تمرین تئوری هشتم

ساختار و زبان کامپیوتر

پاییز 1403

سیدمحمدرضا جوادی
402105868

Contents

سوال اول)

منبع این سوال https://www.quora.com/How-do-floating-point-operands-are-accessed-and-stored-within-the-x86-processor

بوده است که دارای سولات و جواب های بسیاری درباره پردازنده ها و تفاوت های آن ها با تمرکز بر اینتل است. همینطور:
https://www.quora.com/What-is-the-speed-difference-between-IBM-POWER8-servers-and-Intel-x86-64-servers-when-running-the-same-workload

میپس:

از استاندارد IEEE 754 پیروی می کند. در این استانداردی که پشتیبانی می کند با اختصاص 32 رجیستر32 بیتی، همزمان هم از single precision استفاده می کند و هم double . به دلیل کاربرد در نهفته، پردازنده floatآن جداست و دستورات جدایی برای این اختصاص دارد. این پردازش موازی و مجرای موازی سبب شده تا محاسبات های صحیح موازی اعشاری ها انجام شوند و خط مخصوص و جدای خود را دارند.

X86: و x87(در کل اینتل)

میپس جدای بقیه سیستم اعشاری خود را گسترش داده و به دلیل backward compatibility همان سیستم را ادامه داده برای ذخیره اعداد. در این سیستم ما دو نوع single و double به علاوه extended را داریم که عملا همه اعداد وقتی وارد می شوند به فرمت extendedدر می آیند.

IBM:

Ibm هم ابتدا فرمت خود را گسترش داد و بعدا به IEEE754 منتقل شداز سه سیستم زیر استفاده می کند:

base 16 S/390® hexadecimal format, base 2 IEEE-754 binary format, or base 10 IEEE-754 decimal format

خب حالا مقایسه:

دقت

هردوی میپس و اینتل به روش های گوناگون توانستند دقت را بالا ببرند، میپس با پشتیبانی از پروتکل بهینه و اینتل هم با سیستم سنتی که خود طی سال ها گسترش داده بود، هرچند در مواقعی سیستم اینتل در ضرب ها حاصل را به عدد بهتری گرد می کند (دقت بالاتر) ولی میپس با استفاده از پروتکل امن و شناخته شده امنیت بیشتری هنگام طراحی الگوریتم ها فراهم می کند. Ibm هم با وجود پشتیبانی از برد بزرگی از اعداد، دقت مناسبی دارد.

سرعت

بهینه سازی های اینتل آن را بسیار به میپس که از پردازش موازی استفاده می کند نزدیک می کند اما در مورد خاص گفته شده، اینتل همچنان از سیستم پشته برای محاسبات استفاده می کند که این باعث دسترسی کند به اعداد برای ضرب می شود،درحالی که میپس با پردازش موازی ضرب ها سریع تر است. Ibm به دلیل تفاوت های لایه های پایین تر از ISA و همینطور خود ISA سرعت به شدت بالایی دارد.

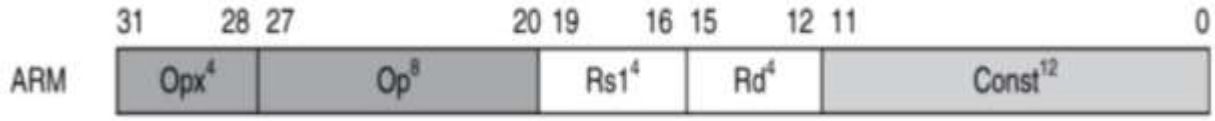
پیچیدگی

میپس یک RISC است. از همینجا این نکته واضح به نظر می رسد که به شدت ساده تر از دو رقیب خود است. اینتل هم با وجود هسته RISC همچنان به دلیل استفاده از استک سنتی خود کمی پیچیدگی هایی دارد. IBM که تماما CISC است و از روش های ذخیره سازی گوناگون در مبنا های مختلف پشتیبانی می کند پیچیدگی بسیار بالایی دارد.

سوال2)

منبع:https://developer.arm.com/documentation/den0042/latest/Unified-Assembly-Language-Instructions/Data-processing-operations/Multiplication-operations#:~:text=A%20key%20limitation%20is%20that,loaded%20into%20a%20register%20first.

Arm یک RISCاست بنابراین در صورت نیاز به قرار دادن یک alu جدا برای یک دستور، آن دستور را قرار نمی دهند. افزودن دستور گفته شده پیچیدگی هایی را در لایه سخت افزار اضافه می کند همینطور باعث می شود تا کدگذاری کنیم درصورتی که درحالت عادی immediate با مقدار بسیار بزرگ در دستورات جا نمی شود. پس هم محدودیت سخت افزار داریم، هم ISA و هم RISC ماندن معماری. تصویر زیر می تواند کمک کند که چرا فقط در 12 بیت می توانیم مقدار ثابت قرار دهیم.



در مورد دستورات گفته :

با یک لود و ضرب رجیستر ساده پیاده سازی می کنیم:

mov R5, #19

mul R5, R4, R5

سوال3)

نحوه ذخیره شدن عدد در 32 بیت:

1011 1101 1101 1001 1001 1001 1001 1001

7.7 را به باینری تبدیل کنیم:

در واقع 7.69999999 دارد ذخیره می شود. یعنی

111.101 1001 1001 1001 …

یک عدد، از 1 ها هم که دیفالت اضافه می شود یعنی:

0.11 101 1001 1001 1001 …

را نیاز داریم. دنبال این الگو را پیدا و کم کنیم، بیت اول هم که بیت علامت است را حذف کنیم چیزی که می ماند عد 29 است.

از آنجا که توان اکسپوننت باید 2 می بود، پس بایاس 27 داریم. x ذکر شده هم برابر6، 25 بیت هم y است.

سوال4)

الف)

برای اثبات اینکه دو عدد مثبت که هم خودشان در nبیت جا می شوند هم جمعشان که مسئله ای نداریم و از خواص بنیادین اعداد است.

A + B → A+B

در حالتی که مخالف علامت هم باشند بدیهی است چون(بدون کاستن از کلیت، فرض کنید A>0 و B<0 همینطور از جمع نمایش مکمل دوی آنها شروع می کنیم و به اینکه همان جمع عادی آنها است می رسیم. توجه داشته باشید فبش ها دوطرفه اند به دلیل محدودیت ها یکطرفه گذاشتم)

A + 2^n – |B| =2^n + (A-|B|) → B<0 → 2^n + A + B →

در نمایش nبیتی،2^n نمایش داده نمی شود و بیرون می رود

→ = A+B

برای حالتی که هردو منفی هستند هم با توجه به مطالب قبل راحت اثبات می شود:

2^n – |A| + 2^n – |B| = 2^(n+1) + A + B = A+B

که 2^(n+1) در n بیت جا نمی شود و بیرون می رود.

ب)

در مواقعی overflowداریم که حاصل در nبیت جا نشود. برای این امر باید A + B در (n-1) بیت جا نشود(کری داشته باشد). به عبارت دیگر گاهی وقت هایی که هردو مثبت و هردو منی اند اورفلو رخ می دهد. مثلا برای n=3:

-3-2=-5

101 + 110 = 011 → -3-2=3 !!!!!!

3 + 3 = 6

011 + 011 = 110 → 3+3 = -2 !!!!!!