Zadanie 2

Otwarto: środa, 24 kwietnia 2024, 00:00
Wymagane do: niedziela, 19 maja 2024, 23:59

Dzielenie

Zaimplementuj w asemblerze wołaną z języka C funkcję o następującej deklaracji:

```
int64_t mdiv(int64_t *x, size_t n, int64_t y);
```

Funkcja wykonuje dzielenie całkowitoliczbowe z resztą. Funkcja traktuje dzielną, dzielnik, iloraz i resztę jako liczby zapisane w kodowaniu uzupełnieniowym do dwójki. Pierwszy i drugi parametr funkcji określają dzielną: `x` jest wskaźnikiem na niepustą tablicę `n` liczb 64-bitowych. Dzielna ma `64 * n` bitów i jest zapisana w pamięci w porządku cienkokońcówkowym (ang. *little-endian*). Trzeci parametr `y` jest dzielnikiem. Wynikiem funkcji jest reszta z dzielenia `x` przez `y`. Funkcja umieszcza iloraz w tablicy `x`.

Jeśli iloraz nie daje się zapisać w tablicy `x`, to oznacza wystąpienie nadmiaru (ang. *overflow*). Szczególnym przypadkiem nadmiaru jest dzielenie przez zero. Funkcja powinna reagować na nadmiar tak jak rozkazy `div` i `idiv`, czyli zgłaszać przerwanie numer 0. Obsługa tego przerwania w Linuksie polega na wysłaniu do procesu sygnału `SIGFPE`. Opis tego sygnału „błąd w obliczeniach zmiennoprzecinkowych” jest nieco mylący.

Wolno założyć, że wskaźnik `x` jest poprawny oraz że `n` ma wartość dodatnią.

Przykład użycia

Przykład użycia jest częścią treści zadania. W szczególności z przykładu użycia należy wywnioskować, jakie są zależności między znakami dzielnej, dzielnika, ilorazu i reszty. Przykład użycia znajduje się w niżej załączonym pliku `mdiv_example.c`. Można go skompilować i skonsolidować z rozwiązaniem poleceniami:

```
gcc -c -Wall -Wextra -std=c17 -O2 -o mdiv_example.o mdiv_example.c
gcc -z noexecstack -o mdiv_example mdiv_example.o mdiv.o
```

Oddawanie rozwiązania

Jako rozwiązanie należy wstawić w Moodle plik o nazwie `mdiv.asm`.

Kompilowanie

Rozwiązanie będzie kompilowane poleceniem:

```
nasm -f elf64 -w+all -w+error -o mdiv.o mdiv.asm
```

Rozwiązanie musi się kompilować w laboratorium komputerowym.

Ocenianie

Ocena będzie się składała z dwóch części.

1. Zgodność rozwiązania ze specyfikacją będzie oceniania za pomocą testów automatycznych, za które można dostać maksymalnie 7 punktów. Sprawdzane będą też przestrzeganie reguł ABI, poprawność odwołań do pamięci i zajętość pamięci. Od wyniku testów automatycznych zostanie odjęta wartość proporcjonalna do rozmiaru dodatkowej pamięci wykorzystywanej przez rozwiązanie (sekcje `.bss`, `.data`, `.rodata`, stos, sterta). Ponadto zostanie ustalony próg rozmiaru sekcji `.text`. Przekroczenie tego progu będzie skutkowało odjęciem od oceny wartości proporcjonalnej do wartości tego przekroczenia. Dodatkowym kryterium automatycznej oceny rozwiązania będzie szybkość jego działania. Rozwiązanie zbyt wolne nie uzyska maksymalnej oceny. Za błędną nazwę pliku odejmiemy jeden punkt.
2. Za formatowanie i jakość kodu można dostać maksymalnie 3 punkty. Tradycyjne formatowanie programów w

?

asemblerze polega na rozpoczynaniu etykiet od pierwszej kolumny, a mnemoników rozkazów od wybranej ustalonej kolumny. Nie stosuje się innych wcięć. Taki format mają przykłady pokazywane na zajęciach. Kod powinien być dobrze skomentowany, co oznacza między innymi, że każdy blok kodu powinien być opatrzony informacją, co robi. Należy opisać przeznaczenie rejestrów. Komentarza wymagają wszystkie kluczowe lub nietrywialne linie kodu. W przypadku asemblera nie jest przesadą komentowanie prawie każdej linii kodu, ale należy unikać komentarzy opisujących to, co widać.

Wystawienie oceny może zostać uzależnione od osobistego wyjaśnienia szczegółów działania programu prowadzącemu zajęcia.

Rozwiązania należy implementować samodzielnie pod rygorem niezaliczenia przedmiotu. Zarówno korzystanie z cudzego kodu, jak i prywatne lub publiczne udostępnianie własnego kodu jest zabronione.

Załącznik

mdiv_example.c 17 kwietnia 2024, 11:13