آزمایش 1

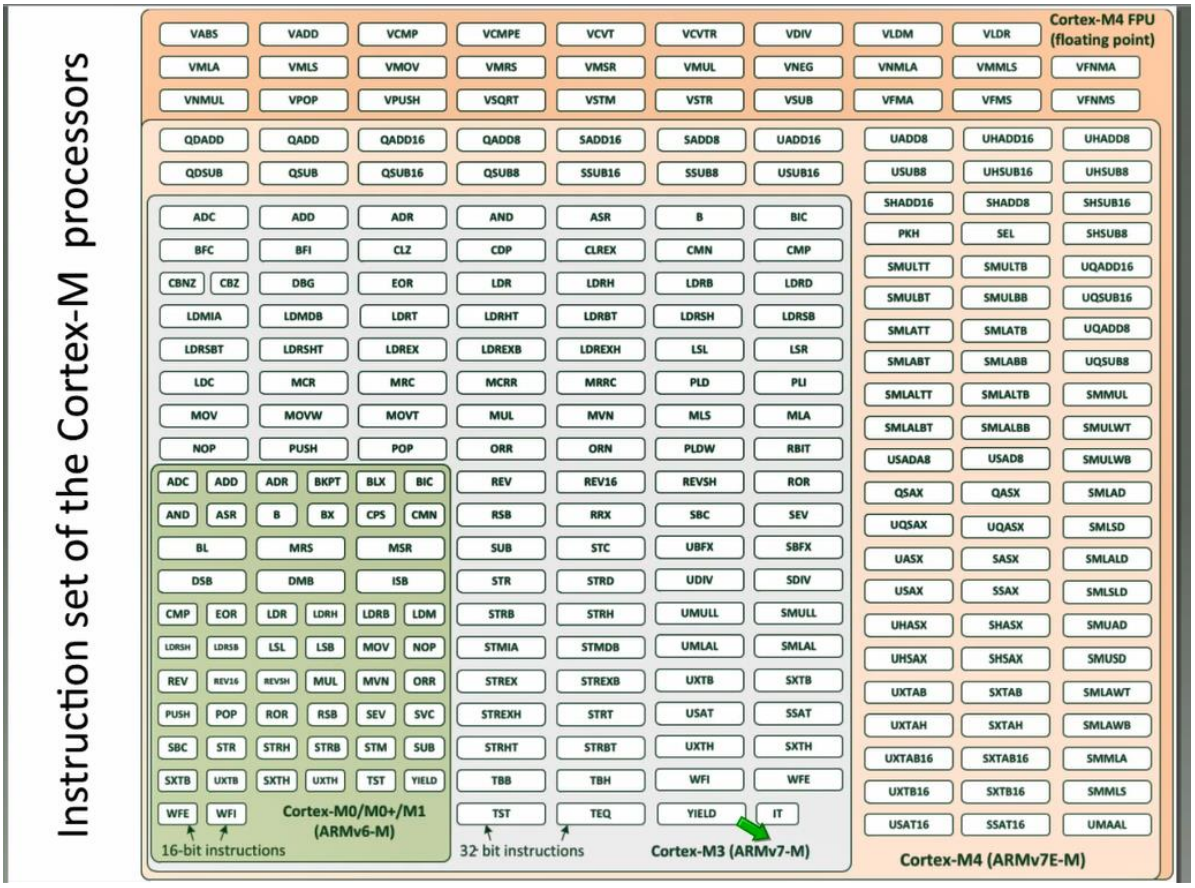
طاها موسوی 98243058

نیلوفر مرادی جم 97243063

گروه 2

آ. سوالات تحلیلی

1 - چهار مجموعه دستور العمل جهت برنامه نویسی میکروهای ARM را نام ببرید و کاربرد هر یک را توضیح دهید.



مجموعه دستورات پشتیبانی شده توسط هر پردازندهArm

1 – اینستراکشن ست A32:

در این مجموعه، دستورات 4 بایتی هستند. در این مجموعه اجراهای شرطی قابل اجرا هستند. همچنین برای هندل کردن اکسپن های سخت افزاری هم مناسب هستند. و به دلیل سرعت و عملکرد خوب نسبت به دیگر مجموعه ها، ازین مجموعه برای برنامه هایی که نیاز به سرعت بالایی دارند، استفاده می‌شود. اما به دلیل اینکه دستورات 4 بایتی هستند، حافظه بیشتری را مصرف می‌کنند.

2 – اینستراکشن ست Thumb:

همانطور که گفتیم در مجموعه قبلی حافظه زیادی مصرف می‌شد. به همین دلیل مجموعه دستورات Thumb ارائه شد که در آن دستورات 2 بایتی هستند. مزیت این معماری این است که سایز کد کمتر می‌شود و حافظه کمتری مصرف می‌کند. اما همین موضوع باعث یک سری محدودیت ها در دستورات را برای آن ایجاد می‌کند. و از طرفی اکثر اوقات فقط از low register(R0 –R7) ها استفاده می‌کند که باعث محدودیت در آدرس دهی می-شود. و در نهایت باعث می‌شود پرفورمنس و کارایی آن کمتر شود نسبت به مجموعه قبلی.

3 - اینستراکشن ست Thumb2:

در این مجموعه ضعف مجموعه Thumb به نوعی رفع شد و در آن علاوه بر دستورات 2 بایتی، از دستورات 4 بایتی هم استفاده شد. یعنی طول دستورات متفاوت است، به نوعی شبیه معماری سیسک عمل می‌کند.
از دستورات 4 بایتی می‌توان برای اجرای سریعتر و از دستورات 2 بایتی می‌توان برای سایز کد و توان کمتر استفاده کرد.
پس در مجموع می‌توان مجموعه دستوراتی با سرعت بالاتر نسبت به مجموعه دساورات قبلی داشت با توجه به اینکه سایز و توان هم بهینه بمانند.

4- مجموعه دستورات A64:

در این مجموعه، دستورات به صورت 8 بایتی هستند و می‌تواند مزیت باشد. و مانند مجموعه A32 از دستورات شرطی پشتیبانی می‌کند، اما نه به اندازه و کاملی آن. در این مجموعه تغییرات زیادی رخ داده است. از جمله اینکه دسترسی به آدرس های مجازی بیشتری شده. PC و SP رجیستر های عمومی نیستند. و از دستورات Load و Store چند رجیستری هم پشتیبانی می‌کند. و تعداد رجیستر های عمومی آن به 31 افزایش پیدا کرد.

منابع:

https://developer.arm.com/

https://courseware.sbu.ac.ir/courses/68823/files/534034/download?verifier=u2DlgGXbWrOh4aOol5N7Rt7Pu86Mry78IR00WzZr&wrap=1

2 – در حد دو پاراگراف مزیت استفاده از زبان اسمبلی نسبت به زبان سطح بالا را در کار با میکروکنترلرها را توضیح دهید.

همانطور که می‌دانیم، زبان های سطح بالا ابتدا به زبان اسمبلی تبدیل می‌شوند و همین موضوع می‌تواند در سرعت برنامه تاثیر بگذارد که در ادامه به مزیت های آن اشاره می‌کنیم. از طرفی زبان اسمبلی توانایی استفاده مستقیم و واقعی از رجیستر ها در میکروکنترلر ها یا هر سخت افزار هدفی را به برنامه نویس می‌دهد. ولی این امکان در زبان های سطح بالا وجود ندارد. همین موضوع باعث می‌شود برنامه ریزی برای میکروکنترلر ها راحت تر شود. از طرفی عملیات های مختلف و بیشتری در سطح پایین پشتیبانی می‌کند. و در مواقعی که برای یک میکروکنترلر خاصی برنامه می‌نویسیم، حجم کد کمتر می‌شود چون نیاز به دستورات اضافی برای دیگر میکروکنترلر ها نیست. و این یکی از دلایل بهینه تر بودن زبان اسمبلی نسبت به زبان های سطح بالاست.

پس زبان اسمبلی سرعت بیشتری دارد، به این دلیل که در کار کردن با منابع ضروری بهینه تر است. همینطور در استفاده کردن از حافظه بهینه تر عمل می‌کند. و در کنترل کردن عملیات سخت افزاری در سطح بالاتری نسبت به زبان های دیگر قرار دارد. همچنین زبان اسمبلی چون مستقیم به زبان ماشین تبدیل می‌شود، نیاز به کامپایلر ندارد تا کد را بهینه کند و باگ های برنامه راحت تر قابل شناسایی هستند و دیباگ می‌شوند.

منابع:

https://byjus.com/gate/difference-between-assembly-language-and-high-level-language/

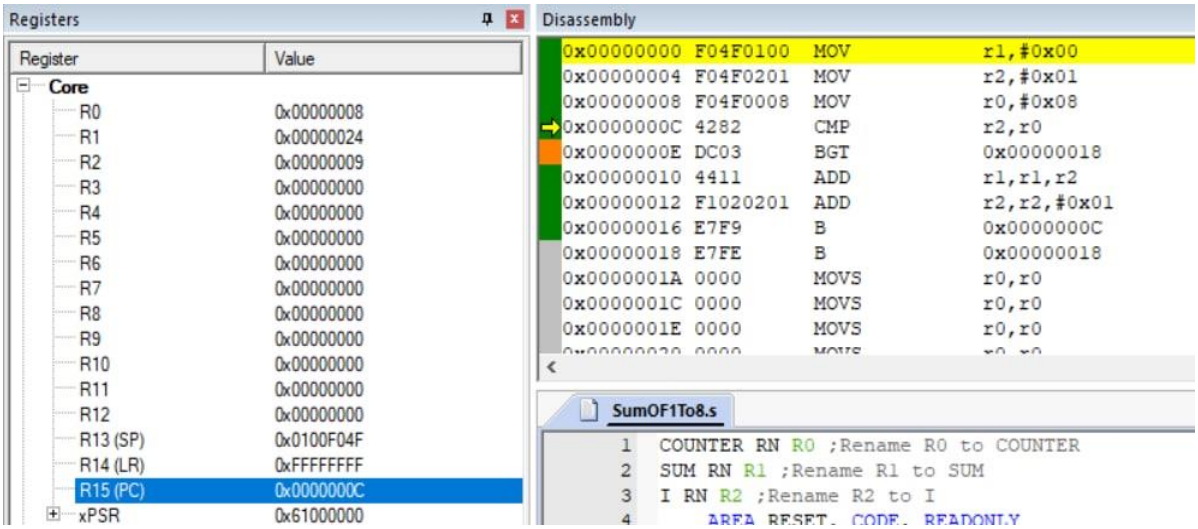
https://www.gadgetronicx.com/assembly-language-features-uses-advantages-disadvantages/

ب. سوال کدی اول

در این برنامه جمع اعداد 1 تا 8 را محاسبه می‌کنیم و در نهایت در رجیستر R1 ذخیره می‌کنیم.

```
1  COUNTER RN R0 ;Rename R0 to COUNTER
2  SUM RN R1 ;Rename R1 to SUM
3  I RN R2 ;Rename R2 to I
4      AREA RESET, CODE, READONLY
5      ENTRY
6      MOV SUM, #0x0 ;sum = 0
7      MOV I, #0x01 ; I = 1
8      MOV COUNTER, #0x08 ;COUNTER = 8
9
10 LOOP ; Start of loop
11     CMP I, COUNTER  ;Compare I and COUNTER
12     BGT HERE ; Jump to HERE ( end of programm ) if I > COUNTER
13     ADD SUM, I ; SUM = SUM + I
14     ADD I, #0x01 ; I = I + 1
15     B LOOP
16 HERE ; The value of the result of the addition operation is stored in SUM ( R1 )
17     B HERE ;stay here forever
18 END
```

ابتدا برای خوانایی کد رجیستر های R0، R1 و R2 را به ترتیب به COUNTER، SUM و I تغییر نام می‌دهیم.
سپس مقدار 0 را در SUM، 1 را در I (که شمارنده حلقه است) و 8 را در COUNTER طبق خواسته سوال ذخیره می‌کنیم.
سپس حلقه شروع می‌شود و ابتدا مقدار I و COUNTER را با هم مقایسه می‌کند و سپس در صورت بزرگتر شدن I از COUNTER از حلقه خارج می‌شود و به انتهای برنامه می‌رود. در غیر اینصورت مقدار I را به SUM اضافه می‌کند و مقدار I را هم یکی افزایش می‌دهد. و دوباره به ابتدای حلقه برمی‌گردد.



و در نهایت نتیجه دیباگ برنامه به این صورت می‌شود که مقدار I برابر 9 شده و حلقه به اتمام رسیده و مقدار R1 برابر 24 در مبنای 16 شده که برابر همان 36 در مبنای 10 است.

ج. سوال کدی دوم

در ابتدا نیاز است همانند توضیحات SP و Vector مقداردهی و ریست شوند. سپس ناحیه پشته به طول 6 کلمه در حافظه تعریف شود. به همین جهت AREA جدید با استفاده از برچسب SS با فضای 24 بایت (4بایت * 6 کلمه) تعریف میشود.

```
1      AREA RESET, DATA, READONLY
2          DCD           0        ;initial_sp
3          DCD           MAIN     ;reset_vector
4
5      AREA StackArea,DATA,READWRITE
6  SS SPACE 0x18                  ;Allocating 6 words for Stack
7
```

سپس در AREA کد , از آنجا که قرار است پشته به صورت Descending مقداردهی شود, نیاز است که پوینتر را به بالاترین آدرس پشته منتقل کنیم.

```
13
14      AREA MyCode, CODE,    READONLY
15
16 MAIN
17     LDR R5, =SS
18     ADD R5, R5, #24
19     MOV SP, R5
```

به R0 مقدار دلخواه می‌دهیم. اندازه ورودی 32 بیت میباشد و نیاز است که با یک حلقه, هرکدام از بیت ها را جهت صفر یا یک بودن, بررسی می‌کنیم .

```
20     MOV NUMBER, #1   ;Register Rd
21     MOV COUNTER, #0  ;Counter
22
23 WHILE
24     MOVS NUMBER, NUMBER, LSR #1    ;Logical right shift
25     ADDCS ONES, ONES, #1           ;ONES = number of ones
26     ADDCC ZEROES, ZEROES, #1       ;ZEROES = number of zeros
27     ADD COUNTER, COUNTER, #1       ;counter++
28     CMP COUNTER, #32
29     BNE WHILE
30     BL SUBROUTINE
31
```

در هر مرتبه تکرار حلقه, ابتدا ورودی یک واحد به سمت راست شیفت پیدا کند و فلگ ها ست شوند. در صورتی که Bit Carryبرابر با یک شود, آنگاه LSB برابر با یک بوده و تعداد یک ها زیاد شود. در غیر این صورت LSB برابر با صفر بوده است و تعداد صفر ها زیاد شود. و در هر بار اجرای حلقه نیز از مقدار شمارنده یک واحد اضافه میشود تا به 32 برسد و سپس از حلقه خارج می شویم. (منبع: https://www.codesexplorer.com/2017/09/arm-assembly-code-to-find-number-of-ones-and-zeros-in-32-bit-number.html)

بعد از محاسبه تعداد تکرار بیت های صفر و یک نیاز است سابروتینی اجرا شود. ابتدا با دستور BL، PC را به سابروتین می‌بریم. دستور BL ,LR که آدرس برگشت هستند آپدیت میکند. در ابتدای سابروتین نیاز است که مقادیر موجود در LR,R2,R1 وارد پشته شوند چرا که نیاز داریم مقدارهای اصلی آن ها بدون تغییر باقی بمانند. از آنجا که به صورت پیشفرض پشته به صورت Descending میباشد, تنها از دستور PUSH استفاده میکنیم در ادامه عملیات های خواسته شده انجام می دهیمدر نهایت بعد از انجام عملیات ها و انتقال خروجی روی R3 طبق خواسته سوال, نیاز است به بخش اصلی کد با مقادیر اصلی رجیسترها برگردیم. پس با همان ترتیب وارد کردن رجیسترها به پشته, آن ها را از پشته خارج میکنیم و مقدار LR که همان آدرس برگشت می باشد را به عنوان PCدر نظر می گیریم. دستورات POP و , PUSH آدرس SP را نیز آپدیت میکنند.

```
30    BL SUBROUTINE
31
32  HERE
33    B HERE ;stay here forever
34
35  SUBROUTINE
36    PUSH {ONES, ZEROES, LR}
37    MOV TEMP, #3
38    MUL ZEROES, ONES, TEMP
39    MOV TEMP, #100
40    SUB TEMP, ZEROES
41    MOV COUNTER, TEMP
42    POP {ONES, ZEROES, PC}
43
```