Michał Surmacki 241130
TN ŚR. $17^{05} - 20^{05}$

Wrocław, dn. 8 maja 2019

Laboratorium 4
OiAK, tydzień 14/15

sprawozdanie z laboratorium przedmiotu „Organizacja i Architektura Komputerów”

Rok akad. 2018/2019, kierunek: INF

PROWADZĄCY:
dr inż. Piotr Patronik

Spis treści

1 Cel ćwiczenia **2**

2 Przebieg ćwiczenia **2**
2.1 Dodawanie i odejmowanie liczb zmiennoprzecinkowych w formacie pojedynczej precyzji - single 2
2.2 Mnożenie liczb zmiennoprzecinkowych 3
2.3 Dzielenie liczb zmiennoprzecinkowych 4
2.4 Jednostka zmiennoprzecinkowa - Floating Point Unit - FPU 5
2.5 Wyświetlanie wartości dziesiętnej liczby zmiennoprzecinkowej 5

3 Podsumowanie i wnioski **6**

Bibliografia **6**

4 Uwagi **6**

1 Cel ćwiczenia

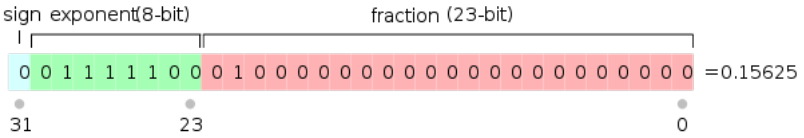
Celem ćwiczenia było wykonanie programów realizujących operacje arytmetyczne na liczbach zmiennoprzecinkowych (bez użycia rozkazów operujących na liczbach zmiennoprzecinkowych). Należało również napisać program, który drukuje na standardowe wyjście wartość liczby zmiennoprzecinkowej w podstawie dziesiętnej.
Zapoznano się też z:

- architekturą jednostki zmiennoprzecinkowej,
- powtórzono i zaznajomiono się ze specjalnymi kodami liczb zmiennoprzecinkowych w notacji szestnastkowej (± 0, $\pm INF$, $\pm NaN$, ± 1.0).

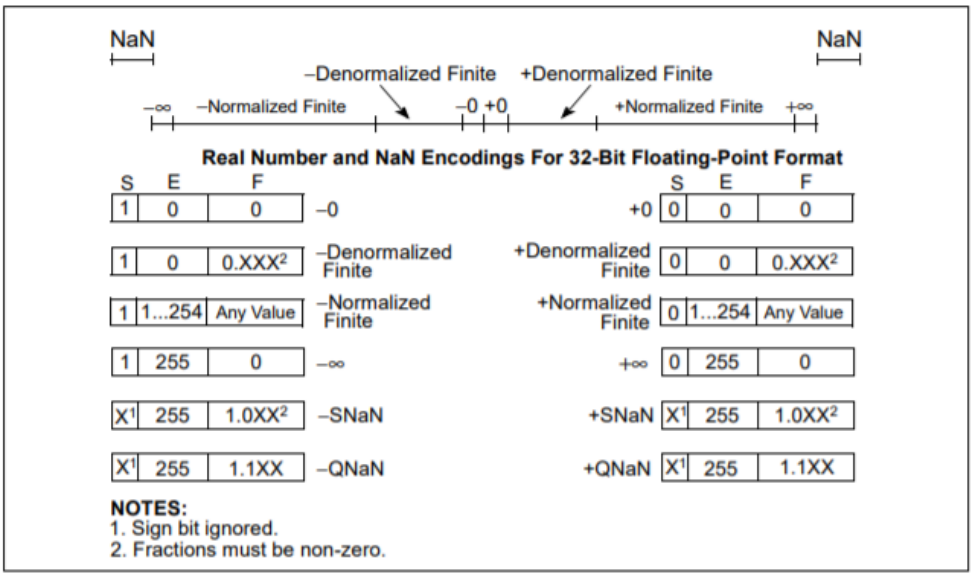
2 Przebieg ćwiczenia

2.1 Dodawanie i odejmowanie liczb zmiennoprzecinkowych w formacie pojedynczej precyzji - single

Wykonanie pierwszego programu rozpoczęto od przypomnienia zasad działań wykonywanych na liczbach zmiennoprzecinkowych pojedynczej precyzji. Liczba pojedynczej precyzji przechowywana jest w pamięci na 32 bitach, z czego pierwszy bit odpowiedzialny jest za znak liczby, osiem następnych definiują wykładnik w kodzie +N (+127 dla formatu single), pozostałe bity tworzą mantysę. Poniższa grafika ilustruje jak wygląda liczba zmiennoprzecinkowa w formacie single.



Zamieszczam również grafikę z dokumentacji FPU ilustrującą kodowanie liczb zmiennoprzecinkowych formatu single:



Gdy wykładnik jest równy 00000000, to liczbę nazywamy zdenormalizowaną, i przyjmujemy, że nie posiada ona dodatkowej jedynki przed przecinkiem. W przypadku liczb niezdenormalizowanych, przed przecinkiem występuje jedynka. Należy pamiętać również o 'ukrytej jedynce', która występuje kiedy wykładnik jest postaci 00000000 (wtedy do obliczeń przyjmuje się wykładnik równy 00000001).

Program rozpoczęto od wczytania ze standardowego wejścia 2 liczb, które zostały wczytane do pamięci programu za pomocą funkcji bibliotecznej **scanf**. Aby liczby zostały zapisane w pamięci jako liczny zmiennoprzecinkowe wykorzystano odpowiednie formatowanie podczas wywoływania funkcji **scanf** - "%f". Wartości liczb zostały zapisane w pamięci przy pomocy etykiet `input1` oraz `input2`. Wykorzystując **gdb**, po wczytaniu liczb funkcją **scanf** upewniono się, że są one poprawinie wczytywane.

Operacja dodawania liczb zmiennoprzecinkowych polega na sprowadzeniu liczb do wspólnego wykładnika, następnie dodaniu/odjęciu mantys (jeśli podczas wykonywanie operacji otrzymamy liczbę ujemną należy obliczyć wartość bezwzględną liczby, ja umieścić w mantysie, a znak zakodować bitem znaku) i znormalizowaniu liczby (przesunięciu przecinka z jednoczesnym dodaniem lub odjęciem odpowiedniej wartości do wykładnika).

Za pomocą rozkazów wykonywujących operacje logiczne (`and, or`) wyogrębniano potrzebne części liczb zmiennoprzecinkowych - bit znaku, wykładnik, mantysa. Dokonano sprowadzenia obydwu liczb do wspólnego wykładnika, następnie za pomocą przesunięć bitowych (`shl, shr`) ospowiednio sformatowano mantysy liczb. Następnie na sformatowanych mantysach można wykonać operację arytmetyczną.

Napisano funkcję, która dokonuje przesunięcia bitowego i zaokrągla **symetrycznie do parzystej**, jeśli wystąpi taka potrzeba (wszystkie bity nie mieszczą się na 23 pozycjach). Aby móc wykorzystać napisaną funkcję w programie, zlinkowano odpowiednio program.

Wykonując operacje na liczbach zmiennoprzecinkowych, może dojść do sytuacji, że uzyskany wynik będzie kodem specjalnym, zarezerwowanym. Tabela kodów zarezerwowanych wraz z opisem kodów znajduje się poniżej:

E (wykładnik)	f (mantysa)	wartość
0 ... 0	0 ... 0	$\pm$ 0
0 ... 0	f $\neq$ 0	liczba zdenormalizowana
1 ... 1	0 ... 0	$\pm$ inf (nieskończoność)
1 ... 1	f $\neq$ 0	NaN (Not a Number)

Operacje **odejmowania** liczb zmiennoprzecinkowych, oparto na dodawaniu ponieważ z zależności $A - B = A + (-B)$ wynika, można wykonać operację odejmowania, dodając liczbę przeciwną (liczbę przeciwną uzyskano stosując rozkaz `neg`).

2.2 Mnożenie liczb zmiennoprzecinkowych

Podczas mnożenia liczb zmiennoprzecinkowych bit znaku wyniku możemy określić odrazu na podstawie dwóch bitów znaku operandów. Jeśli są one zgodne - wynikiem mnożenia będzie liczba dodatnia, jeśli natomiast są przeciwne to wynikiem mnożenia będzie liczba ujemna. Aby ominąć niepotrzebnie wykonywane działania, na początku sprawdzono, czy któraś z liczb nie odpowiada kodowi specjalnemu ± 0. Jeśli odpowiada, wynikiem mnożenia będzie 0 ze znakiem określonym zgodnie z zasadami napisanymi powyżej.

Tak jak w przypadku programu wykonującego dodawanie/odejmowanie liczb zmiennoprzecinkowych zostały wykorzystane rozkazy `and, or`, dzięki którym udało się wyodrębnić poszczególne części liczby. Przy wykonywaniu mnożenia, tak samo jak przy dodawaniu i odejmowaniu, obowiązują takie same zasady dla liczb zdenormalizowanych - "ukryta jedynka", zero przed przecinkiem, oraz takie same zasady dotyczące kodów specjalnych. Wykonując mnożenie nie ma potrzeby korygować mantys, jak w przypadku dodawania/odejmowania, dlatego po ich wyodrębnieniu i dodaniu odpowiedniej liczby przed przecinkiem (na pozycji 24) użyto rozkazu `mull` aby pomnożyć je przez siebie. Wykładniki przy operacji mnożenia należy dodać do siebie, jednak trzeba pamiętać, że w formacie single wykładniki są zapisane w kodnie +N (+127). Aby dodać wartości wykładników należy odjąć obciążenie, następnie dodać do siebie uzyskane wartości i dodać wcześniej odjęte obciążenie +127.

Po wymnożeniu mantys i obliczeniu wykładnika należy znormalizować uzyskaną mantysę. Ponieważ wynik mnożenia mantys z dodaną liczbą przed przecinkiem (24-bity), będzie długości 48 bitów, należy sprawdzać od lewej pozycję wystąpienia pierwszej jedynki, następnie przesunąć w prawo (`shr`) o wyznaczoną wartość (pozycja wystąpienia pierwszej jedynki) - (liczba pozycji od wystąpienia pierwszej jedynki do pozycji 23), zaokrąglając utracone bity - realizuje to napisana funkcja `shiftRight`, która została również użyta podczas operacji dodawania/odejmowania (identyczne przesuwanie w lewo realizuje funkcja `shiftLeft`, wtedy obliczona wartość przesunięcia jest ujemna co oznacza, że należy przesunąć liczbę w lewo o wartość bezwzględną tej liczby). Następnie jak w przypadku dodawania/odemnowania, korygowany jest wykładnik - dodawana jest do niego wartość kiedy przesuwamy za pomocą funkcji `shiftRight` a odejmowania kiedy przesuwamy za pomocą dunkcji `shiftLeft`.

Na samym końcu poszczególne części są łączone za pomocą operacji `or`.

Należy również mieć na uwadze czy w wyniku wykonywanych zadań liczba wynikowa nie przekroczyła zakresu, w takiej sytuacji występuje sytuacja, że wynikiem jest ± inf.

2.3 Dzielenie liczb zmiennoprzecinkowych

Tak jak w przypadku mnożenia, na początku sprawdzane są operandy, w celu ustalenia czy wynikiem dzielenia nie będzie liczba o kodzie specjalnym. Dlatego dla przypadków opisanych w poniższej tabeli odrazu możemy podać wartość wyniku.

dzielna [D]	dzielnik [d]	wynik
0	d $\neq$ 0	± 0
0	0	NaN
D $\neq$ 0	0	± inf

Znak wyniku będzie dodatni jeśli znaki operandów są zgodne, oraz ujemny jeśli znaki operandów są przeciwne. Przeciwnie niż w przypadku mnożenia, aby uzyskać wykładnik liczby wynikowej należy **odjąć** od siebie wykładniki. Aby to zrobić należy odjąć od operandów obciążenie (+127), odjąć od siebie operandy, następnie dodać obciążenie do wyniku działania. Mantysy liczb należy podzielić, następnie znormalizować liczbę przesuwając odpowiednio przecinek jednocześnie dodając lub odejmując wartość przesunięcia do obliczonego wcześniej wykładnika. W wyniku przekroczenia zakresu występuje ± 0.

2.4 Jednostka zmiennoprzecinkowa - Floating Point Unit - FPU

Jednostka zmiennoprzecinkowa jest specjalnie zaprojektowana do przeprowadzania działań na liczbach zmiennoprzecinkowych. W przeszłości FPU była implementowana jako oddzielny koprocesor (pierwsze pojawiły się w latach 70), który wspomagał pracę procesora wykonując operacja na liczbach zmiennoprzecinkowych (IBM PC posiadał socket na koprocesor Inter 8087), jednak w dzisiejszych czasach FPU jest zintegrowaną częścią procesora. Floating Point Unit posiada 8 rejestrów od `st0` do `st7`, które tworzą stos - liczby zmiennoprzecinkowe mogą przyjmować formaty single (32 bity) oraz double (64 bity), jednak aby zredukować błędy zaokrągleń i tracenia bitów, rejestry FPU mają rozmiar 80 bitów. Rozkazy dla architektury: IA-32 Intel®Architecture są opisane w dokumentacji,

- Architecture Software Developer's Manual Volume 1: Basic Architecture,

strona 5-9. Natomiast instrukcjie korzystania z tych rozkazów znajdują się w

- Architecture Software Developer's Manual Volume 2: Instruction Set Reference.

Większość konwencji przyjmuje, że wyniki działań na liczbach zmiennoprzecinkowych są przechowywane w rejestrze st0.

2.5 Wyświetlanie wartości dziesiętnej liczby zmiennoprzecinkowej

Z racji tego, że programy pisane są w architekturze 32 bitowej, a wyświetlanie liczby zmiennoprzecinkowej biblioteczną funkcją **printf** z opcją - `"%f"` wymaga, by liczba była w formacie double (64 bity) pojawia się błędne wyświetlenie liczby przy próbie podania jako argument funkcji liczby 32-bitowej w formacie single. Dlatego aby przekonwertować liczbę do formatu double posłóżono się stosem rejestrów FPU, użyto rozkazu `fld` (Load Floating Point Value), który wrzuca liczbę na stos rejestrów, dokonując kopiowania do rejestru st0. Rokaz `fld` konwertuje wrzucaną na stos liczbę do formatu double. Następnie dokonywana jest operacja przesunięcia wskaźnika stosu o 2 słowa - textttsubl $8, %esp, w celu zrobienia miejsca liczbie formatu double zapisanej na 64-bitach. Do wczytania we wcześniej zrobione miejsce liczby zmiennoprzecinkowej, użyto rozkazu - `fst` - (Store Floating Point Value), który kopiuje zawartość rejestru st0 do miejsca wskazanego przez operand w rozkazie, który w tym przypadku jest wskaźnikiem stosu - `fst (%esp)`. Następnie na stos wrzucany jest format funkcji **printf** - `"%f"` i funkcja jest wywoływana. Na ekranie widzimy liczbę zmiennoprzecinkową wyświetloną w postaci dziesiętnej.

3 Podsumowanie i wnioski

Wykonując ćwiczenia zadane na laboratoriach, zostały powtórzone operacje na liczbach zmiennoprzecinkowych. Operacje zostały zaimplementowane w języku assemblera, z wyłączeniem rozkazów przeznaczonych do działań na liczbach zmiennoprzecinkowych, co poskutkowało lepszym zrozumieniem zasad wykonywania tych działań. Przećwiczono również konwersje liczb binarnych na liczby w formacie szestnastkowym i odwrotnie, aby za pomocą operacji logicznych wyodrębniać części liczby zmiennoprzecinkowej. Programy zostały przetestowane narzędziem: **gdb**, w szczególności przydatna była opcja komendy **x - /f**, która wyświetla zawartość wskazanej pamięci jako liczbę zmiennoprzecinkową. W programie użyto napisanych w osobnych plikach funkcji. Pliki z funkcjami zlinkowano z programem głównym. Pracę nad programami zautomatyzowały napisane pliki `Makefile`.

Literatura

[1] Jonathan Bartlett, *Programming from the Ground Up*, 2004.

[2] IA-32 Intel® Architecture Software Developer's Manual Volume 1: Basic Architecture,

[3] IA-32 Intel® Architecture Software Developer's Manual Volume 2: Instruction Set Reference,

[4] Randall Hyde, *THE ART OF ASSEMBLY LANGUAGE, 2ND EDITION* 2010

[5] Intel®, *Floating point unit*

4 Uwagi

Listingi kodów nie zostały zamieszczone, natomiast podaję link do repozytorium na githubie, gdzie znajdować się będą kody wykonanych programów. Programy zostały przetestowane i działają poprawnie, zgodnie z początkowymi założeniami.

https://github.com/MichalSurmacki/OiAK-Lab/tree/master/Lab3.