به نام خدا

گزارش کار آزمایشگاه ریزپردازنده

آزمایش ۱

مدرس : مهندس بی طالبی

تارا برقیان

مهرشاد سعادتی نیا

نیم سال اول ۰۱-۱۴۰۰

فهرست

مقدمه :

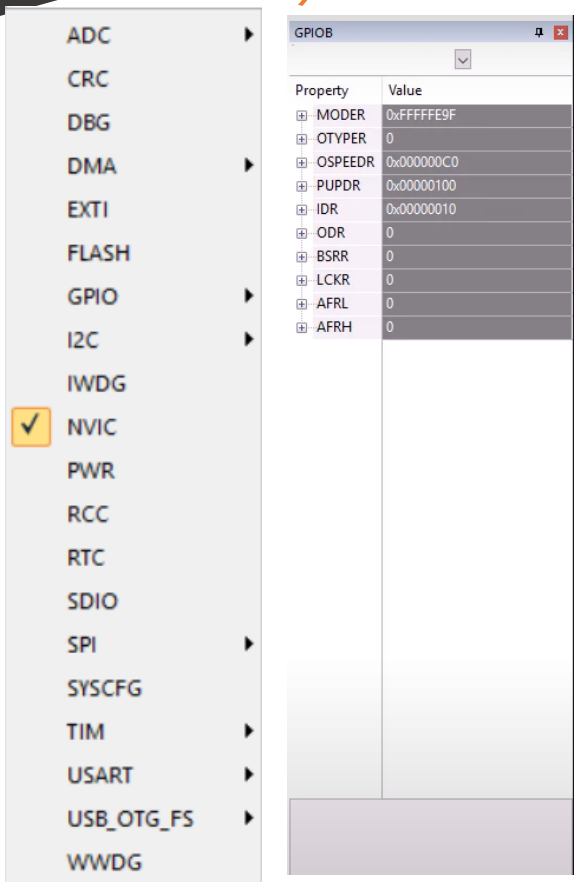
در این آزمایش هدف آشنایی با مفاهیم و دستوارت زبان اسمبلی برای دیوایس STM32F401Re بود. با محیط keil v5 از پیش آشنایی داشتیم و تحویل پیش آزمایش۱ بر روی محیط و ابزار ها بود.

پاسخ سوالات تحلیلی:

- CMSIS مخفف شده ی Cortex Microcontroller Software Interface Standard است. در واقع یک لایه‌ی نرم افزاری برای سخت افزار پردازنده های cortex M است و از محصولات ARM می باشد و کاملا فارغ از شرکت سازنده میکروکنترلر است. چون CMSIS برای همه ی میکروکنترلرهای این خانواده یکسان است، کمک میکند تا portability برنامه های نوشته شده بالاتر رود. برای مثال شما یک بار روی میکروی شرکت فیلیپس کد میزنید و با تغییرات کمی آن را به میکرو شرکت ST انتقال میدهید.
CMSIS بیشتر از #define های مختلف برای رجیسترها تشکیل شده و کمتر دارای توابع است. به صورت کلی میتوان گفت نقش اصلی آن اسم گذاری روی رجیسترهاست تا کار با آنها اسان شود.
اما HAL، مخفف Hardware Abstarction Layer است. HAL برخلاف CMSIS دارای توابع متنوعی برای کار کردن با قسمت های مختلف میکروکنترلر است. در واقع از تعاریف داخل CMSIS هم برای این توابع استفاده کرده است. این کتابخانه توسط شرکت STارائه شده است و در حال حاضر پشتیبانی خوبی هم میشود. برخی بر این باورند کتابخانه های ساده تر و قدیمی تر این شرکت مانند SPL بهتر بوده و در HAL توابع زیادی است که ممکن است به کار نیاید و این پردازش ار سنگین میکند. برای همین شرکت یک نسخه سبک تر از همین کتابخانه به نام LL منتشر کرده است که سرعت بالاتر دارد اما به راحتی HAL نخواهد بود.

- برای زمان هایی که با زبان C میکرو را پروگرم میکنیم میتوان از بخش peripherals -> system viewer آنرا فعال کرد که پنجره ای مطابق شکل زیر خواهد بود :

(به صفحه بعدی بروید.)



هر بخش آن امکانات مختلفی دارد که من از یک ویدیو در یوتیوب

کار با آن را به طور حدودی اموختم، برای دیدن ویدیو اینجا کلیک کنید .

Property	Value
MODER	0xFFFFFE9F
MODER15	0x03
MODER14	0x03
MODER13	0x03
MODER12	0x03
MODER11	0x03
MODER10	0x03
MODER9	0x03
MODER8	0x03
MODER7	0x03
MODER6	0x03

البته به طور کلی در پنجره رجیستر ها میتوان رجیسترهای اصلی و تغییراتشان را مشاهده کرد و بخش های دیگری مثل Banked,FPU و ... هم در آن وجود دارد.

- زبان c و کلا زبان های سطح بالا، به زبان انسان نزدیک ترند و قابلیت خوانایی بیشتری دارند، از طرفی در اکثر آنها مدیریت حافظه امکان پذیر نیست. از طرفی اسمبلی (زبان سطح پایین) سرعت بیشتری برای کدهای سخت افزاری دارد و در مقایسه با زبان های سطح بالا نیاز به تفسیر کمتری توسط کامپیوتر دارد.
در زبان برخی زبان های سطح بالا امروزه روتین های کد زدن مختلف مثل OPP و فانکشنال و ... به وجود امده و در پیرو آن دیزاین پترن هایی تعریف شده که همانگی برنامه نویسان و توسعه را راحت تر میکند.
یک نکته مهم این است که در برخی زبان های سطح بالا، نیازی نیست برنامه نویس به امنیت حافظه و داده هم فکر کند و اینکار توسط زبان انجام میشود ولی در اسمبلی خیلی باید دقت کرد.
در پاسخ به این سوال که کدام یک بهتر است، واقعا نمیتوان پاسخ قاطعانه ای داد چون هر یک کاربر، مزایا و ویژگی های به خصوص خود را دارد.

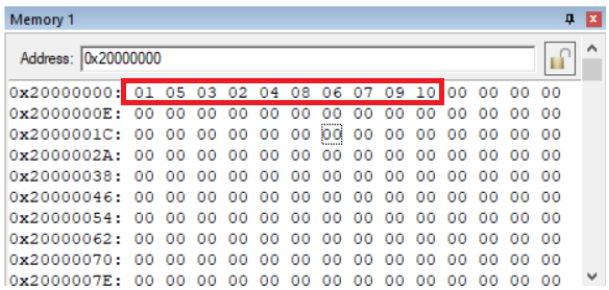
توضیح کدهای اسمبلی

فایل qa حاوی کد بخش a است.

بخش A :

در ابتدا بگویم که نرم افزار keil ، برای من و همگروهی ام کد هارا در حالت عادی ران نمیکرد بنابراین مجبور شدیم در ابتدا startup را فعال کنیم و داخل فایل c ساخته شده را به طور کلی پاک کنیم و از قالبی که در کد ها مشاهده میکنید برای کد زدن استفاده کنیم. توجه فرمایید کد اصلی در تابع main است.

در این سوال به طور رندوم از ادرس 0x20000000 حافظه استفاده کردم ولی هر جای دیگر هم میشود باشد. متاسفانه این قسمت تمامی اعداد ۰ بودند برای همین خودمان اعداد مختلف را به صورت دستی در ان قسمت از حافظه قرار دادیم که در تصویر مشاهده میکنید.




```
16      MOV R0, #11 ;counting
17      LDR R1, =0x20000000 ;start from
18
19      MOV R2, #0 ;R2=0
```

از R2 برای شمارش حلقه ی بیرونی استفاده کردیم و تعداد اولیه اعداد هم در R0 است. (البته یکی بیشتر قرار دادیم چون حلقه ی ما بازه ی باز میچرخد یعنی وقتی میگوییم ۱۱، ۱۰ بار میچرخد. میتوان هم الگوریتم را تغییر داد ولی ما این شیوه را انتخاب کردیم.)

این سوال در واقع عناصر را با بابل سورت مرتب میکند. در حلقه داخلی هم اعضای ارایه از اندیس ۰ تا n-i-1 پیمایش میشوند و مرتب میشوند. (دقیقا مانند الگوریتم بابل سورت)

قسمت مربوط به سورت کردن

```
    BGE OK ; sort them if they are not
    STRB R3, [R1, R6] ; put r3 in r4's position
    SUB  R6, #1
    STRB R4, [R1 ,R6] ; put r4 in r3's position
    ADD  R6, #1
OK
```

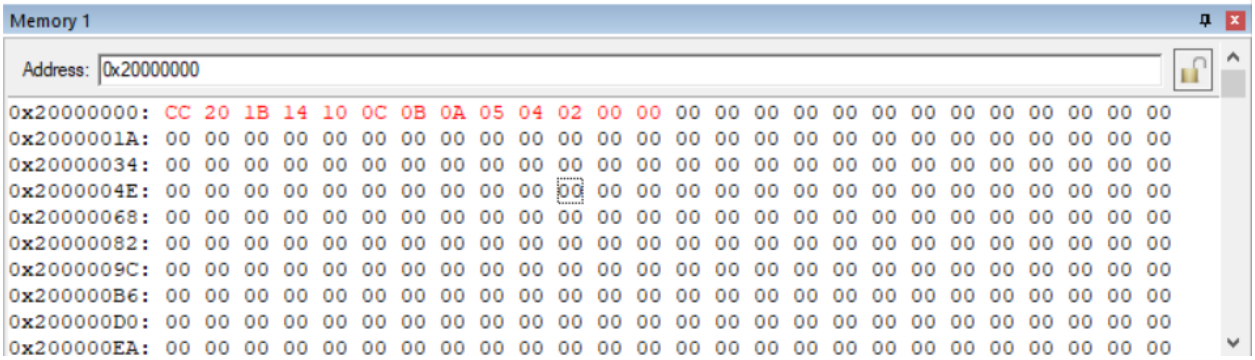
```
if arr[j] > arr[j+1] :
    arr[j], arr[j+1] = arr[j+1], arr[j]
```

```
FOR1
    SUB R5, R0,R2
    MOV R6 , #0
FOR2

    LDRB R3, [R1, R6] ;R3 arr[j]
    ADD  R6, #1 ;j
    LDRB R4, [R1, R6] ;R4 arr[j+1]
    CMP  R3,  R4

    BGE OK ; sort them if they are not
    STRB R3, [R1, R6] ; put r3 in r4's position
    SUB  R6, #1
    STRB R4, [R1 ,R6] ; put r4 in r3's position
    ADD  R6, #1
OK

    CMP R6, R5
    BLE FOR2

    ADD R2,#1
    CMP R2,R0
    BLE FOR1
```

```
# Traverse through all array elements
for i in range(n):

    # Last i elements are already in place
    for j in range(0, n-i-1):

        # traverse the array from 0 to n-i-1
        # Swap if the element found is greater
        # than the next element
        if arr[j] > arr[j+1] :
            arr[j], arr[j+1] = arr[j+1], arr[j]
```

در اخر یک حلقه بینهایت مثل همیشه در انتهای کد قرار دارد. برای تمامی خطوط مهم کد هم کامنت مناسب وجود دارد.

نتیجه یک بار اجرا با اعداد متفاوت و رندم (۸بیتی)



بخش B :

در این بخش یک مشکل بزرگ برای دسترسی به Area ی ReadWrite وجود داشت و از همه پرس و جو کردیم و تقریبا همه این مشکل را داشتند. آن هم این بود که وقتی هم برای خواندن و هم نوشتن سراغ این ادرس میرفت متاسفانه فقط ۰ میخواند ولی

نوشتن مشکلی نداشت. لذا برای حل این مشکل تصمیم گرفتیم کد هر بخش را جدا بزنیم البته در فایل all.s همه روتین ها در یک برنامه موجود است.

فایل های qb1,qb2,qb3 به ترتیب سوالات ۱و۲و۳ این بخش هستند.

: FindMin

مقداردهی های اولیه قبل از فراخوانی تابع :

```
MOV R0, #8
LDR R1, =MYDATA
LDRB R2, [R1]
```

برای اینکه کد متوجه شود چه زمان تابع تمام میشود در ابتدا LR را داخل استک میریزیم و در انتها پاپ میکنیم بنابراین بعد از اتمام تابع، PC به خط بعد از فراخانی اشاره میکند.

برای پیدا کردن min ابتدا اولین عدد از آرایه اعدادمان را در یک متغیر نگهداری میکنیم سپس با کمک یک حلقه تا اخر اعداد را پیمایش میکنیم. هر عددی که از مقدار فعلی min کمتر باشد ، داخل min ریخته میشود و اپدیت میشود.

```
FINDMIN
    PUSH {LR}
FOR
    LDRB R3, [R2]
    CMP R1,R3

    BLS CONTINUE;update min if current number is less than it
    MOV R1,R3

CONTINUE
    ADD  R2, #0x01

    SUBS R0,#1
    BNE FOR

    POP {PC} ;end of function
```

یک نمونه ورودی و خروجی :

```
 ENDFUNC

    AREA DATA_1, DATA, READONLY
MYDATA  DCB 0XD5, 0XC5, 0XA5, 0X90, 0X25, 0X20, 0X55, 0X02 , 0X25, 0X20, 0X88
```

Memory 1

Address: 0x08000204

```
0x08000204: D5 C5 A5 90 25 20 55 02 25 20 88 0C
0x0800021E: 00 00 00 00 00 00 00 00 00 00 00 0C
0x08000238: 00 00 00 00 00 00 00 00 00 00 00 0C
0x08000252: 00 00 00 00 00 00 00 00 00 00 00 0C
0x0800026C: 00 00 00 00 00 00 00 00 00 00 00 0C
0x08000286: 00 00 00 00 00 00 00 00 00 00 00 0C
```

Call Stack + Locals | Memory 1

Registers

Register	Value
Core	
R0	0x00000000
R1	0x00000002
R2	0x0800020E
R3	0x00000020
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000600
R14 (LR)	0x080001E9
R15 (PC)	0x080001E8
xPSR	0x61000000

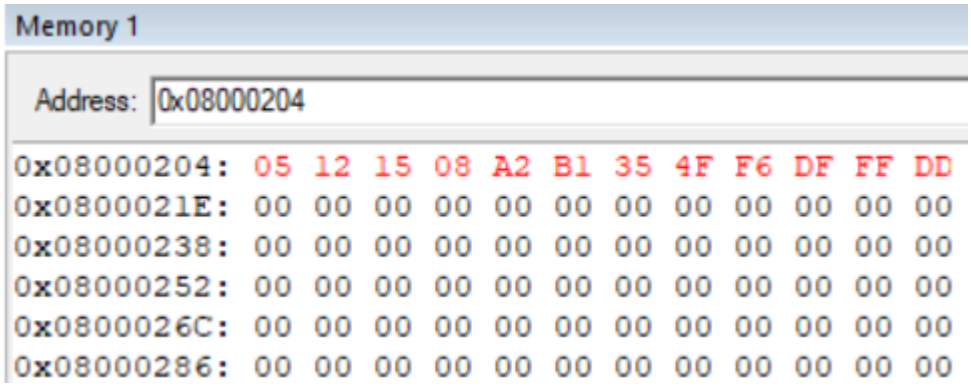
:FindMax

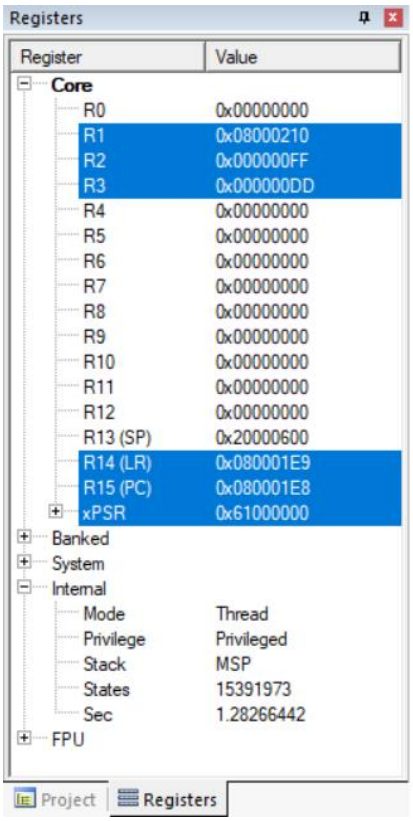
دقیقا مانند الگوریتم سوال قبلی ، فقط در هر پیمایش اگر عدد فعلی از max بزرگتر باشد جایگزین میشود.

```
FINDMAX
    PUSH {LR}
FOR
    LDRB R3, [R1]
    CMP R2,R3

    BHS CONTINUE ;update max if current number is bigger than it
    MOV R2,R3

CONTINUE
    ADD  R1, #0x01
    SUBS R0,#1
    BNE FOR
    POP {PC}
```

```
    AREA DATA_1, DATA, READONLY
MYDATA  DCB 0X05, 0X12, 0X15, 0X08, 0XA2, 0XB1, 0X35, 0X4F , 0XF6, 0XDF, 0XFF, 0XDD
```

یک نمونه ورودی و خروجی :





Population:

ابتدا در انتهای کد یک بخش برای READWRITE قرار میدهیم.

```
47      AREA DATA_1, DATA, READWRITE
48  MYDATA
49
```

مقدار دهی اولیه :

برای مثال با این مقدار دهی میتوان ۲۰ عدد فرد از ۱ به بعد را پیدا کرد.

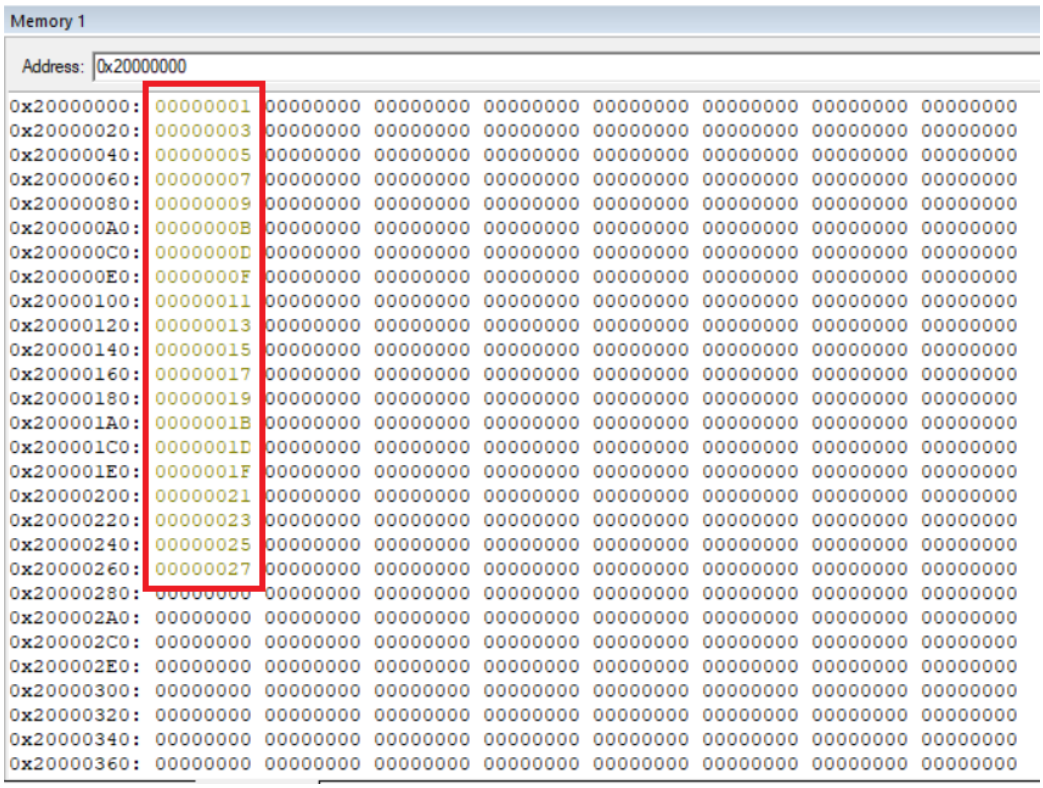
```
LDR   R0, =MYDATA
MOV   R3, #20
MOV   R4, #1
MOV   R5, #2
```

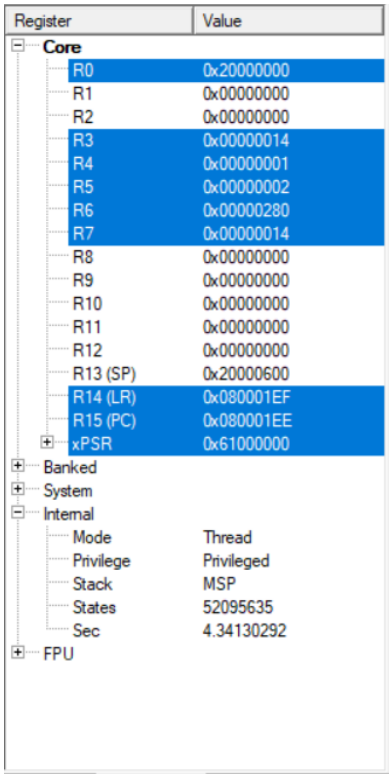
در این بخش، در هر بار چرخش حلقه یک واحد b به عدد فعلی اضافه میشود. میتوانستیم با mul هم پیاده سازی کنیم اما به نظرمان استفاده از add بهتر و سریعتر بود و نیاز به متغیر ها و دستورات اضافه تری نداشت.

```
POPULATION
    PUSH {R4,R5,LR} ;to be sure we dont loose r4 & r5
    MOV R6, #0
    MOV R7, #0 ;COUNTER

FOR
    STR R4, [R0,R6] ;storing r4
    ADD R6, #0X20   ;to go next line and show number in beautiful format ^_^
    ADD R4, R4, R5  ;add another 'b' to r4 (actualy i++)
    ADD R7, #1  ;counetr to control loop
    CMP R7, R3

    BLT FOR

    POP {R4,R5,PC}
```

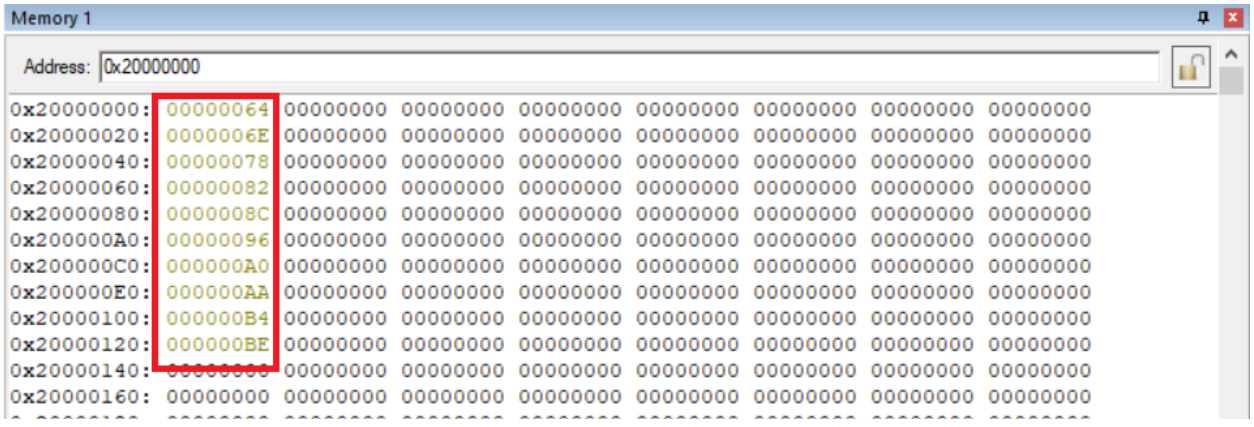
چند نمونه ورودی و خروجی در صفحه بعدی مشاهده میکنید :

۱- تولید اعداد فرد.
۲- شمارش اعداد مضرب ۱۰ از ۱۰۰ تا ۲۰۰


Memory 1
Address: 0x20000000
0x20000000: 00000001 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000020: 00000003 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000040: 00000005 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000060: 00000007 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000080: 00000009 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x200000A0: 0000000B 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x200000C0: 0000000D 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x200000E0: 0000000F 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000100: 00000011 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000120: 00000013 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000140: 00000015 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000160: 00000017 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000180: 00000019 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x200001A0: 0000001B 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x200001C0: 0000001D 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x200001E0: 0000001F 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000200: 00000021 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000220: 00000023 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000240: 00000025 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000260: 00000027 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000280: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x200002A0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x200002C0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x200002E0: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000300: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000320: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000340: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000360: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000


Register
Value
Core
R0 0x20000000
R1 0x00000000
R2 0x00000000
R3 0x00000014
R4 0x00000001
R5 0x00000002
R6 0x00000280
R7 0x00000014
R8 0x00000000
R9 0x00000000
R10 0x00000000
R11 0x00000000
R12 0x00000000
R13 (SP) 0x20000600
R14 (LR) 0x080001EF
R15 (PC) 0x080001EE
xPSR 0x61000000
Banked
System
Internal
Mode Thread
Privilege Privileged
Stack MSP
States 52095635
Sec 4.34130292
FPU

مثال دوم :


Memory 1
Address: 0x20000000
0x20000000: 00000064 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000020: 0000006E 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000040: 00000078 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000060: 00000082 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000080: 0000008C 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x200000A0: 00000096 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x200000C0: 000000A0 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x200000E0: 000000AA 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000100: 000000B4 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000120: 000000BE 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000140: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x20000160: 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000