Sprawozdanie z przedmiotu „Wstęp do wysokowydajnych komputerów”

Grzegorz Szczepanek
Indeks: 280678

12 maja 2025

1 Cel ćwiczenia

Celem ćwiczenia było zapoznanie się z integracją kodu asemblerowego z programem w języku C/C++. W ramach zadania zaimplementowano w pliku `test.s` trzy funkcje w składni AT&T dla architektury x86_64 oraz wywołano je z programu `test.cpp` przy użyciu specyfikatora `extern "C"`.

2 Opis implementacji

2.1 Konwencja wywołań (System V AMD64 ABI)

Argumenty całkowite przekazywane są w rejestrach `RDI`, `RSI`, `RDX`, `RCX`, `R8`, `R9`, wynik w `EAX`. Argumenty zmiennoprzecinkowe trafiają do `XMM0`, `XMM1` itd., a wynik do `XMM0`. Każda funkcja musi zachować odpowiedni prolog i epilog stosu.

2.2 Funkcja `testowa`

```
// zwraca sta    42
testowa:
    pushq   %rbp
    movq    %rsp, %rbp
    movl    $42, %eax
    popq    %rbp
    ret
```

Funkcja ustawia ramkę stosu, wypełnia rejestr `EAX` wartością 42 i zwraca kontrolę. Prolog (`pushq`, `movq`) i epilog (`popq`, `ret`) chronią poprzednią ramkę stosu.

2.3 Funkcja `dodaj`

```
// dodaje dwa parametry typu int
dodaj:
    pushq   %rbp
    movq    %rsp, %rbp
    movl    %edi, %eax    # eax = pierwszy argument
    addl    %esi, %eax    # eax += drugi argument
    popq    %rbp
    ret
```

Pierwszy argument trafia do `EDI`, drugi do `ESI`. Wynik sumy wrzucany jest do `EAX`, skąd C odczytuje wartość zwracaną.

2.4 Funkcja `dodaj2`

```
// dodaje dwa parametry typu double
dodaj2:
    pushq   %rbp
    movq    %rsp, %rbp
    addsd   %xmm1, %xmm0   # xmm0 = xmm0 + xmm1
    popq    %rbp
    ret
```

Argumenty zmiennoprzecinkowe przekazywane są w rejestrach `XMM0` (pierwszy) i `XMM1` (drugi). Instrukcja `addsd` wykonuje dodawanie podwójnej precyzji.

3 Integracja z C++

W pliku `test.cpp` zadeklarowano:

```
// dyrektywy extern "C" zapobiegaj  mangle'owaniu nazw
extern "C" int testowa();
extern "C" int dodaj(int a, int b);
extern "C" double dodaj2(double a, double b);
```

Następnie w `main()` wywołano te funkcje i wypisano wyniki przy użyciu `printf`. Kompilacja odbywa się w dwóch krokach:

1. Kompilacja asemblera: `as test.s -o test.o`
2. Kompilacja i linkowanie: `g++ test.cpp test.o -o test`

4 Podsumowanie

Ćwiczenie pozwoliło na praktyczne zrozumienie:

- konwencji wywołań w x86_64,
- integracji asemblera z kodem C++,
- znaczenia prologu i epilogu w funkcjach asemblerowych.

Dzięki temu łatwiej zrozumieć, jak niskopoziomowy kod wpływa na wydajność i strukturę programu.