

• تفاوت FSUB و FSUBR

FSUBR و FSUB دستورالعمل های مربوط به تفریق ممیزشناور (Floating Point) هستند.

در FSUB اپرند دوم را از اپرند اول کم می کنیم. یعنی:

FSUB: Operand1 – Operand2

در FSUBR معکوس (Reverse) عمل کرده و اپرند اول را از اپرند دوم کم می کنیم. یعنی:

FSUBR: Operand2 – Operand1

• تفاوت D8h و DCh

تفاوت اصلی بین این دو پیاده سازی در اندازه عملوندهای ممیز شناوری است که روی آنها کار می کند.

D8h: برای تفریق ممیزشناور 32 بیتی استفاده می شود. توسط یک بایت ModR/M نشان داده می شود که عملوندهای مبدا و مقصد را مشخص می کند. هم چنین این بایت برای رمزگذاری دو عملوند و عملیات استفاده می شود که شامل اطلاعات مربوط به رجیسترهای مربوط به دستورالعمل است.

DCh: برای تفریق ممیزشناور 64 بیتی استفاده می شود. توسط یک بایت ModR/M نشان داده می شود که عملوندهای مبدا و مقصد را مشخص می کند. این بایت برای تعیین رجیسترهای درگیر در تفریق ممیز شناور با دقت دوگانه استفاده می شود.

• تغییرات ISA X87

در سه مکان مختلف در فایل x87.isa، جملات مرتبط با دستور FSUBR را جستجو کرده و با جملاتی مشابه عبارت های مشخص شده برای دستور FSUB جایگزین می کنیم. با استفاده از عبارتی مانند Inst::FSUB درخواست می کنیم که از این دستور به جای دستور پیش فرض، که فقط یک هشدار عدم پیاده سازی چاپ می کند، استفاده شود.

هدف این تمرین آشنایی با زبان gem 5 برای توصیف مجموعه دستورات است. برای این کار فایل های ISA موجود در مسیر isa86/x/arch/src مورد بررسی قرار می گیرند.

ابتدا یک دستور از معماری X87 با نام FSUBR را برای X86 پیاده سازی می کنیم. سپس برای تست دستور پیاده سازی شده، یک برنامه نوشته که از این دستور خاص با استفاده از ویژگی assembly Inline استفاده کند. سپس برنامه را با استفاده از gem 5 شبیه سازی می کنیم.

1. گام اول: بررسی تفاوت دو پیاده سازی FSUBR

در این گام به بررسی مفهوم ModRM، تفاوت FSUB و FSUBR و تفاوت بین دو پیاده سازی دستور FSUBR با DCh و D8h OpCode می پردازیم.

• ModRM

یک بایت در کدگذاری دستور زبان ماشین X86 است. این فیلد بخشی از کد عملیاتی دستورالعمل است و اطلاعاتی در مورد حالت آدرس دهی و عملوندهای درگیر در دستورالعمل ارائه می دهد.

بایت MODRM به سه قسمت تقسیم می شود: MOD: حالت (Mode) آدرس دهی را مشخص می کند و نشان می دهد که دستورالعمل شامل یک رجیستر، حافظه و یا ترکیبی از هر دو است. REG/Opcode Extension: عملوند رجیستر را مشخص می کند. در برخی موارد Opcode را گسترش می دهد.

RM (Register/Memory): عملوند دوم را مشخص می کند که می تواند یک ثبات یا یک آدرس حافظه باشد. تفسیر فیلد MODRM به دستوری که اجرا می شود، بستگی دارد. دستورالعمل های مختلف از این فیلد برای رمزگذاری حالت های آدرس دهی مختلف و ترکیب های عملوند استفاده می کنند. به طور خلاصه، فیلد MODRM با انتقال اطلاعات در مورد مدهای آدرس دهی و عملوندها، نقش مهمی در رمزگذاری دستورات X86 ایفا می کند و به پردازنده اجازه می دهد دستورالعمل ها را به درستی تفسیر و اجرا کند.

مراحل انجام کار در این گام عبارت است از:

در فایل x87.isa در دایرکتوری زیر:

Src/arch/x86/isa/decoder/x87.isa

تغییرات زیر در فایل ایجاد شده است:

تغییر خط 47 این فایل از

0x5: fsubr();

به

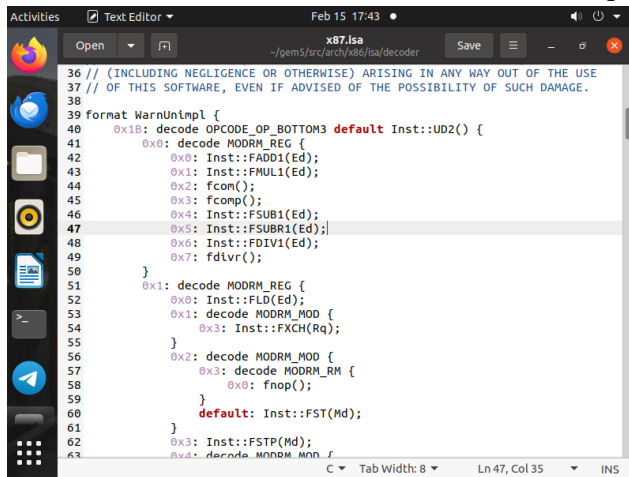
0x5: Inst::FSUBR1(Eq);

و تغییر خط 202 فایل از

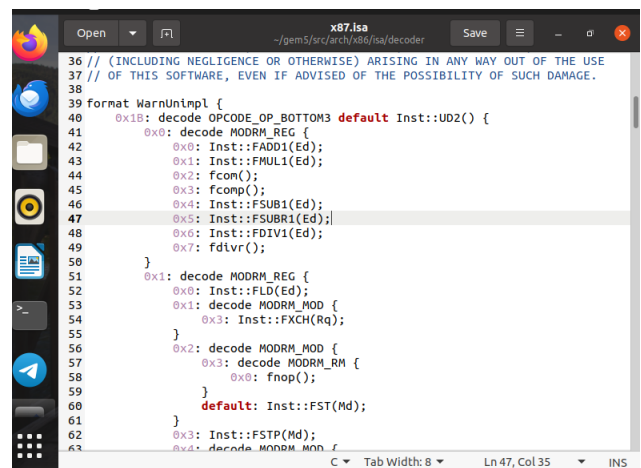
default: fsubr();

به

default: Inst::FSUBR2(Eq);



```
36 // (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
37 // OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
38
39 format WarnUnimpl {
40   0x1B: decode OPCODE_OP_BOTTOM3 default Inst::UD2() {
41     0x0: decode MODRM_REG {
42       0x0: Inst::FADD1(Eq);
43       0x1: Inst::FMUL1(Eq);
44       0x2: fcom();
45       0x3: fcomp();
46       0x4: Inst::FSUB1(Eq);
47       0x5: Inst::FSUBR1(Eq);
48       0x6: Inst::FDIV1(Eq);
49       0x7: fdivr();
50     }
51     0x1: decode MODRM_REG {
52       0x0: Inst::FLD(Eq);
53     }
54     0x1: decode MODRM_MOD {
55       0x3: Inst::FXCH(Rq);
56     }
57     0x2: decode MODRM_MOD {
58       0x3: decode MODRM_RM {
59         0x0: fnop();
60       }
61       default: Inst::FST(Md);
62     }
63     0x3: Inst::FSTP(Md);
64     0x4: decode MODRM_MOD {
```



```
36 // (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
37 // OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
38
39 format WarnUnimpl {
40   0x1B: decode OPCODE_OP_BOTTOM3 default Inst::UD2() {
41     0x0: decode MODRM_REG {
42       0x0: Inst::FADD1(Eq);
43       0x1: Inst::FMUL1(Eq);
44       0x2: fcom();
45       0x3: fcomp();
46       0x4: Inst::FSUB1(Eq);
47       0x5: Inst::FSUBR1(Eq);
48       0x6: Inst::FDIV1(Eq);
49       0x7: fdivr();
50     }
51     0x1: decode MODRM_REG {
52       0x0: Inst::FLD(Eq);
53     }
54     0x1: decode MODRM_MOD {
55       0x3: Inst::FXCH(Rq);
56     }
57     0x2: decode MODRM_MOD {
58       0x3: decode MODRM_RM {
59         0x0: fnop();
60       }
61       default: Inst::FST(Md);
62     }
63     0x3: Inst::FSTP(Md);
64     0x4: decode MODRM_MOD {
```

فایل تغییر داده شده در پوشه Step1 تمرین قرار داده شده است.

2. گام دوم: پیاده سازی برای FSUBR

در گام دوم، باید پیاده سازی macro-op برای FSUBR به صورت micro-op ها انجام شود. مجددا می توان از پیاده سازی دستور FSUB الگوبرداری کرد.

با توجه به اینکه FSUB1 و FSUB2 به دو opcode مختلف اشاره دارند، بنابراین برای هر نوع، سه پیاده سازی مختلف انجام شد. یکی از این دستورات تنها از ثباتها برای اجرای عملیات استفاده می کند. دستور دیگر یکی از عملوندها را با استفاده از آدرس موجود در دستور از حافظه می خواند و در نهایت، دستور آخر از آدرس مشخص شده توسط اشاره گر دستور برای خواندن یکی از عملوندها استفاده می کند.

برای انجام این گام به دایرکتوری زیر:

Src/arch/x86/isa/insts/x87/arithmetic/subtraction.py

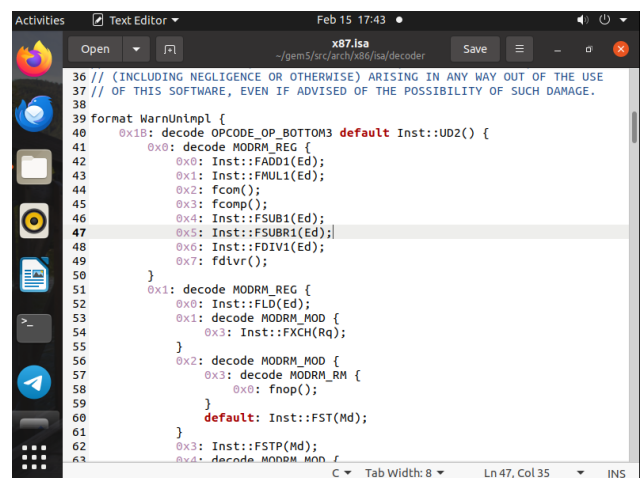
مراجعه کرده و تغییرات زیر را در این فایل، به صورت زیر اعمال می کنیم:

تغییر خط 197 از

0x3: fsubr();

به

0x3: Inst::FSUBR2(Eq);



```
36 // (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
37 // OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
38
39 format WarnUnimpl {
40   0x1B: decode OPCODE_OP_BOTTOM3 default Inst::UD2() {
41     0x0: decode MODRM_REG {
42       0x0: Inst::FADD1(Eq);
43       0x1: Inst::FMUL1(Eq);
44       0x2: fcom();
45       0x3: fcomp();
46       0x4: Inst::FSUB1(Eq);
47       0x5: Inst::FSUBR1(Eq);
48       0x6: Inst::FDIV1(Eq);
49       0x7: fdivr();
50     }
51     0x1: decode MODRM_REG {
52       0x0: Inst::FLD(Eq);
53     }
54     0x1: decode MODRM_MOD {
55       0x3: Inst::FXCH(Rq);
56     }
57     0x2: decode MODRM_MOD {
58       0x3: decode MODRM_RM {
59         0x0: fnop();
60       }
61       default: Inst::FST(Md);
62     }
63     0x3: Inst::FSTP(Md);
64     0x4: decode MODRM_MOD {
```

```
def macroop FSUBR2_P
```

```
{
    rdip t7
    ldftp ufp1, seg, riprel, disp
    subfp st(0), ufp1, st(0)
};
```

```
114 def macroop FSUBR2_R
115 {
116     subfp sti, st(0), sti
117 };
118 def macroop FSUBR2_M
119 {
120     ldftp ufp1, seg, sib, disp
121     subfp st(0), ufp1, st(0)
122 };
123 def macroop FSUBR2_P
124 {
125     rdip t7
126     ldftp ufp1, seg, riprel, disp
127     subfp st(0), ufp1, st(0)
128 };
129
130 # FISUB
131 # FSUBR
132 # FISUBR
133 """
```

این فایل در پوشه Step2 تمرین قرار داده شده است.

سپس برای بررسی درستی تغییرات، GEM5 را مجدداً بیلد کردیم:

```
user@user-virtual-machine: ~/gem5
user@user-virtual-machine:~/gem5$ cd gem5
user@user-virtual-machine:~/gem5$ scons build/X86/gem5.opt -j 4
scons: Reading SConscript files ...

You're missing the pre-commit/commit-msg hooks. These hook help to ensure your
code follows gem5's style rules on git commit and your commit messages follow
our commit message requirements. This script will now install these hooks in
your .git/hooks/ directory.
Press enter to continue, or ctrl-c to abort:
Cannot find 'pre-commit'. Please ensure all Python requirements are
installed. This can be done via 'pip install -r requirements.txt'.
It is strongly recommended you install the pre-commit hooks before working with
gem5. Do you want to continue compilation (y/n)?
y
Mkdir("/home/user/gem5/build/X86/gem5.build")
Checking for linker -Wl,--as-needed support... (cached) yes
Checking for compiler -g support... (cached) yes
Checking for linker -g support... (cached) yes
Info: Using Python config: python3-config
Checking for C header file Python.h... (cached) yes
Checking Python version... (cached) 3.8.10
Checking for accept(0,0,0) in C++ library None... (cached) yes
Checking for zlibVersion() in C++ library z... (cached) yes
Checking for C library tcmalloc_minimal... (cached) yes
Building in /home/user/gem5/build/X86
Using saved variables file(s) /home/user/gem5/build/X86/gem5.build/variables
Checking for C header file linux/kvm.h... (cached) yes
Checking for timer_create(CLOCK_MONOTONIC, NULL, NULL) in C library None... (ca
ched) no
```

در مجموع 6 بلوک به کد آن فایل اضافه کردیم:

```
def macroop FSUBR1_R
{
    subfp st(0), sti, st(0)
};
```

```
def macroop FSUBR1_M
{
    ldftp ufp1, seg, sib, disp
    subfp st(0), ufp1, st(0)
};
```

```
def macroop FSUBR1_P
{
    rdip t7
    ldftp ufp1, seg, riprel, disp
    subfp st(0), ufp1, st(0)
};
```

```
98
99 def macroop FSUBR1_R
100 {
101     subfp st(0), sti, st(0)
102 };
103 def macroop FSUBR1_M
104 {
105     ldftp ufp1, seg, sib, disp
106     subfp st(0), ufp1, st(0)
107 };
108 def macroop FSUBR1_P
109 {
110     rdip t7
111     ldftp ufp1, seg, riprel, disp
112     subfp st(0), ufp1, st(0)
113 };
```

```
def macroop FSUBR2_R
{
    subfp sti, st(0), sti
};
```

```
def macroop FSUBR2_M
{
    ldftp ufp1, seg, sib, disp
    subfp st(0), ufp1, st(0)
};
```

```

user@user-virtual-machine:~/gem5$ build/X86/gem5.opt configs/learning_gem5/part
1/simple.py gem5/aca3/a.out
gem5 Simulator System. https://www.gem5.org
gem5 is copyrighted software; use the --copyright option for details.

gem5 version 23.0.1.0
gem5 compiled Feb 15 2024 18:42:32
gem5 started Feb 15 2024 20:59:36
gem5 executing on user-virtual-machine, pid 7953
command line: build/X86/gem5.opt configs/learning_gem5/part1/simple.py gem5/aca
3/a.out

Global frequency set at 100000000000 ticks per second
warn: No dot file generated. Please install pydot to generate the dot file and
pdf.
src/nen/dram_interface.cc:690: warn: DRAM device capacity (8192 Mbytes) does no
t match the address range assigned (512 Mbytes)
src/base/statistics.hh:279: warn: One of the stats is a legacy stat. Legacy sta
t is a stat that does not belong to any statistics::Group. Legacy stat is depre
cated.
system.remote_gdb: Listening for connections on port 7000
Beginning simulation!
src/sim/simulate.cc:194: info: Entering event queue @ 0. Starting simulation..
.
src/sim/mem_state.cc:443: info: Increasing stack size by one page.
src/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
src/sim/syscall_emul.cc:74: warn: ignoring syscall mprotect(...)
10.099999

```

همانطور که در تصویر مشاهده می شود، خروجی کد حاصل
تفریق دو عدد داده شده در کد، با دستور Fsubr است.

سپس با استفاده از دستور زیر تغییراتی که در فایل isa87.x و
فایل subtraction.py اعمال شده است را می توانیم در فایل
با نام changes.patch ذخیره کنیم:

```
git diff src/arch/x86/isa > /tmp/changes.patch
```

این فایل در پوشه step2 تمرین قرار داده شده است.

3. گام سوم: برنامه تست دستور FSUBR

در این مرحله برای تست پیاده سازی دستور FSUBR برنامه
ای به زبان C نوشته شده است. این برنامه با نام testfsubr.c در
پوشه Step3 تمرین قرار داده شده است.

در این کد، دو عدد اعشاری کد با دستور fsubr از یکدیگر تفریق
می شوند و حاصل چاپ می شود. هم چنین با استفاده از ویژگی
assembly Inline کامپایلر GCC از درستی دستور FSUBR
برای تفریق، اطمینان حاصل شد.

از دستور زیر در ترمینال برای کامپایل کد نوشته شده استفاده
شد:

```
gcc testfsubr.c
```

که با اجرای این دستور فایل a.out ایجاد می شود.

سپس در فایل simple.py مسیر این کد را وارد می کنیم:

```

Open  simple.py  Save  -  a
~/gem5/configs/learning_gem5/part1
Specify the path to the "Hello World" binary. The path can be found in "tests/test-progs/hello".
91 # workloads compiled to those ISAs. Other "hello world" binaries for other
92 # can be found in "tests/test-progs/hello".
93 thispath = os.path.dirname(os.path.realpath(__file__))
94 binary = os.path.join(
95     thispath,
96     "../../..",
97     "aca3/a.out",
98 )
99
100 system.workload = SEWorkload.init_compatible(binary)
101
102 # Create a process for a simple "Hello World" application
103 process = Process()
104 # Set the command
105 # cmd is a list which begins with the executable (like argv)
106 process.cmd = [binary]
107 # Set the cpu to use the process as its workload and create thread contexts
108 system.cpu.workload = process
109 system.cpu.createThreads()
110
111 # set up the root SimObject and start the simulation
112 root = Root(full_system=False, system=system)
113 # instantiate all of the objects we've created above
114 m5.instantiate()
115
116

```

درنهایت برای اجرای کد در ترمینال دستور زیر را وارد می کنیم:

```

build/X86/gem5.opt
configs/learning_gem5/part1/simple.py
aca3/a.out

```