Damian Koper, 241292
SR-TN-17
Wrocław, 08.05.2019r.

Lab 3: FPU, konwencje wywołania, analiza stosu

Cel ćwiczenia

- Złożone działania wykorzystujące FPU - całka oznaczona
- Udostępnienie prostej funkcji na poziomie interfejsu języka C
  - Analiza ramki stosu
  - Konwencja wywołania
- Wyświetlenie śladu stosu
- Pomiar czasu wykonania fragmentu kodu

FPU

FPU jest układem wspomagającym obliczenia, specjalizującym się w obliczeniach na liczbach w formatach stałoprzecinkowych, zmiennoprzecinkowych oraz *packed BCD*. Wyposażony jest w 8 80-bitowych rejestrów obsługiwanych, które tworzą stos. Za śledzenie jego szczytu odpowiadają trzy bity `TOP` słowa statusu FPU.

```
(gdb) info float
  R7: Empty   0x00000000000000000000
  R6: Empty   0x00000000000000000000
  ...
=>R0: Empty   0x00000000000000000000
```

Podczas wstawiania wartości na górę stosu `TOP = (TOP+1) mod 8` i w miejsce `TOP` wstawiana jest nowa wartość.

```
(gdb) info float # Po dodaniu liczby 100
Output:
=>R7: Valid   0x4005c800000000000000 +100
  R6: Empty   0x00000000000000000000
  ...
  R0: Empty   0x00000000000000000000
```

Stan i kontrola działania FPU

FPU posiada trzy specjalne rejestry przechowujące jego status i określające sposób jego działania.

Status word

16-bitowe słowo statusu FPU określające aktualny jego stan. Wyróżnione są tam 4 bity flag C0-C3, bit zajętości FPU, wspomniany licznik `TOP`, bit *Interrupt Request*, oraz 7 bitów wyjątków, które moga zostać wyrzucone podczas pracy FPU.

Control word

Control word o długości 16-bit zawiera 12-bitów ustawiające sposób działania FPU. Ustawia sposób interpretowania nieskończoności, tryb zaokrąglenia, precyzję oraz bity określające obsługę wyjątków.

Tag word

Na każde 2 bity tego słowa 16-bitowego przypada numer określający zawartość poszczególnych rejestrów stosu FPU.

ONP - Odwrócona notacja polska

Podczas projektowania algorytmów przydatne okazuje się odpowiednie wyrażenie złożonej operacji arytmetycznej za pomocą jej składowych. Odwrócone notacja polska jest sposobem zapisu działania, gdzie jego operator znajduje się za jego operandami (zapis postfixowy).

```
      2 + 2 -> 2 2 +
  2 + 2 * 2 -> 2 2 2 * +
(2 + 2) * 2 -> 2 2 + 2 *
```

Jednoznacznie określa to kolejność wykonywania działań. Algorytmy wykorzystujące ONP wykorzystują stos, co idealnie sprawdza się w przypadku problemu projektowania algorytmów wykonywanych przez FPU.

Konwencja wywołania `cdecl`

W tej konwencji wywołania fukncji, wykorzystywanej przez większość kompilatorów języka `C`, strona wywołująca jest odpowiedzialna za przekazanie argumentów poprzez stos oraz ich późniejsze z niego zdjęcie. Stałoprzecinkowy wynik funkcji przekazywany jest w rejestrze `eax`, a zmiennoprzecinkowy w `st(0)` FPU. Reszta rejestrów `st(x)` musi być pusta. W tej konwencji funkcja wywoływana jest odpowiedzialna za zachowanie stanów wszystkich rejestrów oprócz `eax`, `ecx` i `edx`.

Programy

- Kody programów `integral`, `my_sinDemo`, `stacktraceDemo`: https://github.com/damiankoper/OiakLab/tree/master/lab_4

Całka sin(x) oraz log10(x)

Program `integral` liczy dwie całki oznaczone: `sin(x)` oraz `log10(x)` pobierając dane wejściowe(A - początek, B - koniec i ilośc przedziałów) ze standardowego wejścia i wyświetlając wynik na standartowe wyjście. Obliczenia wykonywane są mnetodą trapezów i wykorzystują format 64-bitowy `double` (FLD**L**, Fxxx**L**).

W pierwszej kolejności obliczana jest szerokość przedziału (wysokość trapezu):

```
fildl numOfIntervals
fldl b
fldl a
```

```
fsubp
fdivp

fstpl interval # zapisanie przedziału do pamięci
```

co stosując ONP można napisać według kolejności instrukcji FPU:

```
a - początek, b - koniec, i - ilość przedziałów, x - wysokośc trapezu
(a-b) / i = x
ONP: i b a - / (zapisz i zdejmij ze stosu)
```

Całka sin(x)

Na jedną iteracje liczenia całki składają się dwie operacje:

1. Obliczenia pola trapezu i dodanie go do wyniku
2. Przesunięcie punktu `a` o wysokość trapezu

Przed wejściem do pętli należy włożyć na stos FPU 0 za pomoca instrukcji `FLDZ`. To miejsce na stosie będzie sumować pola kolejnych trapezów. Kolejny raz można rozpisać operacje używając ONP:

```
a - początek, b - koniec, x - wysokośc trapezu, w - wynik (już na stosie)

w += (x * (sin(a) + sin(a + x))) / 2
ONP: w x a (sin) a x + (sin) + * 2 / +

a += a + x
ONP: a x + (zapisz a i zdejmij ze stosu)
```

Wykonując pętlę podaną ilość razy na koniec otrzymamy wartość obliczonej całki w `st(0)`. Aby zapis ONP pokrywał się z działaniem FPU należy użyć instrukcji zdejmujących jeden operand ze stosu - instrukcje z literą `P` (`POOP`).

Całka log10(x)

Całka z logarytmu obliczana jest analogicznie. Istotną różnicą jest obliczenie samego logarytmu o podstawie 10 (zamiast *sin(x)*). Architektura FPU nie udostępnia instrukcji, która bezpośrednio może policzyć owy logarytm. Skorzystać trzeba z zależności:

```
log10(x) = log2(x) / log2(10)
```

oraz:

```
loga(b) = 1 / logb(a)
```

czyli:

```
log10(x) = log2(x) log10(2)
```

Wykorzystując instrukcję `FYL2X` możemy obliczyć wynik działania `y * log2(x)`. Mając zatem na uwadze kolejnośc operandów możemy zapisać następujące działanie:

```
a - początek, b - koniec, x - wysokośc trapezu, w - wynik (już na stosie)

w += (x * (log10(a) + log10(a + x))) / 2
w += (x * (log10(2) * log2(a) + log10(2) * log2(a + x))) / 2

ONP: w x log10(2) a (fyl2x) log10(2) a x + (fyl2x) + * 2 / +
```

Stała `log10(2)` ładowana jest za pomocą rozkazu `FLDL2G`.

Pomiar czasu wykonania

Program `integral` odpowiedzialny jest również za pomiar czasu wykonania fragmentu kodu. Elementem pomiaru jest procedura obliczania całki `sin(x)`. Do pomiaru użyto rozkazu `RDTSC`. Rozkaz ten wypełnia rejestry `edx:eax` zawartością rejestru `TSC`, którego zawartość inkrementowana jest po każdym cyklu procesora. Przy długości 64-bit i maksymalnej częstotliwości taktowania procesora *3.0 GHz* rejestr ten nie ulegnie przepełnieniu przez ok. 200 lat. Przez swoją dokładność może być on użyty do ataków czasowych, na których opierają się między innymi ataki Meltdown i Spectre.

`RDTSC` nie jest instrukcją serializującą - procesor, poprzez spekulatywne wykonywanie kodu, bądź pipielining, może odczytać zawartośc rejestru `TSC` zanim wykona poprzednie instrukcje. Aby temu zapobiec trzeba poprzedzić rozkaz `RDTCS` rozkazem `CPUID`, który jest instrukcją serializującą. Alternatywą jest użycie instrukcji `RDTSCP`, która łączy odczyt `TSC` i serializację.

```
# Pomiar czasu
xor %eax, %eax
cpuid
rdtsc
movl %eax, timeTSC1
movl %edx, timeTSC1 + 4
```

Wnioski

Literatura

1. Wikibooks x86 Assembly - https://en.wikibooks.org/wiki/X86_Assembly
2. Laboratorium AK –ATT asembler (LINUX) - http://zak.ict.pwr.wroc.pl/materials/architektura/laboratorium%20AK2/Linux-AK2-lab-2018%20May.pdf
3. University of Virginia Computer Science - x86 Assembly Guide http://www.cs.virginia.edu/~evans/cs216/guides/x86.html
4. Prezentacja do wykładu
5. Dokumentacja GDB - https://sourceware.org/gdb/current/onlinedocs/gdb/
6. `gdb help`
7. FPU - http://www.website.masmforum.com/tutorials/fptute/fpuchap1.htm