Języki asemblerowe

WYKŁAD 6

Dr Krzysztof Balicki

Operacje zmiennoprzecinkowe

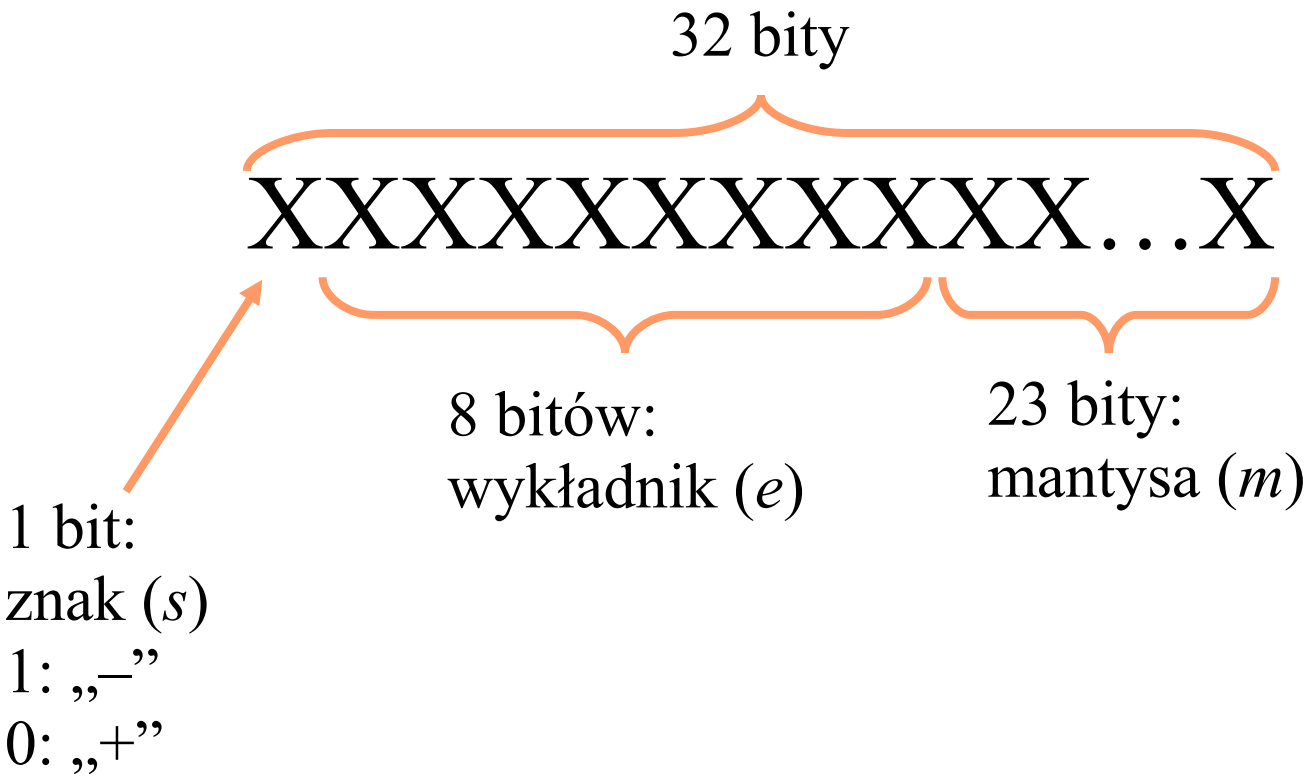
- W procesorze Pentium dla liczb zmiennoprzecinkowych obowiązuje standard IEEE 754.
- Procesor Pentium wspiera trzy formaty liczb zmiennoprzecinkowych:
  - formaty do użytku zewnętrznego:
    - liczby pojedynczej precyzji (32 bity),
    - liczby podwójnej precyzji (64 bity),
  - format do wewnętrznego użytku:
    - liczby rozszerzonej precyzji (80 bitów).

Operacje zmiennoprzecinkowe

- Dla wcześniejszych wersji procesorów rodziny 8086, do obliczeń zmiennoprzecinkowych wykorzystywane były koprocesory, np.
  - dla procesora 8086 stworzono koprocesor 8087,
  - dla procesora 80286 stworzono koprocesor 80287,
  - dla procesora 80386 stworzono koprocesor 80387.
- Począwszy od procesora 80486, koprocesor (jednostka zmiennoprzecinkowa) została zintegrowana z procesorem.

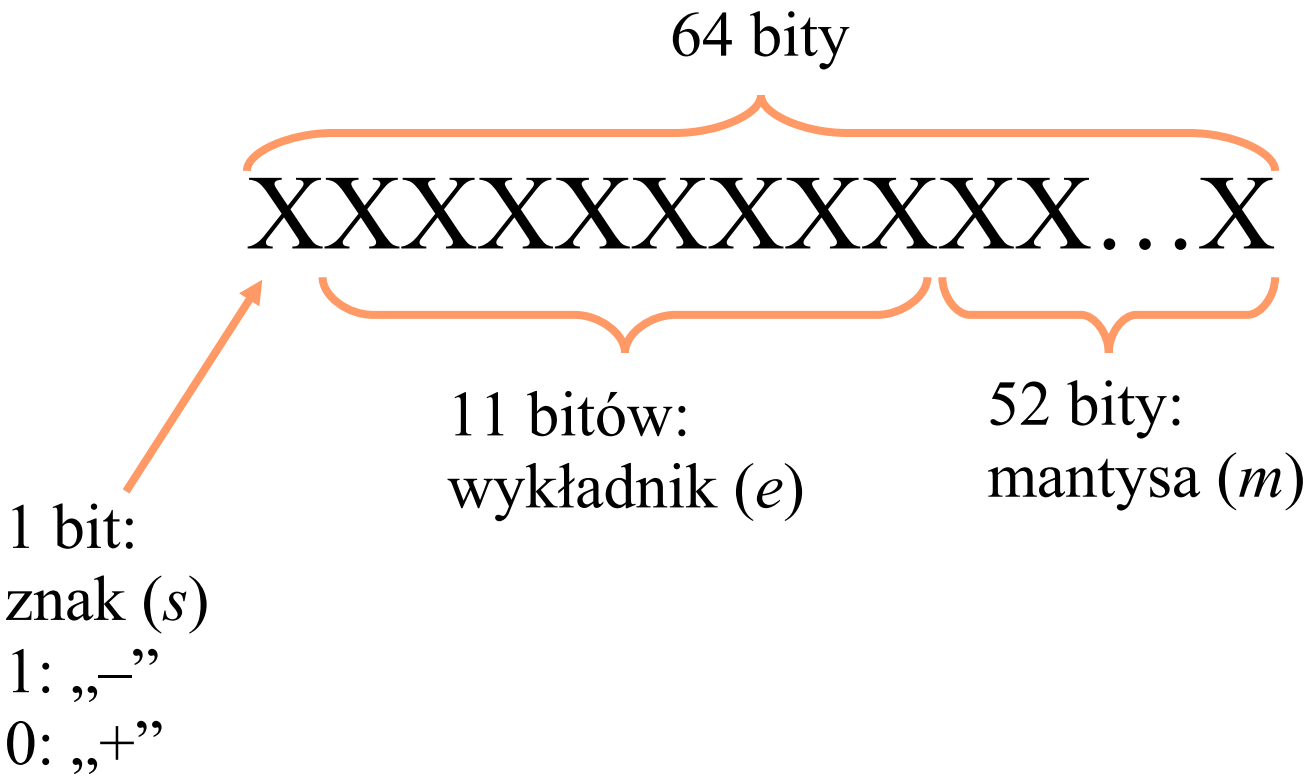
Standard IEEE 754 - liczby pojedynczej precyzji

$$A = (-1)^s \cdot m \cdot 2^{e-127}$$



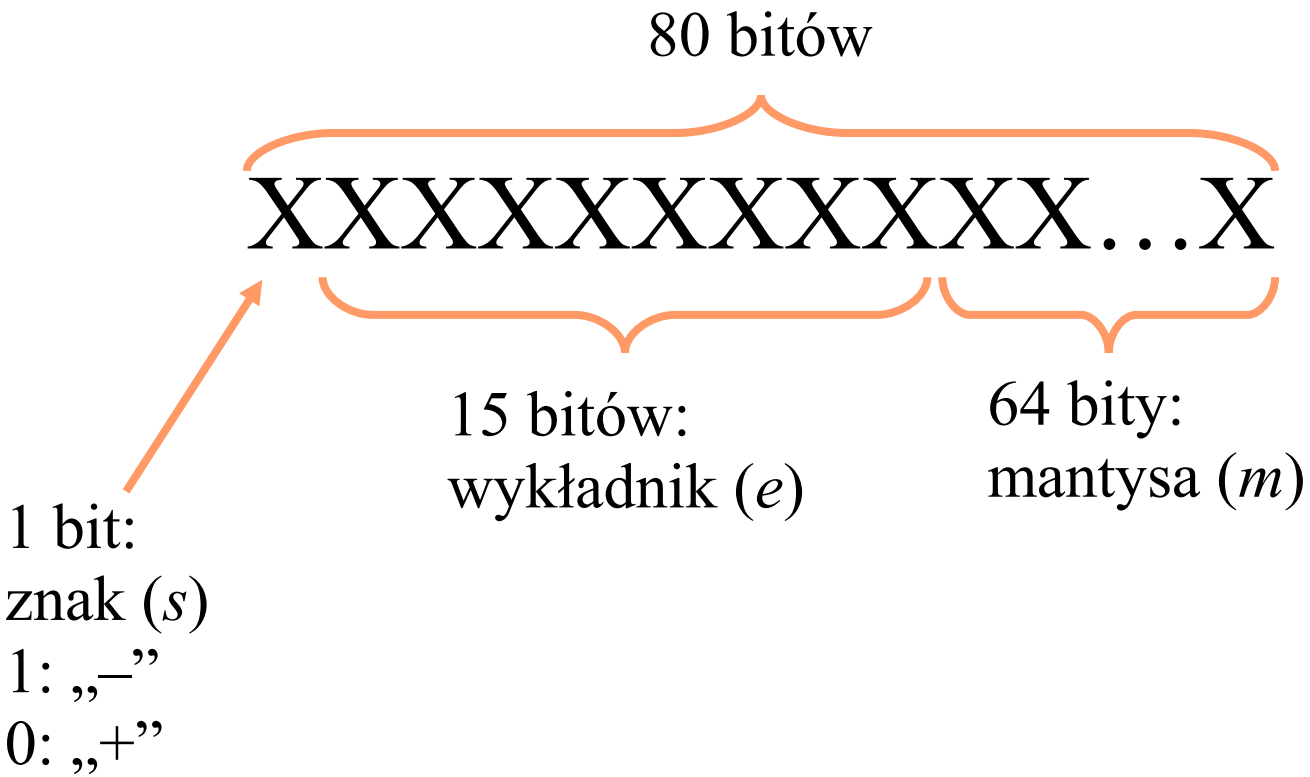
Standard IEEE 754 - liczby podwójnej precyzji

$$A = (-1)^s \cdot m \cdot 2^{e-1023}$$



Liczby rozszerzonej precyzji

$$A = (-1)^s \cdot m \cdot 2^{e-16383}$$



Jednostka zmiennoprzecinkowa (FPU)

- W jednostce zmiennoprzecinkowej występuje cztery rodzaje rejestrów:
  - rejestry danych,
  - rejestry kontrolne i statusu,
  - rejestry wskaźnikowe.

Rejestry danych

- FPU posiada osiem rejestrów danych do przechowywania operandów zmiennoprzecinkowych.
- Rejestry danych zorganizowane są w strukturze stosu (bufor kołowy). Rejestry danych dostępne są także indywidualnie poprzez nazwy ST0, ST1, ..., ST7. Rejestr ST0 jest rejestrem z wierzchołka stosu (Top-Of-Stack, TOS), itd.
- W rejestrze statusu istnieje 3-bitowy wskaźnik rejestru TOS.

Rejestry danych

- Każdy rejestr danych może przechowywać 80 bitową liczbę rozszerzonej precyzji.
- Status i zawartość każdego rejestru danych są identyfikowane przez 2 bitowy znacznik. Znaczniki przechowywane są w rejestrze znaczników.

Rejestr kontrolny

- Rejestr kontrolny (16 bitów) jest używany do kontroli kilku operacji zmiennoprzecinkowych:
  - bity 5 - 0: maska sześciu wyjątków operacji zmiennoprzecinkowych: precyzja, niedomiar, nadmiar, dzielenie przez zero, zdenormalizowany operand, niepoprawna operacja

Rejestr kontrolny

- Rejestr kontrolny (16 bitów) jest używany do kontroli kilku operacji zmiennoprzecinkowych (cd.):
  - bity 9 - 8: kontrola precyzji (aby zachować kompatybilność z poprzednimi FPU o mniejszej precyzji):
    - 00 - 24 bity
    - 01 - nieużywane
    - 10 - 53 bity
    - 11 - 64 bity

Rejestr kontrolny

- Rejestr kontrolny (16 bitów) jest używany do kontroli kilku operacji zmiennoprzecinkowych (cd.):
  - bity 11 - 10: kontrola zaokrąglenia:
    - 00 - zaokrąglenie do najbliższej wartości
    - 01 - zaokrąglenie w dół
    - 10 - zaokrąglenie w górę
    - 11 - obcięcie

Rejestr statusu

- Rejestr statusu (16 bitów) przechowuje informacje o statusie FPU:
  - bity 5 - 0: flagi wyjątków
  - bit 6: błąd stosu
  - bit 7: status błędu
  - bity 14, 10 - 8: kody stanu:
    - bity 14, 10, 8 - flagi C3, C2, C0
    - bit 9: nadmiar/niedomiar stosu
  - bity 13 - 11: wskaźnik rejestru TOS

Rejestr etykiet

- Rejestr etykiet (16 bitów) przechowuje informacje o statusie i zawartości rejestrów danych (dla każdego rejestru używane jest 2 bity, dwa najmłodsze bity dla ST0, itd.):
  - 00 - poprawny
  - 01 - zero
  - 10 - specjalny (niepoprawny, nieskończoność, zdenormalizowany)
  - 11 - pusty

Instrukcje zmiennoprzecinkowe

- Wybrane instrukcje przesłań:
  - **fld zrodlo** - przesyła wartość źródłową na stos FPU (do rejestru ST0)
  - **fldz** - przesłanie +0.0 na stos FPU
  - **fld1** - przesłanie +1.0 na stos FPU
  - **fldpi** - przesłanie π na stos FPU
  - **fldl2t** - przesłanie $\log_2(10)$ na stos FPU
  - **fldl2e** - przesłanie $\log_2(e)$ na stos FPU
  - **fldlg2** - przesłanie $\log_{10}(2)$ na stos FPU
  - **fldln2** - przesłanie $\log_e(2)$ na stos FPU

Instrukcje zmiennoprzecinkowe

- Wybrane instrukcje przesłań:
  - **fild zrodlo** - przesyła całkowitą wartość źródłową na stos FPU
  - **fst przezn** - przesyła wartość ze stosu do miejsca przeznaczenia (rejestr lub pamięć) bez jej usuwania ze stosu
  - **fstp przezn** - przesyła wartość ze stosu do miejsca przeznaczenia (rejestr lub pamięć) z usunięciem jej ze stosu

Instrukcje zmiennoprzecinkowe

- Wybrane instrukcje arytmetyczne:
  - **fadd zrodlo** - dodaje wartość źródłową (z pamięci) do wartości ze stosu FPU (rejestru ST0), wynik przechowywany jest w ST0
  - **fadd przezn, zrodlo** - dodaje wartość z rejestru źródłowego do wartości z rejestru przeznaczenia (oba rejestry muszą być rejestrami FPU), wynik przechowywany jest w rejestrze przeznaczenia
  - **fsub zrodlo** - odejmuje wartość źródłową (z pamięci) od wartości ze stosu FPU (rejestru ST0), wynik przechowywany jest w ST0

Instrukcje zmiennoprzecinkowe

- Wybrane instrukcje arytmetyczne:
  - **fsub przezn, zrodlo** - odejmuje wartość z rejestru źródłowego od wartości z rejestru przeznaczenia (oba rejestry muszą być rejestrami FPU), wynik przechowywany jest w rejestrze przeznaczenia
  - **fmul zrodlo** - mnoży wartość źródłową (z pamięci) z wartością ze stosu FPU (rejestru ST0), wynik przechowywany jest w ST0
  - **fmul przezn, zrodlo** - mnoży wartość z rejestru źródłowego z wartością z rejestru przeznaczenia (oba rejestry muszą być rejestrami FPU), wynik przechowywany jest w rejestrze przeznaczenia

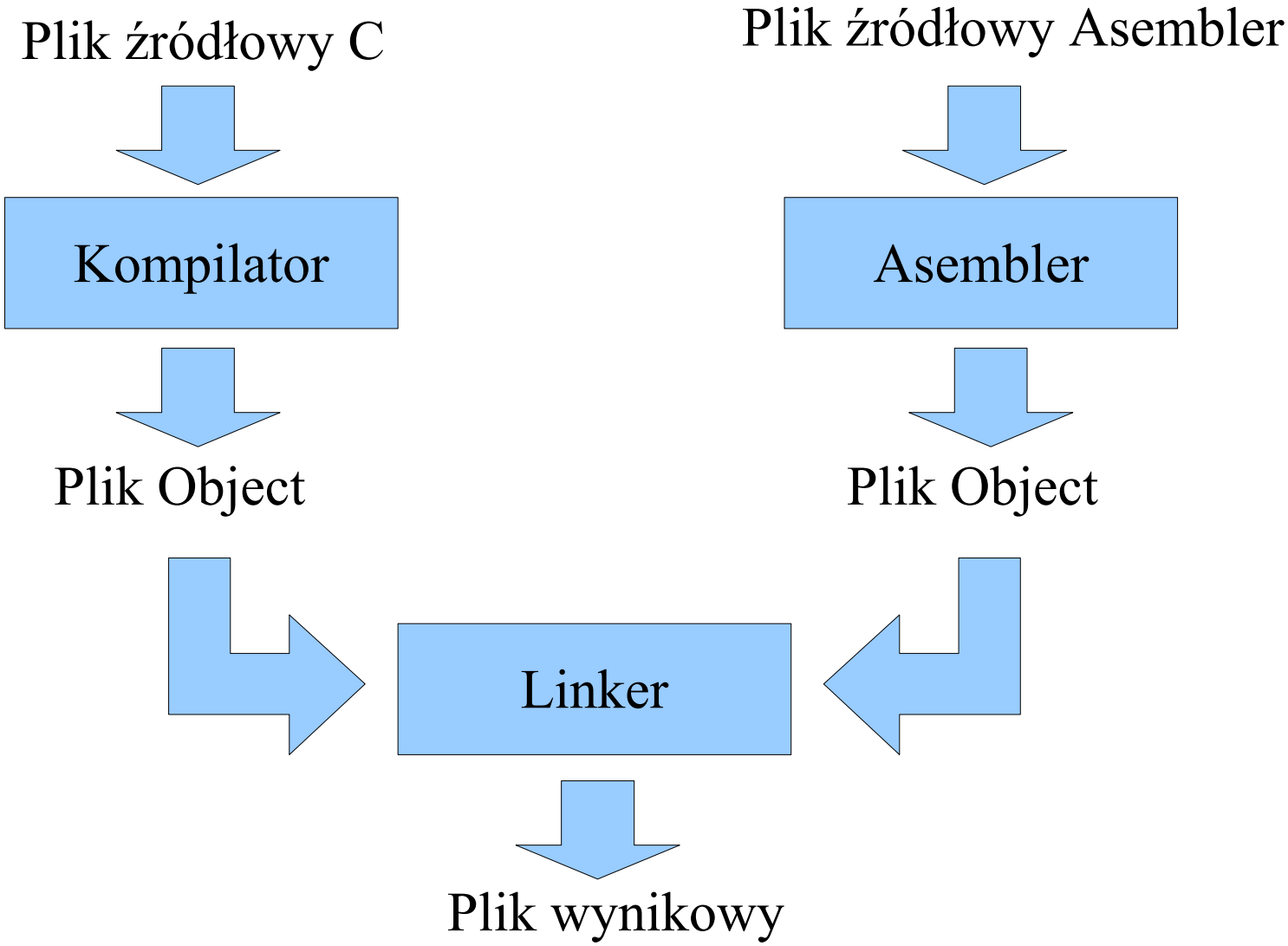
Instrukcje zmiennoprzecinkowe

- Wybrane instrukcje arytmetyczne:
  - **fdiv zrodlo** - dzieli wartość ze stosu FPU (rejestru ST0) przez wartość źródłową (z pamięci), wynik przechowywany jest w ST0
  - **fdiv przezn, zrodlo** - dzieli wartość z rejestru przeznaczenia z wartością z rejestru źródłowego (oba rejestry muszą być rejestrami FPU), wynik przechowywany jest w rejestrze przeznaczenia

Instrukcje zmiennoprzecinkowe

- Wybrana instrukcja porównania:
  - **fcom zrodlo -** porównuje wartość ze stosu FPU (rejestru ST0) z wartością źródłową (z pamięci) i ustawia flagi:
    - C3 C2 C0 = 000 jeśli ST0 > zrodlo
    - C3 C2 C0 = 100 jeśli ST0 = zrodlo
    - C3 C2 C0 = 001 jeśli ST0 < zrodlo

Interfejs dla języków wysokiego poziomu



Wywoływanie procedur Asemblera w języku C

- Przekazywanie parametrów (argumentów) odbywa się przez stos na dwa sposoby:
  - parametry pobierane są od prawej do lewej i umieszczane na stosie,
  - parametry pobierane są od lewej do prawej i umieszczane na stosie.
- Większość języków wysokiego poziomu używa sposobu od lewej do prawej, ale np. język C używa sposobu od prawej do lewej.

Wywoływanie procedur Asemblera w języku C

- Deklaracja procedury zewnętrznej w języku C:

extern *typ_zwr nazwa_proc*(*lista_typów_argumentów*) ;

- Wartość zwracana jest przez rejestr EAX (8, 16 i 32 bitowe wartości) lub przez rejestry EDX+EAX (64 bitowe wartości).

Wywoływanie procedur Asemblera w języku C

- Przykład:

```
#include <stdio.h>

int main(void)
{
   int a=2,b=3,c=4;
   int s;
   extern int suma (int, int, int);

   s=suma(a,b,c);
   printf("Suma=%d\n", s);

   return 0;
}
```

Wywoływanie procedur Asemblera w języku C

- Przykład (cd.):

```
segment .text
global suma  ;procedura globalna

suma:
enter 0,0
mov EAX,[EBP+8] ;pobranie a
add EAX,[EBP+12] ;dodanie b
add EAX,[EBP+16] ;dodanie c
leave
ret
```

- Instrukcje enter i leave służą odpowiednio do alokacji i zwalniania ramki stosu dla procedury

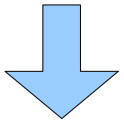
Zagnieżdżony kod asemblerowy

- Notacja AT&T
  - nazwy rejestrów poprzedzone są prefiksem %
  - instrukcje zakończone są sufiksem:
    - b - bajt
    - w - słowo
    - l - podwójne słowo
  - stałe i wartości bezpośrednie poprzedzone są prefiksem $

Zagnieżdżony kod asemblerowy

- Notacja AT&T
  - kolejność operandów w instrukcjach asemblerowych jest odwrócona (najpierw jest operand źródłowy, później operand przeznaczenia), np.:

mov EAX,EBX

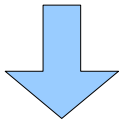


movl %ebx,%eax

Zagnieżdżony kod asemblerowy

- Notacja AT&T
  - adresy niebezpośrednie umieszczane są w nawiasach okrągłych, np.:

mov EAX,[EBX]



movl (%ebx),%eax

Zagnieżdżony kod asemblerowy

- Instrukcja asm:

```
asm(kod
:wyjscia
:wejscia
:lista_modyfikowanych_rejestrow);
```

Zagnieżdżony kod asemblerowy

- Instrukcja asm:
  - kod - kod asemblerowy
  - wyjscia - operandy wyjściowe dla kodu asemblerowego
  - wejscia - operandy wejściowe dla kodu asemblerowego
  - lista_modyfikowanych_rejestrow - lista rejestrów modyfikowanych przez kod asemblerowy

Zagnieżdżony kod asemblerowy

- Instrukcja asm:
  - operandom wyjściowym i wejściowym przypisane są kolejno oznaczenia liczbowe 0, 1, itd. (maksymalnie może być 10 operandów)
  - w kodzie asemblerowym odwołanie do operandów poprzedzone jest prefiksem %, np.: %0, %1, itd.
  - w nawiasie okrągłym podawane jest mapowanie zmiennych kodu C na rejestry

Zagnieżdżony kod asemblerowy

- Instrukcja asm:
  - operand wyjściowy specyfikowany jest przez =
  - lista modyfikowanych rejestrów może być używana przez kompilator C (informacja, że wartości rejestrów wymienionych na liście mogą być niepoprawne po wykonaniu bloku asemblerowego)

Zagnieżdżony kod asemblerowy

- Przykład 1:

```
asm("movl %1,%0"
:"=r"(a)
:"r"(b)
);
```

Zagnieżdżony kod asemblerowy

- Przykład 2:

```
int suma(int a,int b,int c)
{
   asm("movl %0,%%eax;"
      "addl %1,%%eax;"
      "addl %2,%%eax;"
      :
      : "r"(a), "r"(b), "r"(c)
      :"cc","%eax");
}
```

Rejestr znaczników

Przerwania

- Wyjątki
- Przerwania programowe
- Przerwania sprzętowe

Przerwania programowe

- Przerwania programowe są inicjalizowane przez instrukcję przerwania:

```
int typ_przerwania
```

- Typ przerwania jest liczbą całkowitą z przedziału 0 ... 255.