Example: Same Code but Different Architectures I

- Let's start with a simple example

```
#include <stdio.h>

int main()
{
float a = 123.123;
printf("%.10f\n", a);
printf("%.10f\n", a*a);

a = 123.125;
```

Example: Same Code but Different Architectures II

```
printf("%.10f\n", a);
printf("%.10f\n", a*a);

}
```

- Results are

Example: Same Code but Different Architectures III

```
$gcc test.c;./a.out
123.1230010986
15159.2734375000
123.1250000000
15159.7656250000
$gcc -m32 test.c;./a.out
123.1230010986
15159.2733995339
123.1250000000
15159.7656250000
```

Example: Same Code but Different Architectures IV

- -m 32 generates code for a 32-bit environment (because we don't have a 32-bit machine)
- Therefore, same code gives different results under 32 and 64-bit environments
- Why?
- On 32 bit, 387 floating-point coprocessor is used. From gcc manual, "The temporary results are computed in 80-bit precision instead of the precision specified by the type, resulting in slightly different results compared to most of other chips."

Example: Same Code but Different Architectures V

- In other words, they somehow violate IEEE standard
- The number 123.123 has infinite digits after transformed to binary
- Compiler options can help to make things more consistent.
- For example, we use `-mfpmath=387` to let the 64-bit machine run like a 32-bit one:

Example: Same Code but Different Architectures VI

```
$gcc -mfpmath=387 test.c;./a.out
123.1230010986
15159.2733995339
123.1250000000
15159.7656250000
```

- For example, we use `-ffloat-store` to make the 32-bit machine like a 64-bit one Manual of this option said: “Do not store floating-point variables in registers, and inhibit other options that might

Example: Same Code but Different Architectures VII

change whether a floating-point value is taken from a register or memory."

```
$gcc -ffloat-store test.c;./a.out
123.1230010986
15159.2734375000
123.1250000000
15159.7656250000
$gcc -ffloat-store -m32 test.c;./a.out
123.1230010986
15159.2734375000
```

Example: Same Code but Different Architectures VIII

```
123.1250000000
15159.7656250000
```

Example: Order of Operations I

- For the same code, other issues such as order of operations can also affect results.
- Consider running a real example using a machine learning software LIBSVM (`https://www.csie.ntu.edu.tw/~cjlin/libsvm/`)
- O0:

Example: Order of Operations II

```
$ g++ -O0 svm-train.c svm.cpp -o svm-train -
$ ./svm-train -c 100 -e 0.00001 heart_scale
........*..*
optimization finished, #iter = 2872
nu = 0.148045
obj = -2526.925470, rho = 1.145512
nSV = 107, nBSV = 9
Total nSV = 107
```

- Ofast:

Example: Order of Operations III

```
$ g++ -Ofast svm-train.c svm.cpp -o svm-trai
$ ./svm-train -c 100 -e 0.00001 heart_scale
........*..*
optimization finished, #iter = 2910
nu = 0.148045
obj = -2526.925470, rho = 1.145510
nSV = 107, nBSV = 9
Total nSV = 107
```

- They are different

Example: Order of Operations IV

- Some compiler optimizations may change the order of operations
- On default settings for 64-bit environments, O0 to O3 produce the same results
- From gcc manual, `-Ofast` "disregards strict standards compliance"
- Thus order of operations become different
- -mfpmath=387 is even more sensitive to optimizations
- O0:

Example: Order of Operations V

```
$ g++ -O0 -mfpmath=387 svm-train.c svm.cpp -
$ ./svm-train -c 100 -e 0.00001 heart_scale
........*..*
optimization finished, #iter = 2941
nu = 0.148045
obj = -2526.925470, rho = 1.145513
nSV = 107, nBSV = 9
Total nSV = 107
```

- O1:

Example: Order of Operations VI

```
$ g++ -O1 -mfpmath=387 svm-train.c svm.cpp -
$ ./svm-train -c 100 -e 0.00001 heart_scale
........*..*
optimization finished, #iter = 2826
nu = 0.148045
obj = -2526.925470, rho = 1.145510
nSV = 107, nBSV = 9
Total nSV = 107
```

Example: Order of Operations VII

- To produce the same results with -mfpmath=387, we need to disable all optimizations due to more complicated interactions with registers and memory. See https://gcc.gnu.org/wiki/x87note for more details.