**ACS|Индивидуальное Домашнее Задание 2**
**Горбачева Маргарита Валерьевна|БПИ-245**

**Задание:**
Разработать программы на языке Ассемблера процесса RISC-V, с использованием команд арифметического сопроцессора, выполняемые в симуляторе RARS. Разработанные программы должны принимать числа в допустимом диапазоне. Например, нужно учитывать области определения и допустимых значений, если это связано с условием задачи.

**Вариант 22:**
Разработать программу вычисления числа π с точностью не хуже 0,1% посредством дзета-функции Римана.

**Файл программы хранится на Github по ссылке:**
https://github.com/Misss-Lacoste/HSE-FCS-SE-2-course/blob/main/ACS/assembly/IHW2/riscv1.asm

**Общий алгоритм решения:**
Математическая основа: используется тождество Эйлера для дзета( $\zeta$ )-функции Римана. Данная формула имеет вид:

$$\zeta(s) = \lim_{n\to\infty}\left(1+\frac{1}{2^s}+\frac{1}{3^s}+\ldots+\frac{1}{n^s}\right), \mathrm{Re}(s) > 1$$

, что эквивалентно:

$$\zeta(s) = \sum_{i=1}^{\infty}\frac{1}{i^s}$$

. Таким образом, можно представить числовой ряд = $Pi^2/6$.

Выразив Pi, получим: $\pi = \sqrt{(6 \times \Sigma(1/k^2))}$, где k=i.

**Подробный алгоритм:**
Основная функция программы - compute_pi_zeta. Ее основные шаги:

1) Запрашиваем у пользователя число - количество итераций (число должно быть integer)
2) Инициализируем сумму нулем, берем k = 1.

3) Далее реализуем цикл вычисления суммы ряда: преобразуем k в double, инициализируем переменную новую переменную, которой присваиваем значение $k^2$.
4) Далее вычисляем $1/k^2$
5) Следующим шагом добавляем получившееся значение из предыдущего шага к сумме, т.е. обновляем сумму.
6) Инкрементируем k после предыдущих шагов.

Также существует функция  вычисления погрешности - `calculate_error`

1) На вход функции подается вычисленное значение Pi, на выходе получаем погрешность в процентах.
2) Вычисляем абсолютную погрешность как |computed_pi - reference_pi|, то есть разница эталонного значения Pi и вычисленного.
3) Следующим шагом также важно вычислить относительную погрешность: absolute_error / reference_pi и умножить результат на 100%.
4) Вернуть процент ошибки.

Также в программе имеется **алгоритм автоматического тестирования**, в которых проходит цикл тестирования для каждого числа(итерации) - 10, 100, 1000, 10000 при помощи функции compute_pi_zeta(n). Вычисляется погрешность при помощи функции calculate_error(π). Если погрешность < 0.1%, выводится сообщение “DONE”, что означает успешное завершение тестинга, иначе вывод сообщения “Mistakes have been occurred”.
Также реализован **алгоритм интерактивного тестирования**, в котором реализовано взаимодействие программы и пользователя. Для этого пользователю необходимо ввести число целое (количество итераций вычисления числа Pi), далее идет работа программы благодаря функциям compute_pi_zeta(n) и calculate_error(π). Если получаемая точность менее 0.1%, программа выводит сообщение о необходимости ввести другое, большее по модулю число итераций.

**Решение, претендующее на 9 баллов:**

1. **Требования на 4-5:**
    - Приведено решение задачи на ассемблере. Ввод данных осуществляется с клавиатуры, вывод данных осуществляется на дисплей.
    - В программе присутствуют комментарии, поясняющие выполняемые действия.
    - Имеются требуемые подпрограммы.
    - Чуть ниже в скриншотах будет представлено тестовое покрытие.
2. **Требования на 6-7:**
    - Реализованы подпрограммы с передачей аргументов через параметры: `compute_pi_zeta` и `calculate_error`.

- Внутри подпрограмм использованы локальные переменные, чтобы при нехватке временных регистров обеспечить сохранение данных на стеке в соответствии с соглашениями, принятыми для процессора,например, локальная переменная `fsd f24, 8(sp)`.
- В местах вызова функции добавлены комментарии, описывающие передачу фактических параметров и перенос возвращаемого результата. При этом отметить, какая переменная или результат какого выражения соответствует тому или иному фактическому параметру.
- Информация о проведенных изменениях(включая предпоследний пункт выше):

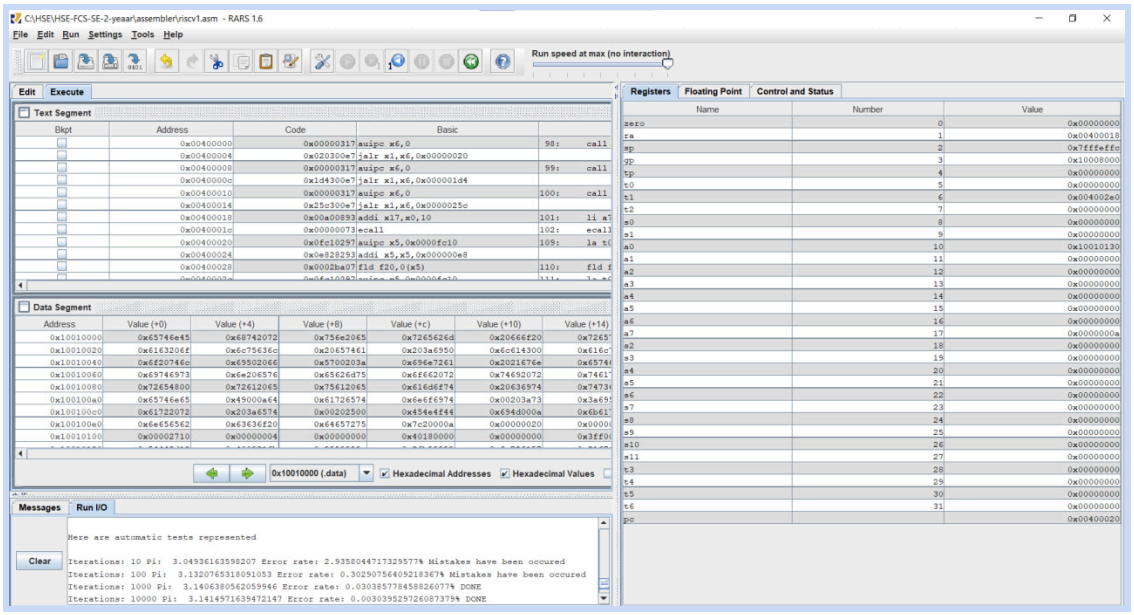
```
254 automated_test_loop:
255     beqz s1, automated_test_done
256 
257     #загружаем количество итераций и запускаем тест
258     lw a0, 0(s0) #параметр: a0 = текущее количество итераций из test_iteratio
259     call run_single_test #результат: вывод в консоль, возвращаемое значение н
260 
261     #следующий тестовый набор
262     addi s0, s0, 4 #перемещаем указатель на следующий элемент массива
263     addi s1, s1, -1 #уменьшаем счетчик оставшихся тестов
264     j automated_test_loop
```

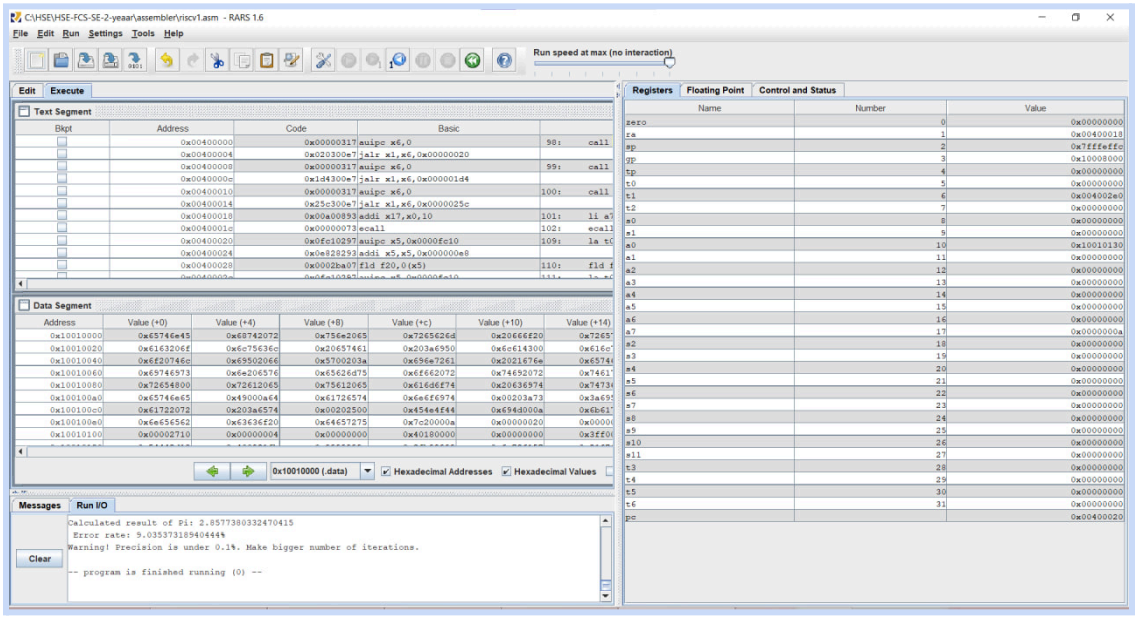
riscv1.asm*

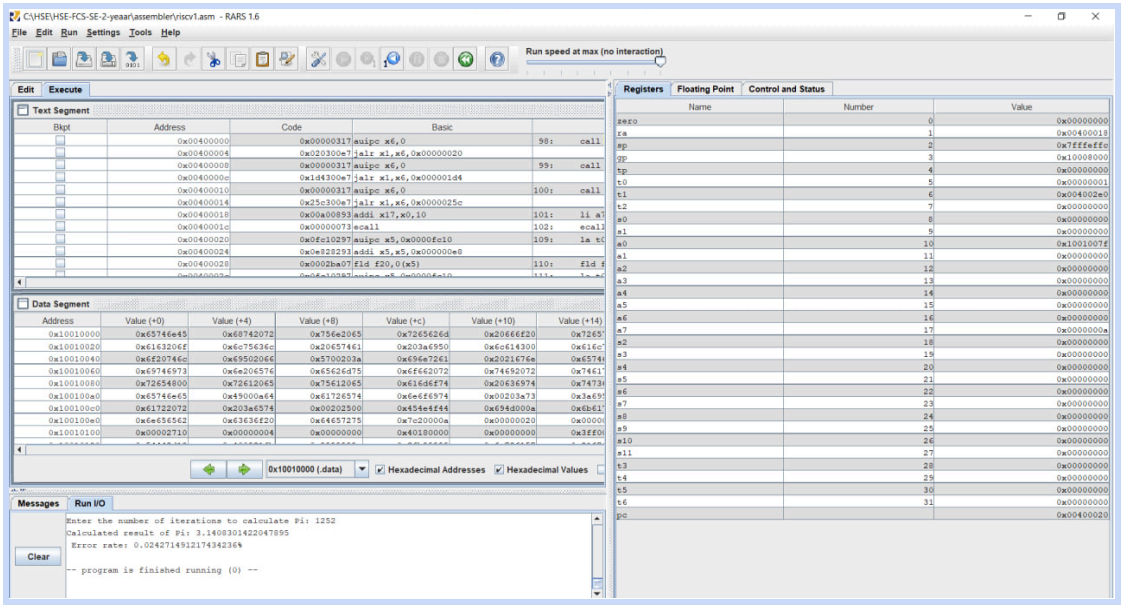
```
190 #a0 - кол-во итераций
191 run_single_test:
192     addi sp, sp, -32
193     sw ra, 0(sp)
194     fsd f8, 8(sp)     # вычисленный π
195     fsd f9, 16(sp)    # погрешность
196     sw a0, 24(sp)     # сохраняем итерации
197 
198     # Вычисляем π
199     call compute_pi_zeta
200     #параметр: a0 = количество итераций (сохранено из входного параметра)
201     #возврат: fa0 = вычисленное значение Pi
202     fmv.d f8, fa0  # Сохраняем результат: f8 = вычисленное значение Pi
203 
204     # Вычисляем погрешность
205     fmv.d fa0, f8 #подготовка параметра: fa0 = вычисленный Pi из f8
206     call calculate_error
207     fmv.d f9, fa0   #сохраняем результат: f9=погрешность в процентах
208 
209     # Выводим результаты теста
210     PRINT_STRING(test_iter_msg)
211     lw a0, 24(sp) #восстанавливаем количество итераций для вывода
212     PRINT_INT(a0)  #параметр: a0 = количество итераций
213 
214     PRINT_STRING(test_pi_msg)
215     PRINT_DOUBLE(f8)  #параметр: f8 = вычисленное значение Pi
216 
```

```
277 #интерактивный режим
278 interactive_mode:
279     addi sp, sp, -32
280     sw ra, 0(sp)
281     fsd f24, 8(sp)       # Вычисленное знач Pi
282     fsd f25, 16(sp)      #погрешность
283     sw s0, 24(sp)        #кол-во итераций
284
285     #ввод данных
286     PRINT_STRING(prompt)
287     #вызов макроса INPUT_INT_TO: ввод целого числа от пользователя
288     #параметр: %reg = s0 (регистр для сохранения результата)
289     INPUT_INT_TO(s0)
290
291     #корректность ввода
292     blez s0, interactive_error #если s0 (количество итераций) <= 0
293
294     #вычисление Pi с использованием макроса-обёртки
295     #Вызов макроса COMPUTE_PI: вычисление п через обертку
296     COMPUTE_PI(s0, f24)  #параметры: %iter_reg = s0 (количество итераций), %r
297     #возврат: f24 = вычисленное значение Pi
298
299     #вычисление погрешности
300     #параметры: %computed_freg = f24 (вычисленный п), %error_freg = f25 (реги
301     COMPUTE_ERROR(f24, f25) #возврат: f25 = погрешность в процентах
302
303     #вывод результатов
```

3. Требования на 8:
    - Разработанные подпрограммы поддерживают многократное использование с различными наборами исходных данных, включая возможность подключения различных исходных и результирующих массивов. Например, подпрограммы `compute_pi_zeta`, `calculate_error`, `run_single_test`. Реализовано многократное использование макросов-обёртки. `COMPUTE_PI(s0, f24)`, `COMPUTE_PI(t0, f25)`.
    - Реализовано автоматизированное тестирование за счет создания дополнительной тестовой программы, осуществляющей прогон подпрограмм, осуществляющих вычисления для различных тестовых данных (вместо их ввода). Осуществлен прогон тестов, обеспечивающих покрытие различных ситуаций. Описание работ тестовых программ (алгоритм автоматического тестирования и алгоритм интерактивного тестирования) описаны в отчете выше, после описания алгоритма работы всей программы.


C:\HSE\HSE-FCS-SE-2-yeaar\assembler\riscv1.asm - RARS 1.6
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
Text Segment
Bkpt Address Code Basic
0x00400000 0x00000317 auipc x6,0 98: call
0x00400004 0x020300e7 jalr x1,x6,0x00000020
0x00400008 0x00000317 auipc x6,0 99: call
0x0040000c 0x1d4300e7 jalr x1,x6,0x000001d4
0x00400010 0x00000317 auipc x6,0 100: call
0x00400014 0x25c300e7 jalr x1,x6,0x0000025c
0x00400018 0x00a00893 addi x17,x0,10 101: li a7
0x0040001c 0x00000073 ecall 102: ecall
0x00400020 0x0fc10297 auipc x5,0x0000fc10 109: la t0
0x00400024 0x0e828293 addi x5,x5,0x000000e8
0x00400028 0x0002ba07 fld f20,0(x5) 110: fld f
Data Segment
Address Value (+0) Value (+4) Value (+8) Value (+c) Value (+10) Value (+14)
0x10010000 0x65746e45 0x68742072 0x756e2065 0x7265626d 0x20666f20 0x7265
0x10010020 0x6163206f 0x6c75636c 0x20657461 0x203a6950 0x6c614300 0x616c
0x10010040 0x6f20746c 0x69502066 0x5700203a 0x696e7261 0x2021676e 0x6574
0x10010060 0x69746973 0x6e206576 0x65626d75 0x6f662072 0x74692072 0x7461
0x10010080 0x72654800 0x72612065 0x75612065 0x616d6f74 0x20636974 0x7473
0x100100a0 0x65746e65 0x49000a64 0x61726574 0x6e6f6974 0x00203a73 0x3a69
0x100100c0 0x61722072 0x203a6574 0x00202500 0x454e4f44 0x694d000a 0x6b61
0x100100e0 0x6e656562 0x63636f20 0x64657275 0x7c20000a 0x00000020 0x0000
0x10010100 0x00002710 0x00000004 0x00000000 0x40180000 0x00000000 0x3ff0
0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values
Messages Run I/O
Clear
Here are automatic tests represented
Iterations: 10 Pi: 3.04936163598207 Error rate: 2.9358044717329577% Mistakes have been occured
Iterations: 100 Pi: 3.1320765318091053 Error rate: 0.30290756405218367% Mistakes have been occured
Iterations: 1000 Pi: 3.1406380562059946 Error rate: 0.03038577845882607% DONE
Iterations: 10000 Pi: 3.1414971635472147 Error rate: 0.00303552572608737 5% DONE
Registers Floating Point Control and Status
Name Number Value
zero 0 0x00000000
ra 1 0x00400018
sp 2 0x7fffeffc
gp 3 0x10008000
tp 4 0x00000000
t0 5 0x00000000
t1 6 0x004002e0
t2 7 0x00000000
s0 8 0x00000000
s1 9 0x00000000
a0 10 0x10010130
a1 11 0x00000000
a2 12 0x00000000
a3 13 0x00000000
a4 14 0x00000000
a5 15 0x00000000
a6 16 0x00000000
a7 17 0x0000000a
s2 18 0x00000000
s3 19 0x00000000
s4 20 0x00000000
s5 21 0x00000000
s6 22 0x00000000
s7 23 0x00000000
s8 24 0x00000000
s9 25 0x00000000
s10 26 0x00000000
s11 27 0x00000000
t3 28 0x00000000
t4 29 0x00000000
t5 30 0x00000000
t6 31 0x00000000
pc 0x00400020


C:\HSE\HSE-FCS-SE-2-yeaar\assembler\riscv1.asm - RARS 1.6
File Edit Run Settings Tools Help
Run speed at max (no interaction)
Edit Execute
Text Segment
Bkpt Address Code Basic
0x00400000 0x00000317 auipc x6,0 98: call
0x00400004 0x020300e7 jalr x1,x6,0x00000020
0x00400008 0x00000317 auipc x6,0 99: call
0x0040000c 0x1d4300e7 jalr x1,x6,0x000001d4
0x00400010 0x00000317 auipc x6,0 100: call
0x00400014 0x25c300e7 jalr x1,x6,0x0000025c
0x00400018 0x00a00893 addi x17,x0,10 101: li a7
0x0040001c 0x00000073 ecall 102: ecall
0x00400020 0x0fc10297 auipc x5,0x0000fc10 109: la t0
0x00400024 0x0e828293 addi x5,x5,0x000000e8
0x00400028 0x0002ba07 fld f20,0(x5) 110: fld f
Data Segment
Address Value (+0) Value (+4) Value (+8) Value (+c) Value (+10) Value (+14)
0x10010000 0x65746e45 0x68742072 0x756e2065 0x7265626d 0x20666f20 0x7265
0x10010020 0x6163206f 0x6c75636c 0x20657461 0x203a6950 0x6c614300 0x616c
0x10010040 0x6f20746c 0x69502066 0x5700203a 0x696e7261 0x2021676e 0x6574
0x10010060 0x69746973 0x6e206576 0x65626d75 0x6f662072 0x74692072 0x7461
0x10010080 0x72654800 0x72612065 0x75612065 0x616d6f74 0x20636974 0x7473
0x100100a0 0x65746e65 0x49000a64 0x61726574 0x6e6f6974 0x00203a73 0x3a69
0x100100c0 0x61722072 0x203a6574 0x00202500 0x454e4f44 0x694d000a 0x6b61
0x100100e0 0x6e656562 0x63636f20 0x64657275 0x7c20000a 0x00000020 0x0000
0x10010100 0x00002710 0x00000004 0x00000000 0x40180000 0x00000000 0x3ff0
0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values
Messages Run I/O
Clear
Calculated result of Pi: 2.8577380332470415
Error rate: 9.035373189404444%
Warning! Precision is under 0.1%. Make bigger number of iterations.
-- program is finished running (0) --
Registers Floating Point Control and Status
Name Number Value
zero 0 0x00000000
ra 1 0x00400018
sp 2 0x7fffeffc
gp 3 0x10008000
tp 4 0x00000000
t0 5 0x00000000
t1 6 0x004002e0
t2 7 0x00000000
s0 8 0x00000000
s1 9 0x00000000
a0 10 0x10010130
a1 11 0x00000000
a2 12 0x00000000
a3 13 0x00000000
a4 14 0x00000000
a5 15 0x00000000
a6 16 0x00000000
a7 17 0x0000000a
s2 18 0x00000000
s3 19 0x00000000
s4 20 0x00000000
s5 21 0x00000000
s6 22 0x00000000
s7 23 0x00000000
s8 24 0x00000000
s9 25 0x00000000
s10 26 0x00000000
s11 27 0x00000000
t3 28 0x00000000
t4 29 0x00000000
t5 30 0x00000000
t6 31 0x00000000
pc 0x00400020




- Для дополнительной проверки корректности вычислений осуществлены аналогичные тестовые прогоны с использованием существующих библиотек и одного из языков программирования высокого уровня по выбору: Python. Файл с кодом (calculatings.py) будет добавлен в каталог, результат работы программы:

```
PS C:\HSE\HSE-FCS-SE-2-course> & C:/Users/МАРГАРИТА/AppData/Local/Programs/Python/Python312/
ngs.py
Auto test
Iteration: 10 | Pi: 3.0493616360 | Error percent is: 2.9358% Mistakes...
Iteration: 100 | Pi: 3.1320765318 | Error percent is: 0.3029% Mistakes...
Iteration: 1000 | Pi: 3.1406380562 | Error percent is: 0.0304% Hurray, it's done!
Iteration: 10000 | Pi: 3.1414971639 | Error percent is: 0.0030% Hurray, it's done!

Interactive mode
Enter the number for iterations for calculating Pi: 1001
Computed value of Pi: 3.1406390095181815
Error percent: 0.0304%
PS C:\HSE\HSE-FCS-SE-2-course>
```

- Информация о проведенных изменениях добавлена в отчет.

4. **Требования на 9:**
   - Добавлено в программу использование макросов для реализации ввода и вывода данных. Добавлены свои макросы, обертывающие подпрограммы обработки данных разработанные для решения основной задачи. Макросы поддерживают повторное использование с различными входными и выходными параметрами. Макросы для ввода/вывода; макросы-обертки для подпрограмм. Повторное использование реализовано - макросы работают с различными параметрами.

```
#внизу будут макросы для ввода/вывода
#макрос для вывода строки
.macro PRINT_STRING(%str_addr)
    la a0, %str_addr #загружаем адрес строки в a0
    li a7, 4 #в регистр a7 кладём код системного вызова
    ecall #системный вызов
.end_macro

#макрос для вывода целого числа
.macro PRINT_INT(%reg)
    mv a0, %reg #копируем значение из регистра в a0
    li a7, 1 #код вывода целого числа
    ecall
.end_macro

#макрос для вывода double
.macro PRINT_DOUBLE(%freg)
    fmv.d fa0, %freg #копируем дабл значение в fa0
    li a7, 3 #код вывода дабла
    ecall
.end_macro

```

```
#макрос-обёртка для вычисления Pi с автоматическим сохранением регистров
.macro COMPUTE_PI(%iter_reg, %result_freg)
    addi sp, sp, -24 #резервируем 24 байта в стеке
    sw ra, 0(sp) #сохраняем адрес возврата
    fsd f8, 8(sp) # сохраняем регистр f8
    sw a0, 16(sp) # сохраняем регистр a0

    mv a0, %iter_reg   #передаем количество итераций - передаем параметр в a0
    call compute_pi_zeta #вызываем функцию
    fmv.d %result_freg, fa0 #сохраняем результат

    lw a0, 16(sp) #почти аналогично, как выше(работаем с регистрами, загружае
    fld f8, 8(sp)
    lw ra, 0(sp)
    addi sp, sp, 24
.end_macro

#макрос-обертка для вычисления погрешности
.macro COMPUTE_ERROR(%computed_freg, %error_freg)
    addi sp, sp, -16
    fsd f8, 0(sp)
    sw ra, 8(sp)

    fmv.d fa0, %computed_freg
    call calculate_error
    fmv.d %error_freg, fa0
```