

تمرین ۲

درس ریزپردازنده

اعضا گروه: علی ذوالجلالی، محمد حسین طحانی

شماره گروه: ۹

سوالات تحلیلی

سوال (۱)

- Arm 32 : برنامه هایی که بیشترین کارایی را میخواهند داشته باشند و یا اکسپشن های سخت افزاری را میخواهند برطرف کنند کاربرد بیشتری دارند. همچنین در راه اندازی پردازنده هم کاربرد دارند.
- Arm 64: زمانی که پردازنده موردنظر روی aarch64 عمل میکند کاربرد دارد. همچنین دستورالعمل هایی ۶۴بیتی و مثال های مانند آن را دارند نیز استفاده میشوند.
- Thumb: برای انعطاف پذیری بیشتر در کارایی و اندازه کد کارایی دارد (متناسب با نیاز برنامه)
- Thumb2: در این مدل دستورالعمل های 32بیتی اضافه شده که میتوانند با دستورالعمل های ۱۶بیتی در برنامه ترکیب شوند

سوال (۲)

- زبان اسمبلی جزو زبان های سطح پایین و نزدیک به زبان ماشین میباشد. همین امر موجب شده تا نسبت به زبان های درجه بالا از سرعت اجرای بیشتری برخوردار باشد. همچنین این زبان نسبت به زبان های high lvl حافظه کمتری را اشغال میکنند.
- زبان اسمبلی به دلیل امکاناتی که دارد میتواند به طور مستقیم با میکروکنترلرها در ارتباط بود مثلا ram، io و سایر اجزا آنها.
- زبان اسمبلی نسبت به سایر زبان های سطح بالا از قوانین و محدودیت های کمتری برخوردار هست و کار را برای برنامه نویس برای طراحی یک میکروکنترلر آسانتر میکند

بخش عملی

قسمت اول)

برای سوال عملی اول ما باید با یک حلقه ساده اعداد یک تا ۸ را جمع کرده و در رجیستر R1 ذخیره کنیم
ابتدا عدد ۸ را با استفاده از دستور MOV به رجیستر R0 دادیم و ۰ را به R1.

سپس هر بار R1 را با R0 جمع میکنیم و در R1 میگذاریم. بعد از آن از R0 یکی کم میکنیم و پس از آن مقایسه با صفر میکنیم اگر ۰ بود از حلقه خراج میشویم.

```
1  AREA RESET, DATA, READONLY
2  DCD      0      ;initial_sp
3  DCD      Example;reset_vector
4
5  AREA      MyCode, CODE, READONLY
6  Example PROC
7  MOV R0,#8 ; COUNT=8
8  MOV R1,#0 ; SUM=0
9  FOR
10 ADD R1,R1,R0 ; R1+=R0
11 SUB R0,R0,#1 ; R0--
12 CMP R0,#0 ; moghayese ba 0
13 BNE FOR
14 BEQ ENDL
15
16 ENDL
17 B ENDL
18
19 END
20
```

در انتها جواب نهایی در همان رجیستر R1 خواهد بود

قسمت دوم)

در این بخش ابتدا ما عدد ۳۲ بیتی را به R0 می‌دهیم پس از یک حلقه با ۳۲ بار تکرار استفاده می‌کنیم.

ابتدا R0 را شیفت می‌دهیم به راست و کم ارزش ترین بیت را به عنوان کری ذخیره می‌کنیم.

سپس با استفاده از ADDCS,ADDCC یک بودن یا نبودن را ست

می‌کنیم. (ADDCC=>CARRY=0)(ADDCS=CARRY=1)

براساس اینکه ۱ بود یا نه رجیسترهای R1 و R2 را که به ترتیب برای نگه داشتن تعداد ۱ها و ۰ها هستند را اپدیت می‌کنیم.

در قدم بعد با استفاده از branch به تابع مورد نظر می‌رویم.

چون در ابتدای کد ما ادرس sp یا همان استک پوینتر را تعیین کرده ایم و به اندازه ۶ ورد برای آن در نظر گرفته ایم، کافی است تا از همان ادرس شروع به پوش کردن مقادیر مورد نظر در آن بکنیم.

در انتها برنامه هم برای پاپ کردن از استک می‌توانیم از دستور ldr استفاده کنیم که همان ترتیب که پوش کرده بودیم از آن پاپ می‌کنیم

نکته: با دادن مقدار LR به PC دیگر لازم نیست از دستور Branch برای برگشتن به برنامه اصلی استفاده کنیم

نکته: میتوان به جای دستورات str و ldr از push و pop استفاده کرد که همان به شکل decending استک را برایمان پر میکنند

منابع:

<https://developer.arm.com/architectures/instruction-sets/base-isas/a64>

<https://developer.arm.com/architectures/instruction-sets/base-isas/a32>

<https://www.quora.com/How-much-more-efficient-is-assembly-compared-to-a-high-level-programming-language>

<https://www.codeexplorer.com/2017/09/arm-assembly-code-to-find-number-of-ones-and-zeros-in-32-bit-number.html>

<https://developer.arm.com/documentation/dui0473/c/condition-codes/condition-code-suffixes>

https://www.keil.com/support/man/docs/armasm/armasm_dom13612_90023271.htm