پروژه اول درس ریزپردازنده و زبان اسمبلی

بهینه‌سازی عملیات شبکه‌های عصبی کانولوشنی با استفاده از دستورالعمل‌های سفارشی در معماری RISC-V

دانشگاه صنعتی امیرکبیر، دانشکده مهندسی کامپیوتر

موعد تحویل تمرین ۱۸ خرداد ماه ساعت ۲۳:۵۹ می‌باشد.

- سوالات خود را می‌توانید از طریق تلگرام از مهیار اسماعیلی (@mahyar8345es) و پارسا غفرانی (@Some-onepa) بپرسید.
- یک گزارش مختصر (فایل PDF) شامل شرح روش‌ها و نتایج را آپلود کنید.
- فایل‌های پروژه شامل کد اسمبلی(فایل s.) و کد C بخش دوم را همراه با گزارش پروژه در قالب یک فایل Zip به فرمت P1-studentNumber.zip بارگذاری کنید.

شرح کلی پروژه

در این پروژه با هدف بهینه‌سازی اجرای عملیات رایج در شبکه‌های عصبی کانولوشنی (مانند Convolution، Pooling و ضرب ماتریس)، اقدام به طراحی و پیاده‌سازی دستورالعمل‌های سفارشی در معماری RISC-V می‌شود. این دستورالعمل‌ها در سطح اسمبلی تعریف شده و در محیط شبیه‌ساز اجرا خواهند شد تا میزان تأثیر آنها بر عملکرد بررسی گردد.

اهداف پروژه

- پیاده‌سازی عملیات با استفاده از RV32IM
- اضافه کردن دستورالعمل‌های سفارشی به کراس‌کامپایلر و پیاده‌سازی منطق دستورها در سورس کد شبیه‌ساز
- پیاده‌سازی عملیات‌ها با استفاده از دستورالعمل‌های سفارشی اضافه‌شده، کامپایل و اجرای موفق برنامه‌ها

عملیات‌های CNN

در شبکه‌های عصبی کانولوشنی، عملیات‌های متعددی برای استخراج ویژگی‌ها و پردازش داده‌ها انجام می‌شود، اما در این پروژه تمرکز ما بر روی سه عملیات کلیدی و پرتکرار است:

- Convolution: عمل اصلی در CNN که با اعمال فیلترهای کوچک بر روی تصویر یا داده ورودی، ویژگی‌های محلی را استخراج می‌کند. این عملیات شامل ضرب و جمع ماتریسی در یک ناحیه کوچک از ورودی است.
- Pooling: برای کاهش ابعاد داده و افزایش مقاومت نسبت به جابجایی استفاده می‌شود. رایج‌ترین نوع آن max pooling است که بیشترین مقدار را از یک ناحیه انتخاب می‌کند.
- ضرب ماتریس: یکی از عملیات پایه‌ای در شبکه عصبی است که برای ترکیب داده‌ها و وزن‌ها استفاده می‌شود. این عملیات در لایه‌های نهایی شبکه کاربرد زیادی دارد و شامل ضرب عناصر ردیف‌ها و ستون‌های دو ماتریس و جمع آنهاست.

بخش اول: پیاده‌سازی با RV32IM و اجرا در محیط شبیه‌ساز

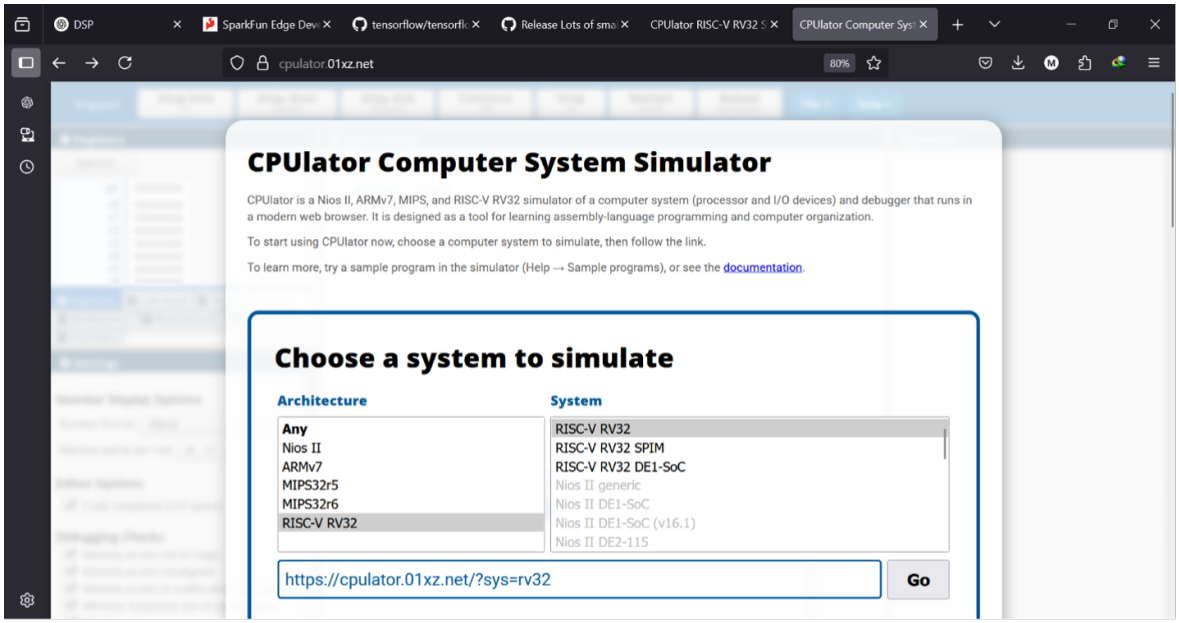
در این قسمت باید عملیات مورد نظر (یکی از سه عملیات ذکر شده) را با استفاده از دستورالعمل‌های استاندارد RISC-V و بدون استفاده از دستورالعمل‌های شخصی‌سازی‌شده پیاده‌سازی کنید.

هدف از این بخش مقایسه حجم کد و در نتیجه میزان کاری است که پردازنده باید در دو حالت با و بدون استفاده از custom instructionها انجام دهد.

معماری RV32IM یکی از زیرمجموعه‌های استاندارد معماری RISC-V است که شامل:

- RV32I: مجموعه‌دستور پایه برای عملیات ۳۲-بیتی (جمع، تفریق، بارگذاری، پرش و مقایسه)
- M: افزونه‌ای برای پشتیبانی از ضرب و تقسیم صحیح

محیط شبیه‌ساز CPUlator امکان شبیه‌سازی RV32IM را فراهم کرده است؛ بنابراین شما می‌توانید در همان محیطی که در آزمایشگاه دستورات ARMv7 را شبیه‌سازی می‌کردید، دستورات RV32IM را نیز شبیه‌سازی کنید. در ابتدای ورود به لینک مطابق شکل زیر گزینه‌ها را انتخاب کنید:



شکل ۱: محیط شبیه‌ساز CPUlator

یکی دیگر از شبیه‌سازهایی که می‌توانید از آن استفاده کنید RARS است که در محیط بسیار مشابه CPUlator کار می‌کند و از قابلیت autocompletion پشتیبانی می‌کند. برای استفاده، ابتدا به این لینک مراجعه کرده و فایل jar. آن را دانلود کنید. در صورتی که از پیش یک ماشین مجازی جاوا (JVM) روی سیستم خود داشته باشید، می‌توانید فایل را اجرا کنید.

توضیحاتی درباره اسمبلی RISC-V

- اسمبلی RISC-V تا حد بسیار زیادی مشابه ARMv7 است و تنها تفاوت‌های جزئی با هم دارند؛ بنابراین نیازی نیست نگران آشنا نبودن با این زبان باشید.
- ABI معماری RISC-V برای هر رجیستر عام‌منظوره یک اسم در نظر گرفته که باعث خوانایی بیشتر کد می‌شود. برای آشنایی بیشتر به این لینک مراجعه کنید.
- RV32 دارای تعداد رجیسترهای عام‌منظوره بیشتری نسبت به ARM ۳۲-بیتی است که برنامه‌نویسی را راحت‌تر می‌کند.
- RV32 از دستورهای PUSH و POP پشتیبانی نمی‌کند و مدیریت استک باید با استفاده از رجیستر sp و با توجه به این نکته که رشد استک در حافظه نزولی است (مانند ARM) صورت گیرد.
- مثالی از کد RISC-V برای شما قرار داده خواهد شد که سعی می‌شود در تفاوت‌های آن با ARM تاکید شود.

بخش دوم: اضافه کردن دستورالعمل‌های سفارشی و پیاده‌سازی عملیات‌ها

در این لینک پروسه کامل راه‌اندازی و اضافه کردن دستورها در قالب یک مثال نشان داده شده و آموزش‌ها نیز بر پایه همین سایت است که در صورت تمایل می‌توانید از آن استفاده کنید.

راه‌اندازی (setup)

محیط لازم: برای این بخش نیاز به یک سیستم لینوکسی یا ماشین مجازی دارید. در صورت استفاده از ویندوز می‌توانید از WSL[1] بهره بگیرید، اما اجباری نیست. اگر از ماشین مجازی یا WSL استفاده می‌کنید، حتماً بیشینه منابع CPU و RAM را تخصیص دهید تا در مرحله build کراس‌کامپایلر با مشکل مواجه نشوید. برای تغییر منابع WSL، فایل C:\Users\<yourUsername>\.wslconfig را ویرایش کنید.

Toolchain: پس از فراهم کردن محیط، نیازمندی‌ها را نصب و ریپازیتوری RISC-V را کلون کنید:

```
sudo apt-get install autoconf automake autotools-dev curl python3
libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo
gperf libtool patchutils bc zlib1g-dev libexpat-dev device-tree-compiler
```

```
git clone --recurse-submodules \
https://github.com/riscv/riscv-gnu-toolchain.git
```

نکته: حدوداً ۷ گیگابایت برای دانلود تمام ریپازیتوری‌ها نیاز است. پس از کلون، یک بار کراس‌کامپایلر را build کنید:

```
cd riscv-gnu-toolchain
./configure --prefix=/opt/riscv_custom
make -j$(nproc)
```

دستور دوم محل قرار گرفتن باینری‌های کراس‌کامپایلر پس از اتمام پروسه کامپایل شدن را مشخص می‌کند.

نکته مهم: کامپایل شدن کراس کامپایلر، بسته به سیستم شما ممکن است تا یک ساعت و بیشتر. در برنامه‌ریزی خود برای تحویل پروژه حتما به این نکته توجه داشته باشید.

[1] Windows Subsystem for Linux

در صورتی که در این مرحله به مشکل یا اروری برنخورید و کراس‌کامپایلر با موفقیت کامپایل شود، دستور زیر ورژن کراس کامپایلر را خروجی می‌دهد.

```
/opt/riscv_custom/bin/riscv64-unknown-elf-gcc --version
```

اضافه کردن دستورالعمل‌های اسمبلی سفارشی

برای هر موضوع انتخابی، باید دستورهای زیر را اضافه کنید:

Max pooling	Convolution	ضرب ماتریس
max2x2	conv2x2	mac

جدول ۱: دستورالعمل‌های سفارشی

برای سادگی، همه دستورها R-type هستند.

۱. دستور mac:

```
mac rd, rs1, rs2

rd = rd + (rs1 * rs2)
```

۲. دستور conv2x2:

```
conv2x2 rd, rs1, rs2

rs1: pointer to the top left of the input channel

rs2: pointer to the base of kernel (filter)

rd: destination register to store the convolution result

stride = 1
```

۳. دستور max2x2:

```
max2x2 rd, rs1, rs2

rs1: pointer to the top left of a 2x2 window

rs2: col-stride

rd: destination register to store the maximum element

col-stride = 2
```

پس از افزودن و پیاده‌سازی منطق این دستورها در شبیه‌ساز و کراس‌کامپایلر، برنامه‌ای بنویسید که با inline assembly in C دقیقاً همان کاری را انجام دهد که در بخش اول با RV32IM انجام دادید.

برای پیاده‌سازی قسمت شبیه‌ساز دستورات max2x2 و conv2x2 میتوانید از توابع استفاده شده در فایل lw.h در مسیر riscv-isa-sim/riscv/insns/ زیر استفاده کنید