به نام خدا

معماری کامپیوتر

پروژهٔ پایانی ـ بخش دوم ـ شبیه‌سازی gem5

دانشکدهٔ مهندسی کامپیوتر

دانشگاه صنعتی شریف

استاد:

جناب آقای دکتر اسدی

نام، نام خانوادگی و شمارهٔ دانشجویی اعضای گروه:

سپهر پورقناد ـ ۹۷۱۰۱۳۵۹

سیدمحمدصادق کشاورزی ـ ۹۷۱۰۶۲۴۹

امیرمهدی نامجو ـ ۹۷۱۰۷۲۱۲

۱ بررسی دستورات

الگوریتم SHA-256 دارای دو دسته از دستورات است که به صورت دستورات رایج ISA نیست. در واقع این دستورات ترکیبی از چند دستور رایج ISA هستند که مقادیر محاسبه‌شدهٔ میانی دوباره استفاده نمی‌شوند. اگر هر کدام از این دستورات را بتوان در مدت زمان یک دستور رایج ISA انجام داد، سرعت اجرای برنامه سریع‌تر می‌شود. این دو دسته به صورت زیر هستند:
دسته اول:

s0 := (w[i−15] **rightrotate** 7) **xor** (w[i−15] **rightrotate** 18)
xor (w[i−15] rightshift 3)

s1 := (w[i− 2] **rightrotate** 17) **xor** (w[i− 2] **rightrotate** 19)
xor (w[i− 2] rightshift 10)

w[i] := w[i−16] + s0 + w[i−7] + s1

دسته دوم:

S1 := (e **rightrotate** 6) **xor** (e **rightrotate** 11) **xor** (e **rightrotate** 25)
ch := (e **and** f) **xor** ((**not** e) **and** g)
temp1 := h + S1 + ch + k[i] + w[i]
S0 := (a **rightrotate** 2) **xor** (a **rightrotate** 13) **xor** (a **rightrotate** 22)
maj := (a **and** b) **xor** (a **and** c) **xor** (b **and** c)

هر کدام از دستورات بالا گزینه‌ای برای اضافه کردن به ISA است.
ما در اینجا دستور `a = (b & c) (~b & d)` پیاده‌سازی می‌کنیم و اثر آن بر بهبود سرعت اجرای برنامه را بررسی می‌کنیم.

۲ اضافه کردن دستور

فایل src/arch/x86/isa/decoder/two_byte_opcodes.isa:

```
0x56: mynewop({{
    Rax = PseudoInst::mynewop(xc->tcBase(), Rdi, Rsi, Rdx);
}}, IsNonSpeculative);
```

فایل src/sim/pseudo_inst.cc:

```
uint64_t
mynewop(ThreadContext *tc, uint64_t arg1, uint64_t arg2, uint64_t arg3)
{
    quiesceCycle(tc, 100);
    return (((arg1) & (arg2)) ^ (~(arg1) & (arg3)));
}
```

که تأخیر اجرای دستور را (بر حسب تعداد چرخهٔ ساعت) به عنوان ورودی به دستور quiesceCycle می‌دهیم (که در مثال بالا برابر ۱۰۰ چرخهٔ ساعت است).

۳ تأخیر زیر را برای این دستور در نظر می‌گیریم:

۱. ۱ چرخهٔ ساعت: که حالت ایده‌آل است.

۲. ۲۰ چرخهٔ ساعت: که برابر تعداد چرخهٔ لازم برای اجرای این دستور در پردازندهٔ multicycle آموزش‌داده‌شده در درس است (با فرض موجود بودن متغیرها در ثبات‌ها).

۳. ۱۰۰ چرخهٔ ساعت: که تخمینی برای حد بالای تعداد چرخهٔ لازم برای اجرای این دستور در پردازندهٔ ۸۰۸۶ است (با فرض موجود بودن متغیرها در حافظه).

زمان‌های اجرا به ازای رشتهٔ ورودی "how are you doing today?" به شرح زیر است:

۱. ISA اصلی: ۱۶۳,۰۲۱,۰۰۰ تیک

۲. دستور اختصاصی با تأخیر ۱ چرخهٔ ساعت: ۱۶۳,۴۶۰,۰۰۰ تیک ← بهبود = ۰/۹۹۷ (احتمالاً علت این که نه تنها بهبود نداشته‌ایم، بلکه زمان اجرا بدتر نیز شده‌است، این است که کامپایلر این دستور را نمی‌شناسد و اجرای این دستور جدید، بهینه‌سازی‌های کامپایلر (از جمله تخصیص ثبات) را به هم می‌ریزد و از بین می‌برد).

۳. دستور اختصاصی با تأخیر ۲۰ چرخهٔ ساعت: ۱۶۴,۰۶۸,۰۰۰ تیک ← بهبود = ۰/۹۹۳

۴. دستور اختصاصی با تأخیر ۱۰۰ چرخهٔ ساعت: ۳۲,۱۶۳,۴۲۸,۰۰۰ تیک ← بهبود = ۰/۰۰۵۰۶

همهٔ حالات با تنظیمات se.py بدون هیچ تغییری اجرا شدند و هر ثانیه برابر 10^{12} تیک است.

۳ تأثیر حافظه نهان

- ISA اصلی:
  - بدون حافظهٔ نهان: ۱۳,۲۸۸,۷۳۰,۰۰۰ تیک
  - حافظهٔ نهان دوسطحی: ۷۳۵,۴۷۴,۰۰۰ تیک ← بهبود = ۱۸/۰۶۸

تعداد دسترسی‌های خواندن داده از حافظه از ۴۵۰۵۲ به ۴۵۰۴۶ و نوشتن دستور به حافظه از ۱۹۹۸۴۳ به ۱۹۹۷۵۶ کاهش یافته‌است. تعداد دسترسی‌ها و missهای دیگر تغییر نداشته‌است.

- دستور اختصاصی با تأخیر ۱۰۰ چرخهٔ ساعت:
  - بدون حافظهٔ نهان: ۱۳,۳۳۲,۸۹۴,۰۰۰
  - حافظهٔ نهان دوسطحی: ۷۴۲,۸۴۳,۰۰۰ ← بهبود = ۱۷/۹۴۸

تعداد خواندن‌های داده از حافظه نیز از ۴۵۰۵۲ به ۴۵۰۴۶ و تعداد نوشتن‌های دستور به حافظه از ۲۰۰۶۳۴ به ۲۰۰۵۴۷ کاهش یافته‌است. تعداد دسترسی‌ها و missهای دیگر تغییر نداشته‌است.

۴ تنظیمات حافظه نهان

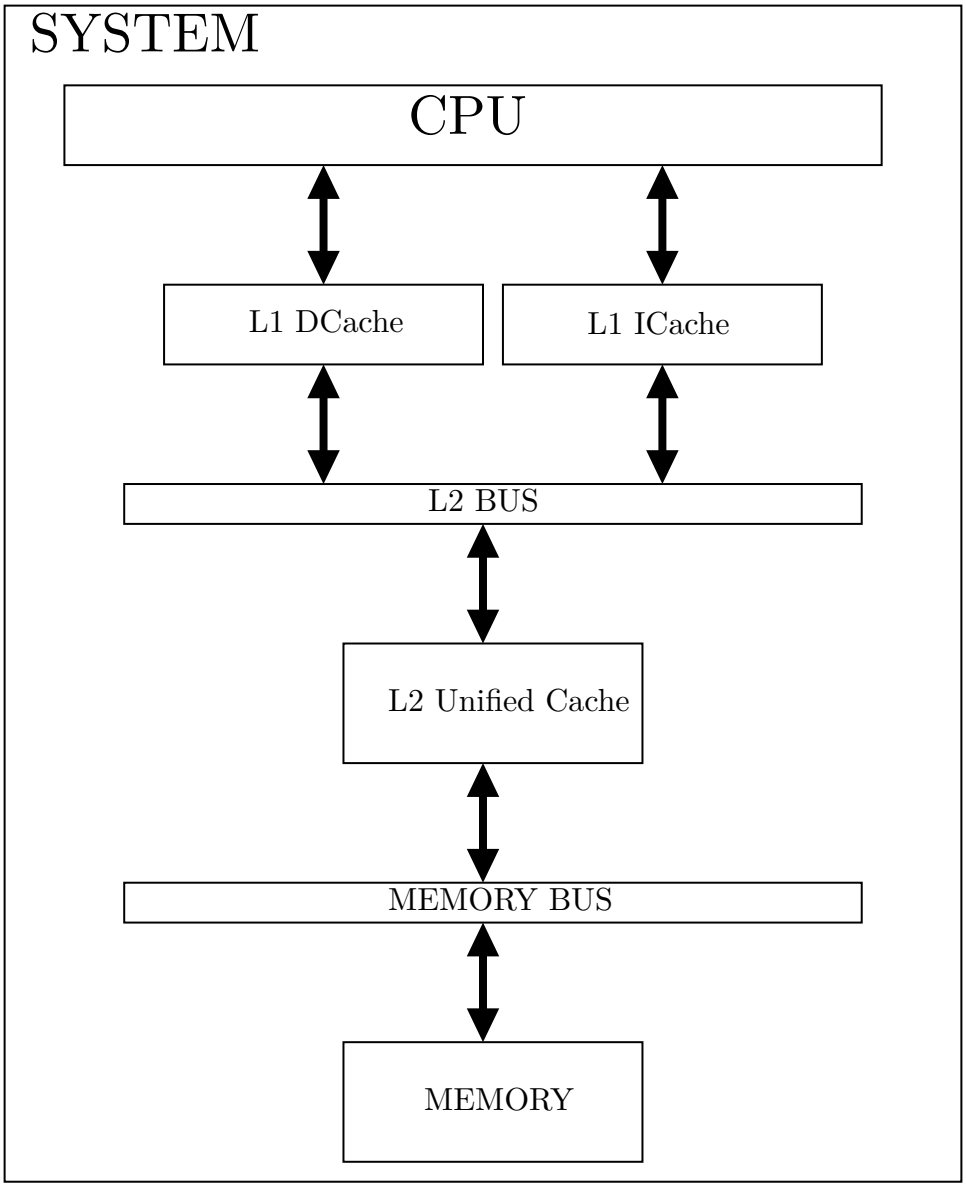
اثر پارامترهای مختلف بر کارایی برنامه را در حالت استفاده از دستور اختصاصی با تأخیر ۱۰۰ چرخهٔ ساعت بررسی می‌کنیم:

- بدون تغییرات: ۷۴۲,۸۴۳,۰۰۰ تیک (در این حالت associativity برابر ۶۴ است.)
- Cacheline:
  - ۳۲: ۸۷۳,۷۷۹,۰۰۰ ← بهبود = ۰٫۸۵۰
  - ۶۴: ۷۸۲,۸۴۳,۰۰۰؛ احتمالاً مقدار از پیش تعیین‌شدهٔ این پارامتر در سیستم همین مقدار است و به همین علت هیچ بهبودی مشاهده نمی‌شود.
- L2 Cache Associativity:
  - ۸: ۷۴۲,۸۴۳,۰۰۰
  - ۱۶: ۷۴۲,۸۴۳,۰۰۰

در واقع در اجراهای مختلف مشاهده شد که زمان اجرا به ازای مقادیر مختلف این پارامتر به غیر از ۱ با یکدیگر برابر هستند، که احتمالاً به علت کم بودن تعداد برخوردها در حافظهٔ نهان است. به طور کلی تقریباً همهٔ آمار و ارقام مربوط به دو مقدار متفاوت associativity با یکدیگر برابر بودند که احتمالاً به دلیل تعداد کم متغیرهای موجود در برنامه، و تکرار عملیات یکسان در طی اجرا است.

۵ ساختار اتصالات پیاده‌سازی شده در فایل‌های تنظیمات

ساختار کلی سیستم (اتصالات بین کش‌ها و مموری و پردازنده) که در فایل‌های اصلی تنظیمات ارسالی پیاده سازی شده به صورت زیر است:



۶ فایل‌های تحویل داده شده

در پوشه مربوط به این پروژه، چند فایل و یک پوشه به نام Stats قرار دارد. فایل sha256.cpp کد به زبان C++ الگوریتم SHA256 است که در بخش main آن، این الگوریتم روی رشته "how are you doing today?" اجرا شده است.

دو فایل caches.py و main_config.py مربوط به حالت توصیف شده در سوال هستند و در فایل cahces تنظیمات مربوط به cache ها قرار گرفته است و در فایل main_config.py تنظیمات کلی مربوط به سایر قطعات و در نهایت اتصالات آن‌ها قرار دارد. یک فایل دیگر هم به نام withoutcache.py وجود دارد که مربوط به تنظیمات بدون استفاده از حافظه نهان است.

درون پوشه Stats سه پوشه با نام‌های Phase1 و Phase2 و Phase3 قرار دارد. Phase1 مربوط به نتایج اجرای برنامه در اثر اضافه کردن دستور با سه تاخیر (۱ و ۲۰ و ۱۰۰ چرخه ساعت) گفته شده در صفحات قبل است. Phase2 مربوط به نتیجه اجرا با و بدون استفاده از کش است. Phase3 مربوط به تست کردن حالات مختلف cacheline و associativity که در مستند پروژه گفته شده بود است. درون هر کدام از پوشه‌ها، فایل‌های txt. خروجی‌های GEM5 قرار گرفته است.