Architektura komputerów – laboratorium 14 (2 godziny)
Temat: Operacje na liczbach zmiennoprzecinkowych

Do obsługi liczb zmiennoprzecinkowych w asemblerze MIPS przeznaczony jest Koprocesor 1. Zawiera on 32 rejestry po 32 bity każdy. Rejestry numerowane są odpowiednio $f0, $f1, …$f31. W przypadku liczb zmiennoprzecinkowych rejestr $f0 nie jest rejestrem specjalnym. Wszystkie operacje na liczbach rzeczywistych wykorzystują te rejestry (niedozwolone jest korzystanie z rejestrów ogólnego przeznaczenia).

Działania na liczbach rzeczywistych mogą być wykonywane na liczbach pojedynczej precyzji (single precision) – wtedy korzysta się z wyżej wymienionych rejestrów – lub na liczbach podwójnej precyzji (double precision). W tym drugim przypadku wartości przechowywane są w parach rejestrów, np. $f0 i $f1, $2 i $f3, $f4 i $f5 itd. lub w tzw. rejestrach zmiennoprzecinkowych podwójnej precyzji, np. $f0, $f2, $f4 itd. W parach rejestrów adresowany jest pierwszy rejestr z pary ($f0, $f2, $f4 itd.). Poszczególne rejestry mają swoje ustalone przeznaczenie, które przedstawiono w tabeli poniżej.

Nazwa rejestru	Numer rejestru	Zastosowanie
$fv0 - $fv1	$f0, $f2	Zwraca wartości z podprogramu
$ft0 - $ft3	$f4, $f6, $f8, $f10	Przechowuje zmienne tymczasowe
$fa0 - $fa1	$f12, $f14	Przechowuje argumenty funkcji
$ft4 - $ft8	$f16, $f18	Przechowuje zmienne tymczasowe
$fs0 - $fs5	$f20, $f22, $f24, $f26, $f28, $f30	Przechowuje zapisane zmienne

Instrukcje operacji arytmetycznych dla liczb zmiennoprzecinkowych różnią się w zależności od tego czy przeprowadzane są na liczbach pojedynczej- czy podwójnej precyzji. Podstawowe instrukcje operacji arytmetycznych realizowane na liczbach pojedynczej precyzji:

- add.s $f0, $f1, $f3 – suma zmiennoprzecinkowa pojedynczej precyzji: załaduj do $f0 wartość sumy liczb z rejestrów $f1 i $f3
- sub.s $f0, $f1, $f3 – różnica zmiennoprzecinkowa pojedynczej precyzji: załaduj do $f0 wartość różnicy liczb z rejestrów $f1 i $f3
- mul.s $f0, $f1, $f3 – iloczyn zmiennoprzecinkowy pojedynczej precyzji: załaduj do $f0 wartość iloczynu liczb z rejestrów $f1 i $f3
- div.s $f0, $f1, $f3 – iloraz zmiennoprzecinkowy pojedynczej precyzji: załaduj do $f0 wynik dzielenia liczby zawartej w $f1 przez liczbę z $f3

Analogiczne instrukcje dla liczb zmiennoprzecinkowych podwójnej precyzji przedstawiają się następująco:

- add.d $f0, $f1, $f3 – suma zmiennoprzecinkowa podwójnej precyzji: załaduj do $f0 wartość sumy liczb z rejestrów $f1 i $f3

- sub.d $f0, $f1, $f3 – różnica zmiennoprzecinkowa podwójnej precyzji: załaduj do $f0 wartość różnicy liczb z rejestrów $f1 i $f3
- mul.d $f0, $f1, $f3 – iloczyn zmiennoprzecinkowy podwójnej precyzji: załaduj do $f0 wartość iloczynu liczb z rejestrów $f1 i $f3
- div.d $f0, $f1, $f3 – iloraz zmiennoprzecinkowy podwójnej precyzji: załaduj do $f0 wynik dzielenia liczby zawartej w $f1 przez liczbę z $f3

Dla liczb zmiennoprzecinkowych różne są również instrukcje przesyłania danych:

- lwc1 $ft1, 42($s1) – załaduj słowo do Koprocesora 1: załaduj do rejestru $ft1 32 bitową wartość pojedynczej precyzji zapisaną w pamięci o adresie $s1+42
- swc1 $fs2, 17($sp) – przechowaj słowo z Koprocesora 1: przechowaj 32 bitową wartość pojedynczej precyzji zawartą w rejestrze $fs2 w pamięci o adresie $sp + 17
- ldc1 $ft2, 42($s1) – załaduj słowo zmiennoprzecinkowe podwójnej precyzji do Koprocesora 1: załaduj do rejestru $f2 64 bitową wartość podwójnej precyzji zapisaną w pamięci o adresie $s1+42
- sdc1 $fs2, 17($sp) – przechowaj słowo zmiennoprzecinkowe podwójnej precyzji z Koprocesora 1: przechowaj 64 bitową wartość podwójnej precyzji zawartą w rejestrze $fs2 w pamięci o adresie $sp + 17

Przykład wczytywania liczb zmiennopozycyjnych:

```
.data
.text
floats:         .float  -1.827, 3.14
(...)
                lwc1 $f0, floats

                lwc1 $f1, floats + 4
```

Dla liczb zmiennopozycyjnych zdefiniowano osobne instrukcje warunkowe oraz instrukcje skoku:

- Równa:
  - Dla liczb pojedynczej precyzji: **c.eq.s $f0, $f1** – jeśli wartość $f0 jest równa $f1, to ustaw flagę 0 Koprocesora 1 na TRUE, w przeciwnym przypadku na FALSE
  - Dla liczb podwójnej precyzji: **c.eq.d $f2, $f4** – jeśli wartość $f2 jest równa $f4, to ustaw flagę 0 Koprocesora 1 na TRUE, w przeciwnym przypadku na FALSE
- Mniejsza niż:
  - Dla liczb pojedynczej precyzji: **c.lt.s $f0, $f1** – jeśli wartość $f0 jest mniejsza niż $f1, to ustaw flagę 0 Koprocesora 1 na TRUE, w przeciwnym przypadku na FALSE
  - Dla liczb podwójnej precyzji: **c.lt.d $f2, $f4** – jeśli wartość $f2 jest mniejsza niż $f4, to ustaw flagę 0 Koprocesora 1 na TRUE, w przeciwnym przypadku na FALSE
- Mniejsza lub równa:
  - Dla liczb pojedynczej precyzji: **c.le.s $f0, $f1** – jeśli wartość $f0 jest mniejsza lub równa $f1, to ustaw flagę 0 Koprocesora 1 na TRUE, w przeciwnym przypadku na FALSE

  - Dla liczb podwójnej precyzji: **c.le.d $f2, $f4** – jeśli wartość $f2 jest mniejsza lub równa $f4, to ustaw flagę 0 Koprocesora 1 na TRUE, w przeciwnym przypadku na FALSE
- Instrukcja skoku: **bc1t etykieta1** – jeśli flaga 0 Koprocesora 1 jest ustawiona na TRUE (wartość „1“), to przejdź do instrukcji o adresie określonym etykietą 1
- Instrukcja skoku: **bc1f etykieta 2**- jeśli flaga 0 Koprocesora 1 jest ustawiona na FALSE (wartość „0“), to przejdź do instrukcji o adresie określonym etykietą 2.

Przykład:

```
(...)
        c.eq.s  $f1, $f2
        bc1t    Etykieta1
        bc1f    Etykieta2
```

Sekwencja instrukcji powyżej mówi, że jeśli rejestry $f1 i $f2 przechowują tę samą wartość, to skocz do Etykieta1, w przeciwnym wypadku do Etykieta2. Wynik porównania w specjalnym jednobitowym rejestrze – flaga 0 Koprocesowa 1 - niedostępnym dla programisty.

Pełny zestaw instrukcji dla liczb zmiennoprzecinkowych pojedynczej- i podwójnej precyzji zawiera zakładka Pomoc symulatora MARS.

Ćwiczenia: Implementacja programu z wykorzystaniem operacji na liczbach zmiennoprzecinkowych

Proszę zaimplementować kalkulator, który będzie operował na liczbach zmiennoprzecinkowych (floating point) podwójnej precyzji.

Program na wejściu pyta, jaką operację ma wykonać:

1. dodawanie,
2. odejmowanie,
3. mnożenie,
4. dzielenie

a następnie prosi o podanie argumentów.

Na wyjściu program drukuje wynik działania i pyta czy kontynuować obliczenia.

Proszę zwrócić uwagę na dzielenie przez zero. Program powinien również sygnalizować brak lub nieprawidłowość podanych przez użytkownika argumentów (np. litera zamiast cyfry, kropka czy przecinek do oddzielenia części ułamkowej, liczba cyfr po przecinku większa niż dopuszczalna itp.). Czy i w jakich okolicznościach wystąpi przepełnienie?