1. Test program and output

Test program:

#include <stdio.h>

float subtract(float in1, float in2)

{

float ret = 0.0;

asm ("fsubr %2, %0" : "=&t" (ret) : "%0" (in1), "u" (in2));

return ret;

}

int main() {

float a = 3.5;

float b = 1.7;

float c = subtract(a, b);

printf("%f\n", c);

return 0;

}

Program output:

[jonnykong@tetracosa HW3\_Gem5\_Instruction\_Hacking]$ ./run.sh

gem5 Simulator System. http://gem5.org

gem5 is copyrighted software; use the --copyright option for details.

gem5 compiled Nov 25 2018 11:23:00

gem5 started Nov 25 2018 11:24:54

gem5 executing on tetracosa.cs.ucla.edu, pid 378725

command line: build/X86/gem5.opt /home/jonnykong/gem5/configs/example/se.py --cmd=/home/jonnykong/UCLA-CS251-Advanced-Computer-Architecture/HW3\_Gem5\_Instruction\_Hacking/test --cpu-type=DerivO3CPU --l1d\_size=64kB --l1i\_size=64kB --caches --l2\_size=2MB --l2cache --sys-clock=1GHz --cpu-clock=1GHz --mem-type=DDR3\_1600\_8x8

Global frequency set at 1000000000000 ticks per second

warn: DRAM device capacity (8192 Mbytes) does not match the address range assigned (512 Mbytes)

0: system.remote\_gdb: listening for remote gdb on port 7000

\*\*\*\* REAL SIMULATION \*\*\*\*

info: Entering event queue @ 0. Starting simulation...

info: Increasing stack size by one page.

info: Increasing stack size by one page.

warn: ignoring syscall access(...)

warn: ignoring syscall mprotect(...)

warn: ignoring syscall mprotect(...)

warn: ignoring syscall mprotect(...)

warn: ignoring syscall mprotect(...)

-1.800000

Exiting @ tick 213108000 because exiting with last active thread context

2. What I like / don’t like about gem5 instruction hacking

My experience is that although I start with pattern matching, but in the meantime I also got to learn how certain features are implemented. For example, I got to know instructions are decoded in gem5, how to use the manual, how macro-ops are defined using micro-ops, etc.

Like:

1) I like how gem5 organizes its source files. For example, all x87 instructions (whose opcodes lies in range of 0xD8 to 0xDF) are defined in the same file, which is very clearly organized.

2) It uses separate files to store mappings from macro-ops to micro-ops, which is quite clear. Each kind of arithmetic operation is listed in a separate file, so it’s not very difficult to locate the file you want to modify.

Dislike:

1) I’m curious why gem5 didn’t implement these missing instructions in the first place. Although x87 is not much used now, modern compilers don’t prevent us from compiling to these instructions explicitly.

3. Patch file

<file: subtraction.py.patch>

diff --git a/src/arch/x86/isa/insts/x87/arithmetic/subtraction.py b/src/arch/x86/isa/insts/x87/arithmetic/subtraction.py

index 2275757..09abfb9 100644

--- a/src/arch/x86/isa/insts/x87/arithmetic/subtraction.py

+++ b/src/arch/x86/isa/insts/x87/arithmetic/subtraction.py

@@ -73,6 +73,43 @@ def macroop FSUB2\_P

subfp st(0), st(0), ufp1

};

+def macroop FSUBR1\_R

+{

+ subfp st(0), sti, st(0)

+};

+

+

+def macroop FSUBR1\_M

+{

+ ldfp ufp1, seg, sib, disp

+ subfp st(0), ufp1, st(0)

+};

+

+def macroop FSUBR1\_P

+{

+ rdip t7

+ ldfp ufp1, seg, riprel, disp

+ subfp st(0), ufp1, st(0)

+};

+

+def macroop FSUBR2\_R

+{

+ subfp sti, st(0), sti

+};

+

+def macroop FSUBR2\_M

+{

+ ldfp ufp1, seg, sib, disp

+ subfp st(0), ufp1, st(0)

+};

+

+def macroop FSUBR2\_P

+{

+ rdip t7

+ ldfp ufp1, seg, riprel, disp

+ subfp st(0), ufp1, st(0)

+};

+

def macroop FSUBP

{

subfp st(1), st(1), st(0), spm=1

<file: x87.isa.patch>

diff --git a/src/arch/x86/isa/decoder/x87.isa b/src/arch/x86/isa/decoder/x87.isa

index 4283d8d..6697cab 100644

--- a/src/arch/x86/isa/decoder/x87.isa

+++ b/src/arch/x86/isa/decoder/x87.isa

@@ -58,7 +58,11 @@ format WarnUnimpl {

0x3: Inst::FSUB1(Eq);

default: Inst::FSUB1(Md);

}

- 0x5: fsubr();

+ // 0x5: fsubr();

+ 0x5: decode MODRM\_MOD {

+ 0x3: Inst::FSUBR1(Eq);

+ default: Inst::FSUBR1(Md);

+ }

0x6: decode MODRM\_MOD {

0x3: Inst::FDIV1(Eq);

default: Inst::FDIV1(Md);

@@ -234,12 +238,14 @@ format WarnUnimpl {

default: fcomp();

}

0x4: decode MODRM\_MOD {

- 0x3: fsubr();

+ // 0x3: fsubr();

+ 0x3: Inst::FSUBR2(Mq);

default: Inst::FSUB2(Mq);

}

0x5: decode MODRM\_MOD {

0x3: Inst::FSUB2(Eq);

- default: fsubr();

+ // default: fsubr();

+ default: Inst::FSUBR2(Mq);

}

0x6: decode MODRM\_MOD {

0x3: fdivr();