PIX

Zaimplementować w asemblerze funkcję, która oblicza wybrany fragment części ułamkowej dwójkowego rozwinięcia liczby π i którą można wywołać z języka C. Należy użyć w tym celu formuły Bailey–Borweina–Plouffe'a. Dla problemu obliczenia początkowych cyfr rozwinięcia π znane są formuły szybciej zbieżne, ale ta ma dwie zalety. Po pierwsze można za jej pomocą obliczać cyfry od pewnego miejsca bez znajomości poprzednich cyfr, co sprawia, że łatwo jest zrównoleglić obliczenia. Po drugie do jej implementacji wystarczy arytmetyka stałopozycyjna, co sprawia, że implementacja może być bardzo szybka.

Implementowana funkcja ma się nazywać `pix` i ma mieć następującą deklarację w języku C:

```
void pix(uint32_t *ppi, uint64_t *pidx, uint64_t max);
```

Parametr `ppi` jest wskaźnikiem na tablicę, gdzie funkcja umieszcza obliczone wartości. W elemencie o indeksie `m` tej tablicy mają się znaleźć 32 bity rozwinięcia od bitu numer `32m+1` do bitu numer `32m+32`. Bit numer `m` ma w rozwinięciu wagę 2^{-m}. Innymi słowy w `ppi[m]` ma się znaleźć wartość $[2^{32}\{2^{32m}\pi\}]$, gdzie $[x]$ i $\{x\}$ oznaczają odpowiednio część całkowitą i ułamkową liczby x. Przykładowo początkowe 8 elementów tej tablicy powinno zostać wypełnione wartościami (dla ułatwienia zapisanymi szesnastkowo):

```
243F6A88
85A308D3
13198A2E
03707344
A4093822
299F31D0
082EFA98
EC4E6C89
```

Parametr `pidx` jest wskaźnikiem na indeks elementu w tablicy `ppi`. Funkcja `pix` działa w pętli. W każdej iteracji wykonuje atomową operację pobrania wartości tego indeksu i zwiększenia jego wartości o jeden. Przyjmijmy, że pobrana (przed zwiększeniem) wartość wynosi `m`. Jeśli `m` jest mniejsze niż wartość parametru `max`, funkcja oblicza bity rozwinięcia o numerach od `32m+1` do `32m+32` i umieszcza wynik w `ppi[m]`. Jeśli `m` jest większe lub równe `max` funkcja kończy działanie.

Jednocześnie może być uruchomionych wiele instancji funkcji `pix`, każda w osobnym wątku. Instancje te mają dostęp do wspólnej globalnej zmiennej wskazywanej przez parametr `pidx` oraz globalnej tablicy `ppi` i dzielą między siebie pracę, używając tych globalnych zmiennych w wyżej opisany sposób.

Dodatkowo funkcja `pix` powinna zaraz na początku swojego działania oraz tuż przed zakończeniem wywołać funkcję `pixtime`, która będzie zaimplementowana w języku C na przykład tak:

```
void pixtime(uint64_t clock_tick) {
  fprintf(stderr, "%016lX\n", clock_tick);
}
```

i przekazać jej jako wartość parametru `clock_tick` liczbę cykli procesora, jaka upłynęła od jego uruchomienia, uzyskaną za pomocą rozkazu `rdtsc`.

Rozwiązanie ma używać wyłącznie arytmetyki stałopozycyjnej.

Wskazówka 1: Tu jest krótkie wyjaśnienie, jak użyć formuły BBP, aby obliczyć cyfrę `n + 1` części ułamkowej szesnastkowego rozwinięcia π. W tym zadaniu należy przyjąć `n = 8m` i obliczyć w jednym sumowaniu od razu 8 cyfr rozwinięcia szesnastkowego, czyli 32 bity. Sumowanie kończymy, gdy kolejny składnik jest mniejszy niż 2^{-64}.

Wskazówka 2: Obliczenia dobrze jest prowadzić na liczbach z przedziału [0; 1) i przy każdym sumowaniu pomijać część całkowitą. Wtedy liczba całkowita x reprezentuje liczbę ułamkową $x2^{-64}$.

Wskazówka 3: Wydaje się, że dla efektywnego rozwiązania tego zadania potrzebne jest zrozumienie działania rozkazów `mul` i `div`, a w szczególności tego, jak za pomocą `div` uzyskać ułamkową część ilorazu, oraz zrozumienie zadania A1 z pierwszych zajęć o asemblerze, a w szczególności tego, jak gcc realizuje dzielenie na przykład przez 3.

Wskazówka 4: Rozkaz `div` podnosi wyjątek przy dzieleniu przez zero lub gdy iloraz jest za duży. Linux wypisuje wówczas komunikat „Błąd w obliczeniach zmiennoprzecinkowych”, mimo że obliczenia nie są zmiennoprzecinkowe.

Wskazówka 5: Warto zapoznać się z kilkoma rozkazami, które nie pojawiły się w dotychczasowych przykładach: `cmov`, `bsr`, `shld`, a także z takimi, które się już pojawiły: `lea`, `xadd`.

Oceniane będą **poprawność** i **szybkość** działania programu, **zajętość pamięci** (rozmiary poszczególnych sekcji), **przestrzeganie konwencji ABI** oraz **styl** kodowania. Tradycyjny styl programowania w asemblerze polega na rozpoczynaniu etykiet od pierwszej kolumny, mnemoników od dziewiątej kolumny, a listy argumentów od siedemnastej kolumny. Inny akceptowalny styl prezentowany jest w przykładach pokazywanych na zajęciach. Kod powinien być dobrze skomentowany, co oznacza między innymi, że każda procedura powinna być opatrzona informacją, co robi, jak przekazywane są do niej parametry, jak przekazywany jest jej wynik, jakie rejestry modyfikuje. To samo dotyczy makr. Komentarza wymagają także wszystkie kluczowe lub nietrywialne linie wewnątrz procedur lub makr. W przypadku asemblera nie jest przesadą komentowania prawie każdej linii kodu, ale należy jak ognia unikać komentarzy typu „zwiększenie wartości rejestru rax o 1”.

Jako rozwiązanie należy oddać plik `pix.asm`. Program będzie kompilowany i linkowany z programem napisanym w języku C poleceniami:

```
nasm -f elf64 -w+all -w+error -o pix.o pix.asm
gcc -std=c11 -Wall -Wextra -O2 -o pix *.c pix.o
```

Od strony języka C będziemy korzystać z następującego pliku nagłówkowego `pix.h`:

```
#ifndef PIX_H
#define PIX_H

#include <stdint.h>

void pix(uint32_t *ppi, uint64_t *pidx, uint64_t max);
void pixtime(uint64_t clock_tick);

#endif
```