyrażenia adresowego rozkazu może zależeć od zawartości dwóch rejestrów bazowego i indeksowego, np.

```
add    edx,  tablica[ebx+esi]
```

- Kodowanie takich rozkazów wymaga wprowadzenia dodatkowego bajtu oznaczonego symbolem **SIB**.
- Obecność tego bajtu wskazuje pole **r/m=100** (w drugim bajcie).

POLITECHNIKA GDAŃSKA

Kodowanie złożonych schematów adresowania (2)



- Pole **base** (3 bity) określa rejestr bazowy, a pole **index** (3 bity) określa rejestr indeksowy.
- Pole **ss** (2 bity) określa współczynnik skalowania, np.
  sub wyniki[ecx+edi*4], bx

POLITECHNIKA GDAŃSKA

Kodowanie złożonych schematów adresowania (3)

- Jeśli występuje bajt SIB, to w trakcie obliczania wartości adresu efektywnego zawartość drugiego rejestru modyfikacji jest mnożona przez współczynnik skalowania, który przyjmuje wartości: 1 (gdy ss=00), 2 (gdy ss=01), 4 (gdy ss=10), 8 (gdy ss=11).

POLITECHNIKA GDAŃSKA

Przypadki szczególne kodowania (1)

- W podanej wcześniej tabeli wymieniono 8 różnych sposobów obliczania adresu efektywnego, ale w każdym z nich zawartość pola przesunięcie jest sumowana z zawartością jakiegoś rejestru.
- W praktyce programowania zdarza się dość często, że adres lokacji pamięci, na której ma być wykonana operacja podany jest w polu przesunięcia i nie potrzeba do niego niczego dodawać — wydaje się, że konstruktorzy procesora nie uwzględnili tego przypadku.

POLITECHNIKA GDAŃSKA

Przypadki szczególne kodowania (2)

- Powstałą sytuację rozwiązano poprzez wprowadzenie przypadku specjalnego: gdy pola **mod=00** i **r/m=101**, to adres efektywny określony jest wyłącznie przez zawartość 32-bitowego pola przesunięcie (indeksowanie nie występuje).

Przykłady kodowania (1)

Pierwszy bajt			Drugi bajt			Trzeci bajt	Czwarty bajt	Piąty bajt	Szósty bajt
Kod operacji	d	w	mod	reg	r/m	Przesunięcie			
mov DL, [ESI] + 800									
10001010			10010110			00100000	00000011	00000000	00000000

sub DX, 0B30AH				
01100110	10000001	11101010	00001010	10110011

Przykłady kodowania (2)

	Pierwszy bajt	Drugi bajt	
jz	01110100	przesunięcie	warunek spełniony gdy ZF = 1
jnz	01110101	przesunięcie	warunek spełniony gdy ZF = 0
jc	01110010	przesunięcie	warunek spełniony gdy CF = 1
jnc	01110011	przesunięcie	warunek spełniony gdy CF = 0

POLITECHNIKA
GDAŃSKA

HISTORIA MĄDROŚCIĄ
PRZYSZŁOŚĆ WYZWANIEM