درس ریزپردازنده

اعضا گروه: على ذو الجلالي، محمد حسين طحاني

شماره گروه: ۹

سوالات تحليلي

سوال1)

- 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

سوال۲)

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 میتوان به طور مستقیم با میکروکنترلرها درارتباط بود مثلا ram ،io

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 عملي

قسمت اول)

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

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

```
AREA RESET, DATA, READONLY
2
     DCD
                  0
                       ;initial sp
3
     DCD
                  Example; reset vector
4
             MyCode, CODE, READONLY
     AREA
6 Example PROC
    MOV R0,#8 ; COUNT=8
     MOV R1,#0 ; SUM=0
10
     ADD R1,R1,R0 ; R1+=R0
     SUB R0, R0, #1 ; R0--
11
     CMP R0,#0
                  ; moghayese ba 0
12
13
     BNE FOR
     BEQ ENDL
14
15
16 ENDL
     B ENDL
17
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 RO میدهیم پس از یک حلقه با ۳۲ بار تکرار استفاده میکنیم.

ابتدا RO 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 (ADDCS=CARRY=1)(ADDCC=>CARRY=0)

براساس اینکه ۱ بود یا نه رجیسترهایR1 و R2 را که به ترتیب برای نگه داشتن تعداد ۱ها و R1 هستند را اپدیت میکنیم.

درقدم بعد با استفاده از branch به تابع مورد نظر میرویم.

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

درانتها 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 او pop استفاده کرد که همان به شکل decending استک را برایمان پر میکنند

```
AREA RESET, DATA, READONLY
                              ; initial sp
                       Example; reset_vector
        AREA SA,DATA,READWRITE
                                 ; 6 word az stack ro baramun reserve mikone
 5 SS SPACE 0x18
       AREA program, CODE, READONLY
 8 Example
            LDR R5, =SS
            ADD R5, R5,#0x00000018 ; baraye handel krdn decending
MOV SP, R5 ; adress aval stack
                               ; adress aval stack
            MOV R0, #0x12
                                  ; input , dec:18 , bin:00010010
            MOV R3,#1
                                  ; counter for
    FOR1 CMP R3,#32
                                  ; 32 bar tekrar baraye 32 bit input
            BGT ENDFOR1 ; bastn for agar R3>32

MOVS R0,R0,LSR #1 ; logical shift right va set kardan carry(S)

ADDCS R1,R1,#1 ; R1++ agar carry=1 (CS) , R1 tedad 1

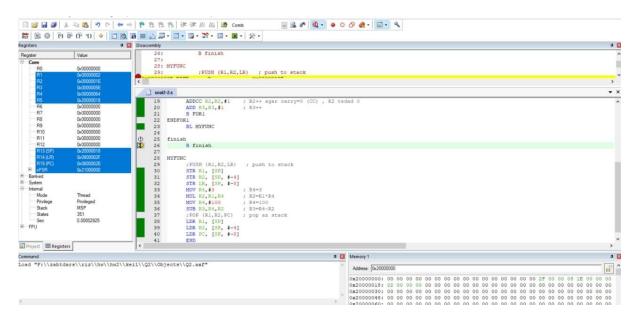
ADDCC R2,R2,#1 ; R2++ agar carry=0 (CC) , R2 tedad 0
19
                                  ; R3++
            ADD R3,R3,#1
            B FOR1
    ENDFOR1
           BL MYFUNC
24
25
    finish
            B finish
26
27
28 MYFUNC
            ; PUSH {R1,R2,LR} ; push to stack
            STR R1, [SP]
STR R2, [SP, #-4]
            STR LR, [SP, #-8]
            MOV R4,#3
                             ; R4=3
34
            MUL R2,R1,R4
                                  ; R2=R1*R4
            MOV R4, #100
                                   ; R4=100
            SUB R3,R4,R2
36
                                   ; R3=R4-R2
            ;POP {R1,R2,PC} ; pop az stack
            LDR R1, [SP]
LDR R2, [SP, #-4]
LDR PC, [SP, #-8]
39
40
41
```

درمثال بالا ورودی : ۱۸ یا همان ۲۰۰۱۰۰۱۰

تعداد ۱: ۲ / 2

1E / ۳۰ :۰ عداد

خروجی 9۴:R3 / خروجی



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-assemblycompared-to-a-high-level-programming-language

https://www.codesexplorer.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-code-suffixes

https://www.keil.com/support/man/docs/armasm/armasm_dom13612 90023271.htm