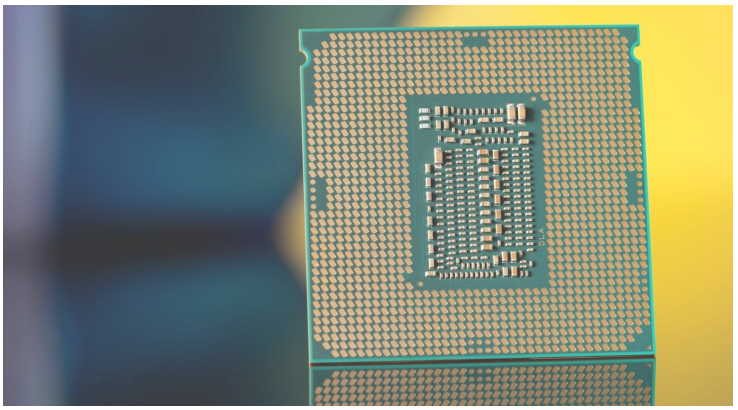
شبیه ساز پردازنده

پروژه پایان ترم درس مبانی کامپیوتر و برنامه سازی
ترم ۴۰۱۱

در این پروژه از مفاهیمی که در طول ترم آموخته اید استفاده خواهید کرد تا یک شبیه ساز تحت کنسول پیاده سازی کنید.



پردازنده یا **پروسسور** (به انگلیسی:Processor) تراشه ای است که می‌تواند عملیات حسابی و منطقی را انجام دهد. این تراشه‌ها از تعداد بسیار زیادی ترانزیستور ساخته شده‌اند که بر روی یک مدار مجتمع (یا بیشتر) پیاده‌سازی می‌شوند.

پردازنده قلب هر رایانه دستی یا رومیزی است که به عنوان واحد پردازشگر مرکزی شناخته می شود. پردازنده یک دستگاه محاسبه‌ای کامل است که روی یک تراشه واحد ساخته می‌شود و مجموعه دستورهای دستگاه را اجرا می‌کند. سه کار مهم را انجام می‌دهد: یکی اینکه از واحد همبستگی منطقی/ حساب، استفاده می‌کند یعنی کارهای وابسته به ریاضی چون جمع، تفریق، ضرب و تقسیم را انجام می‌دهد، دوم می‌تواند اطلاعات را از مکان یک حافظه به حافظه دیگر انتقال دهد و سوم اینکه می‌تواند تصمیم بگیرد و به یک سری از دستورهای جدید که بر اساس آن تصمیمات است، جهش کند.

هر پردازنده شامل تعدادی ثبات است. ثبات‌ها یا Registerهای یک پردازنده قسمتی از حافظه موقتی داخلی پردازنده هستند که وظیفه حفظ داده‌هایی را دارند که پردازنده درحال حاضر درحال کار بر روی آن‌هاست. به همین دلیل به نام

CPU Working Memory (حافظه کاری پردازنده) و CPU Workspace (میزکار پردازنده) نیز خوانده می‌شوند. ثبات‌ها بسیار سریع‌تر از سایر حافظه‌ها هستند؛ زیرا CPU همیشه درحال کار بر روی آن‌هاست و سرعت کم آن به همان اندازه باعث کاهش سرعت پردازنده می‌شود. ثبات‌ها با ذخیره کردن داده‌ها یا آدرس‌ها می‌توانند عمل پردازش را بسیار سریع‌تر کنند.

هر پردازنده دارای تعداد مشخصی ثبات (Register) است که بسیار سریع‌تر از سایر حافظه‌ها (برای مثال حافظه Cache و...) هستند، اما تعداد آن ها محدود است. برای مثال، در پردازنده Intel i7 که از معماری x86 و طراحی CISC استفاده می‌کند، تنها ۸ رجیستر در حالت ۳۲ بیت و ۱۶ رجیستر در حالت ۶۴ بیت قابل استفاده هستند.

حال برای پروژه پایان ترم درس مبانی، شما باید شبیه سازی برای یک پردازنده پیاده کنید.

پردازنده

پردازنده مفروض ما برای شبیه سازی پردازنده ای دارای ۳۲ ثبات ۴ بایتی قابل استفاده است که برای انجام دستورات دریافت شده از آن ها استفاده می کند. این ثبات ها به صورت S0, S1, S2, S3, …, S31 نام گذاری شده اند که محتویات هر یک از آن ها یک عدد صحیح می باشد.

پردازنده شامل چند ثبات غیر قابل استفاده نیز هست که یکی از آن ها ثبات وضعیت است. این ثبات یک ثبات یک بایتی است که پس از اجرای دستورات محاسباتی مقدار آن تغییر می کند. تغییر آن به این گونه است:

7	6	5	4	3	2	1	0

0 - parity flag

اگر در نمایش باینری حاصل محاسبات تعداد یک ها فرد باشد این بیت یک و در غیر این صورت صفر خواهد بود.

1 - zero flag

اگر حاصل محاسبات صفر باشد مقدار این بیت یک و در غیر این صورت صفر خواهد بود.

2 - sign flag

اگر حاصل محاسبات منفی باشد مقدار این بیت یک و در غیر این صورت صفر خواهد بود.

5 - overflow flag

اگر حاصل محاسبات overflow کند مقدار این بیت یک و در غیر این صورت صفر خواهد بود. منظور از overflow این است که عملیات منجر به عدد علامت داری شده است که برای نمایش در تعداد بیت معین خیلی کوچک یا خیلی بزرگ می باشد. به عنوان مثال اگر حاصل جمع دو عدد ۸ بیتی یک عدد ۹ بیتی شود عمل overflow رخ داده است.

نکته: در اینجا مقادیر بیت های ۳و ۴و ۶و ۷ برای ما بدون اهمیت هستند.

دستورات پردازنده

- **ADD S_{rd}, S_{rs}, S_{rt}**

این دستور مقدار ثبات های S_{rs} , S_{rt} را با هم جمع کرده و پاسخ آن را درون ثبات S_{rd} می ریزد.

- **SUB S_{rd}, S_{rs}, S_{rt}**

این دستور مقدار ثبات S_{rs} را از S_{rt} کم کرده و پاسخ آن را درون ثبات S_{rd} می ریزد.

- **AND S_{rd}, S_{rs}, S_{rt}**

این دستور مقدار ثبات های S_{rs} , S_{rt} را با هم به صورت بیتی and کرده و پاسخ آن را درون ثبات S_{rd} می ریزد.

- **XOR S_{rd}, S_{rs}, S_{rt}**

این دستور مقدار ثبات های S_{rs} , S_{rt} را با هم به صورت بیتی xor کرده و پاسخ آن را درون ثبات S_{rd} می ریزد.

- **OR S_{rd}, S_{rs}, S_{rt}**

این دستور مقدار ثبات های S_{rs} , S_{rt} را با هم به صورت بیتی or کرده و پاسخ آن را درون ثبات S_{rd} می ریزد.

- **ADDI S_{rt}, S_{rs}, Imm**

این دستور مقدار ثبات S_{rs} و عدد ثابت Imm را با هم جمع کرده و پاسخ آن را درون ثبات S_{rt} می ریزد.

نکته: در این دستور و در ادامه منظور از عدد ثابت*(immediate)* یک عدد صحیح ۴ بایتی است. مثالی از استفاده این دستورات را در ادامه می بینیم:

```
ADDI S0, S1, -5
```

- **SUBI S_{rt}, S_{rs}, Imm**

این دستور عدد ثابت Imm را از مقدار ثبات S_{rs} کم کرده و پاسخ آن را درون ثبات S_{rt} می ریزد.

- **ANDI S_{rt}, S_{rs}, Imm**

این دستور مقدار ثبات S_{rs} و عدد ثابت Imm را با هم صورت بیتی and کرده و پاسخ آن را درون ثبات S_{rt} می ریزد.

- **XORI S_{rt}, S_{rs}, Imm**

این دستور مقدار ثبات S_{rs} و عدد ثابت Imm را با هم به صورت بیتی xor کرده و پاسخ آن را درون ثبات S_{rt} می ریزد.

- **ORI S_{rt}, S_{rs}, Imm**

این دستور مقدار ثبات S_{rs} و عدد ثابت Imm را با هم به صورت بیتی or کرده و پاسخ آن را درون ثبات S_{rt} می ریزد.

- **MOV S_{rt}, S_{rs} یا MOV S_{rt}, Imm**

این دستور مقدار ثبات S_{rs} یا عدد ثابت Imm را در ثبات S_{rt} می ریزد.

- **SWP S_{rt}, S_{rs}**

این دستور مقدار دو ثبات S_{rt} و S_{rs} را با هم جا به جا می کند.

- **DUMP_REGS**

این دستور مقدار تمام ۳۲ ثبات عمومی و ثبات وضعیت را چاپ می کند.

- **DUMP_REGS_F**

این دستور مشابه دستور DUMP_REGS عمل می کند ولی به جای خروجی در کنسول خروجی خود را درون یک فایل با نام *regs.txt* می ریزد.

- **INPUT**

این دستور یک عدد صحیح از کاربر گرفته و مقدار آن را در ثبات S0 می ریزد.

- **OUTPUT**

این دستور مقدار موجود در ثبات S0 را در خروجی نمایش می دهد.

- **JMP Imm**

اگر دستورات موجود در فایل ورودی را با شروع از عدد ۱ شماره گذاری کنیم این دستور یک عدد به عنوان آرگومان گرفته و به دستور با شماره مد نظر پرش می کند.

مثال:

```
ADD S0, S1, S2
AND S1, S1, S3
JMP 6
SUB S0, S1, S4
OR S1, S1, S4
SWP S1, S2
JMP 2
```

در این مثال در دستور پرش اول به دستور ششم *(دستور SWP)* پرش می کنیم و دستورات چهارم و پنجم اجرا نمی شوند و در دستور پرش دوم به دستور دوم *(دستور AND)* پرش می کنیم.

- **EXIT**

این دستور به کار پردازنده پایان می دهد.

تاثیر دستورات مختلف بر ثبات وضعیت

Instructions	Parity flag	Zero flag	Sign flag	Overflow flag
ADD	✔	✔	✔	✔
SUB	✔	✔	✔	✔
AND	✔	✔	✔	
XOR	✔	✔	✔	
OR	✔	✔	✔	
ADDI	✔	✔	✔	✔
SUBI	✔	✔	✔	✔
ANDI	✔	✔	✔	
XORI	✔	✔	✔	
ORI	✔	✔	✔	
MOV				
SWP				
DUMP_REGS				
DUMP_REGS_F				
INPUT				
OUTPUT				
JMP				
EXIT				

اجرای برنامه

با اجرای برنامه پروژه شبیه ساز شما باید دستوراتی را که در فایل in.txt وارد شده اند را خوانده و رفتار پردازنده ذکر شده را به ازای آن دستورات شبیه سازی کند.

در ادامه مثالی از فایل ورودی (*in.txt*) را با هم مشاهده می کنیم. عملیات انجام شده توسط هر دستور برای درک بهتر شما به صورت comment در مقابل همان دستور تشریح داده شده است.

```
MOV S0, 4           // S0 = 4
MOV S1, 5           // S1 = 5
ADD S2, S1, S0      // S2 = S1 + S0 = 9
SUB S3, S1, S0      // S3 = S1 - S0 = 1
AND S4, S1, S0      // S4 = S1 and S0 = 4
XOR S5, S1, S0      // S5 = S1 xor S0 = 1
OR S6, S1, S0       // S6 = S1 or S0 = 5
ANDI S7, S1, 3      // S6 = S1 and 3 = 1
SWP S7, S6          // S6 = 1, S7 = 5
```

مدیریت خطا ها

برنامه شبیه ساز شما باید به گونه ای پیاده سازی شود تا در شرایط مختلف بدون مشکل به کار خود ادامه دهد. در این خصوص لازم است به نکات زیر توجه ویژه داشته باشید:

- در صورت ورود دستور اشتباه (دستوری به جز دستورات پیاده سازی شده یا دستوری با آرگومان های نادرست) برنامه شما باید پس از نمایش دادن پیغام خطای مناسب به دریافت دستورات بعدی ادامه دهد.
- هنگام کار با فایل ها در صورت بروز هرگونه خطا برنامه باید با نمایش یک پیغام مناسب به کار خود ادامه دهد.

پیاده سازی چند الگوریتم

در پایان چند الگوریتم ساده را با استفاده از دستورات ذکر شده پیاده سازی کنید و آن ها را با استفاده از شبیه ساز خود شبیه سازی کنید.

نکات پیاده سازی

در صورت عدم رعایت موارد زیر نمره این بخش را از دست خواهید داد:

- نامگذاری ها باید مطابق استانداردی باشد که در کارگاه ها آموخته اید و در **تمام قسمت های کد** رعایت شود.
- دندانه گذاری ها باید مطابق استاندارد باشد.
- تا حد امکان کد های تکراری را به توابع تبدیل کنید و از نوشتن کد های تکراری در پروژه خودداری کنید.
- برای توابع و متغیر ها اسامی با معنی انتخاب کنید.
- تمام پروژه باید با زبان برنامه نویسی C پیاده سازی شود. پیاده سازی پروژه با هر زبان دیگر همچون Java، Python، C++ و ... پذیرفته نیست و از دانشجو تحویل گرفته نمی شود. به عبارت دیگر پروژه شما حتما باید با استفاده از یک کامپایلر مختص زبان C (مثل gcc یا clang) کامپایل شود.
- اگر به مسئله ای بر خوردید، ابتدا در سایت های مختلف به دنبال راه حل بگردید، و فقط در صورتی که جوابتان را نیافتید به حل تمرین‌تان مراجعه کنید.(پیشرفت و یادگیری در تلاش برای دیباگ کردن شکل می‌گیرد و دیباگ کردن کدتان به عهده شماست!)
- در نهایت باید یک فایل زیپ در کوئرا آپلود کنید. این فایل زیپ باید حاوی کد پروژه و پیاده سازی چند الگوریتم ساده با استفاده از دستورات ذکر شده برای تست شبیه ساز شما باشد.
- ارائه و تحویل پروژه به صورت آنلاین، با وبکم و میکروفن و از طریق اسکایپ یا پلتفرم های مشابه می‌باشد که بعد از پایان ددلاین تاریخ آن مشخص خواهد شد.
- توصیه می شود کارتان را با موارد ساده شروع کرده و سپس به اضافه کردن ویژگی های جدید به کدتان بپردازید.

منابع

برای یادگیری یک سری از مفاهیم مورد نیاز برای پیاده سازی پروژه می توانید از منابع زیر استفاده کنید:

ویدئو آموزشی کار با فایل

ویدئو آموزشی عملگر های بیتی

موارد امتیازی اضافه

- استفاده از یک برنامه مدیریت نسخه (version control) مشابه git
- استفاده از Doxygen برای مستند کردن کد
- پیاده سازی امکان دریافت نام فایل ورودی به عنوان آرگومان که توسط کنسول به برنامه داده می شود.
- پیاده سازی دستورات زیر:

MULL S_{rt}, S_{rs}

این دستور ثبات های S_{rt} و S_{rs} را در هم ضرب کرده و ۴ بیت کم ارزش تر حاصل آن را در S_{rs} و ۴ بیت با ارزش تر را در S_{rt} می ریزد.

نکته: این دستور بر روی تمام بیت های ذکر شده از ثبات وضعیت تاثیر می گذارد.

DIV S_{rt}, S_{rs}

این دستور ثبات S_{rt} را بر S_{rs} تقسیم کرده، خارج قسمت را در S_{rt} و باقی مانده را در S_{rs} می ریزد.

نکته: با در نظر گرفتن خارج قسمت تقسیم، این دستور بر روی تمام بیت های ثبات وضعیت به جز *overflow flag* تاثیر می گذارد.

SKIE S_{rt}, S_{rs}

در این دستور اگر مقدار S_{rt} و S_{rs} برابر باشند دستور بعد از این دستور اجرا نمی شود.

- پیاده سازی stack

پردازنده علاوه بر ثبات های داخلی می تواند از حافظه رم برای ذخیره سازی استفاده کند. یکی از راه های این کار استفاده از *stack* است. برای آشنایی با *stack* می توانید از این لینک استفاده کنید.

پردازنده ازدو دستور زیر برای نوشتن و خواندن مقادیر بر روی *stack* استفاده می کند:

PUSH S_{rs}

این دستور مقدار *Srs* را روی *stack* قرار می دهد.

POP S_{rt}

این دستور مقدار روی *stack* را برداشته و در *Srt* قرار می دهد.

نکته: دو دستور *PUSH* و *POP* تاثیری بر روی ثبات وضعیت ندارند.

- عدم حساسیت شبیه ساز به بزرگی و کوچکی حروف دستورات
- پیاده سازی پشتیبانی از کامنت در دستورات
- خلاقیت اضافی و پیاده سازی امکاناتی به جز امکانات ذکر شده در مستند

موفق باشید :)