EFREI CA2023F

Computer Architecture – Assembly language programming part 4

# Floating point calculations

## x86 floating point unit registers

Floating point unit has 8, 80-bit, independent registers $st(0)-$st(7) organized as a stack structure. FPU can operate only on it’s own registers. There is no possibility to use general purpose CPU registers like %eax.

Such organization causes limitations in free access to FPU register. In practice free access is possible only to the $st(0) register, and access to the other FPU registers is only indirect using $st(0).

## x86 FPU instructions

There are several types of FPU instructions:

### FPU Data transfer instructions:

FLD/FILD (mem) – load real/integer number from memory.

FST (mem32/64/80) – store in memory contents of st(0).

FSTP (mem32/64/80) - store in memory contents of st(0) and pop them from register stack

FIST (mem16/32) - store in memory contents of st(0) rounded to integer

FISTP (mem16/32/64) - store in memory contents of st(0) rounded to integer and pop st(0) from register stack

FXCH st(i) –exchange value in st(0) with st(i).

### FPU load constant instructions:

FLDZ - load zero: %st(0) = 0.0

FLD1 - load 1: %st(0) = 1.0

FLDPI - load π: %st(0) = π

FLDL2T - load log2(10) : %st(0) = log2(10)

FLDL2E - load log2(e): %st(0) = log2(e)

FLDLG2 - load log(2)=log10(2) : %st(0)= log10(2)

FLDLN2 - load ln(2): %st(0)= ln(2)

### FPU Arithmetic instructions

addition:

FADD, syntax identical to simple substraction

substraction :

FSUB [mem32/64] st(0) := st(0)-[mem] ,

FSUB st(0),st(i) st := st-st(i),

FSUB st(i),st(0) st(i) := st(i)-st(0),

FSUBP st(i), st(0) st(i) := st(i)-st(0) and pop st(0),

FSUBP (no args) = FSUBP st(1),st(0) ,

FISUB [mem16/32int] st := st-[mem]

reverse substraction:

FSUBR [mem32/64] st := [mem]-st(0)

FSUBR st(0),st(i) st := st(i)-st(0)

FSUBR st(i),st(0) st(i) := st(0)-st(i)

FSUBRP st(i),st(0) st(i) := st(0)-st(i) and pop st(0)

FSUBRP (no args) = FSUBRP st(1),st(0)

FISUBR [mem16/32int] st := [mem]-st

multiply:

FMUL, syntax identical to simple substraction.

division:

FDIV, syntax identical to simple substraction.

reverse division:

FDIVR, syntax identical to reverse substraction.

absolute value:

FABS (no args) reolace st(0) his absolute value.

change sign

FCHS: st(0) := -st(0).

square root:

FSQRT: st(0) := SQRT[ st(0) ]

partial reminder:

FPREM, FPREM1 st(0) := st(0) mod st(1).

round to integer:

FRNDINT: st(0) := (int)st(0).

FSIN: st(0) := sin(st(0))

FCOS,

FSINCOS,

FPTAN,

FPATAN

FYL2X st(1) := st(1)\*log2[st(0)] and pop st(0)

FYL2XPI st(1) := st(1)\*log2[ st(0) + 1.0 ] and pop st(0)

F2XM1 st(0) := 2^[st(0)] - 1

### FPU compare instructions

FCOM st(n)/[mem]

FCOMP st(n)/[mem]

FCOMPP

FICOM [mem]

FICOMP [mem]

FCOMI st(0), st(n)

FCOMIP st(0), st(n)

FTST – compare st(0) with zero

FXAM examine contet of st(0).

### FPU control instructions

FINIT/FNINIT - FPU initialization

FLDCW, FSTCW/FNSTCW - Load/Store control word

FSTSW/FNSTSW – store FPU status word

FCLEX/FNCLEX

FLDENV, FSTENV/FNSTENV

FRSTOR, FSAVE/FNSAVE

FINCSTP, FDECSTP

FFREE

FNOP - no operations

WAIT/FWAIT – wait for FPU. Used for synchronization with CPU.

## x86 FP data types

x86 FPU uses several different types of data:

* single precision (32-bit) represented by type .float or .single (4-bytes)
* double precision (64-bit) represented by type .double (8-bytes)
* extended precision (80- bit) (10-bytes).

During this lab only single and double precision numbers will be used.

In this part of the exercise we investigate floating point number representation for .float and .double data types.

**Put your observations and answers into Lab4 answer sheet form** .

### Determining size of .float data type

Analyze the source code available in **fpu1.s** file. do necessary modifications that allows you to print on screen size of .float type variable. Then write down into a frame the size of .float

The size of float variable in x86 architecture is:

|  |
| --- |
| 32-bit (4-bytes) |

### Determining size of .double data type

Analyze the source code available in **fpu1.s** file. do necessary modifications that allows you to print on screen size of .double type variable. Then write down into a frame the size of .double.

The size of double variable in x86 architecture is:

|  |
| --- |
| 64-bit (8-bytes) |

### Representation of double type number

In data section of the source code **fpu1.s** there is a variable user\_double . This variable is initialized by value 4.17. Change that value do the different non-integer number.

assembly and execute the program.

Write down (into the frame) your value and its hexadecimal representation:

Your value:

|  |
| --- |
| 12.3 |

Hexadecimal representation of your value:

|  |
| --- |
| 40289999 9999999a |

Explain in details how this hexadecimal number can be interpreted as your non-integer value.

|  |
| --- |
| The hexadecimal representation 402599999999999a can be interpreted as the non-integer value 12.3 in double precision format according to the IEEE 754 standard. The sign bit is 0, indicating a positive value, and the exponent bits are 10000000001, representing an exponent of 1025 after applying the bias of 1023. The fraction bits are 1.0110011001100110011001100110011001100110011001100, representing the binary fraction 1.0110011001100110011001100110011001100110011001100. By combining these components, we conclude that the hexadecimal representation 402599999999999a represents the non-integer value 12.3 in double precision format. |

### Representation of float number

In data section of the source code **fpu1.s** there is a variable *user\_float* . This variable is initialized by value 4.17. Change that value do the different non-integer number.

Modify fpu1 program in such a way that it displays the hexadecimal representation of the *user\_float* value and run it

Write down (into the frame) your value and its hexadecimal representation:

Your value:

|  |
| --- |
| 12.3 |

Hexadecimal representation of your value:

|  |
| --- |
| 4144cccd |

Explain in details how this hexadecimal number can be interpreted as your non-integer value.

|  |
| --- |
| The hexadecimal number 4144CCCD can be interpreted as the non-integer value 12.3 in single precision float format (float) using the IEEE 754 standard. The sign bit is 0, indicating a positive value. The exponent bits are 10000010, representing an exponent of 130 after applying the bias of 127. The fraction bits are 1.100110011001100110011, representing the binary fraction 1.100110011001100110011. Combining these components, the hexadecimal representation 4144CCCD represents the non-integer value 12.3 in single precision float format. |

## Simple calculations with FPU

In this part of the exercise we focus on simple floating point calculations with FPU.

### Adding two FP numbers

Assembly and run program whose source code is in fpuadd.sa.m This program illustrates how to use FPU to add two floating point numbers.

In .data section there is a declaration of three FP double precision variables: val1 and val2 to store user numbers and result to store result of addition .

Program ask the user for two (non-integer) numbers, then place them in fpu registers and calculate their sum.

### Calculation of the circumference of a circle

The second example (fpucircle.asm) shows how to use other FPU specific instructions.

This program ask the user for single value which is the radius of the circle then calculates and displays the circumference of a circle with the given radius.

Analyze the source code of this program then assembly and run them.

Please remember that this program is not immune to invalid data.

### Assignment

Modify program *fpucircle* in such way that it additionally calculates and displays the area of the circle with the entered radius.

Upload source code of a properly working program on the moodle platform as a solution of this assignment. **Please remember to place your name in commentary at the beginning of the source code (Author section).**

## FPU operations and condition codes

The file **fpucond.asm** contains the source code of the program showing how conditional constructions are created based on the results of FPU calculations.

The program asks the user to enter two numbers and checks their relation to each other (equal, lower, grater).

Analyze (read the code commentary), assembly and execute this program.

Verify that a program that uses other than FCOMI comparison instructions works correctly (details can be found in code commentary)

## FP final assignment

Write a program that solves quadratic equation (ax^2+bx+c=0) in real numbers domain. User should be prompted to enter values a, b, c. The program should determine if the real roots of this equation are available, then calculate and print their value.

Upload source code of a properly working program on the moodle platform as a solution of this assignment. Please remember to place your name in commentary at the beginning of the source code (Author section).