

12/25/2020



---

## Homework 6

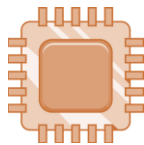
Lec 22-25

---



MICROPROCESSOR  
AND  
ASSEMBLY LANGUAGE

Fall 2021



(1) فرض کنید وضعیت حافظه و رجیسترها به شکل زیر باشد:

آدرس حافظه

0x8010	0x00000001
0x800C	0xFEEDDEAF
0x8008	0x00008888
0x8004	0x12340000
0x8000	0xBABE0000

رجیستر

0x13	R0
0xFFFFFFFF	R1
0xEEEEEEEE	R2
0x8000	R3

پس از اجرای دستور زیر وضعیت و محتوای حافظه و رجیسترها بکشید و دلیل آن را توضیح دهید.  
LDMIA R3!, {R0, R1, R2}

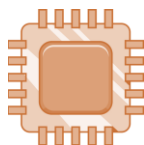
**جواب:**

این دستور همان LDMFD میباشد. که مخفف LOAD MULTIPLE FULL DECSENDING میباشد. یعنی استک پوینتر به محل آخرین داده ی استک اشاره میکند و در صورت push کردن مقدار استک پوینتر کاهش می یابد.

در اینجا LDMIA R3! همان POP به شیوه ی FULL DECSENDING میباشد. و محتوای سه خانه ی حافظه یعنی 0x8000, 0x8004, 0x8008 خوانده میشود و در R0, R1, R2 قرار میگیرد. و سپس مقدار استک پوینتر که در اینجا در رجیستر R3 میباشد به یک خانه بعد تر از این سه خانه ی POP شده یعنی 0x8010 اشاره میکند.

مقادیر بدینصورت میباشند:

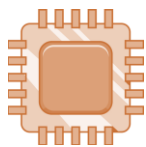
R0 = 0xBABE0000  
R1 = 0x12340000  
R2 = 0xFEEDDEAF  
R3 = 0x00008010



(2) برنامه ای به زبان اسمبلی بنویسید که نشان دهد یک عدد دلخواه اول است یا خیر. (عدد دلخواه را در رجیستر R0 قرار دهید. همچنین برای مشخص کردن اول نبودن عدد 0x00000000 و برای اول بودن عدد 0x11111111 را در رجیستر R3 بریزید)

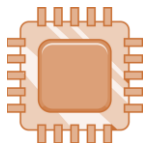
جواب:

```
1      AREA myData, DATA
2      number_const EQU 127
3
4      number RN r0;
5      i      RN r1;
6      result RN r2;
7      TMP    RN r3;
8      TMP_2   RN r4;
9      EXPORT __main
10     AREA myCode, CODE, READONLY
11     ENTRY
12
13     __main
14     LDR number, =number_const
15     LDR result, =0x00000000
16     LDR i, =2;
17     loop
18     CMP i, number
19     BLO inside_loop
20     B HERE
21     inside_loop
22     ;inside loop check that "remainder of i on number" is zero or
23     UDIV TMP, number, i;
24     MUL TMP, TMP, i;
25     CMP TMP, number;
26
27     BEQ composite_section; number is composit;
28     ; or
29     ADDS i, i, #1;
30     B loop;repeat the loop
31
32     composite_section
33     LDR result, =0x11111111;
34     B HERE
35     HERE B HERE;
36     END
37
```



3) عددی را در خانه 0x05000000 ثبت کنید. برنامه ای بنویسید که آن را تقسیم بر توان های 2، از یک تا ده کند و آن را در ده رجیستر اول بریزد.

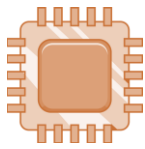
```
1      AREA myData, DATA
2  mem_addr_const equ 0x05000000;
3  value_to_write RN r10;
4  mem_addr      RN r11;
5
6      EXPORT __main
7      AREA myCode, CODE, READONLY
8      ENTRY
9
10     __main
11     LDR mem_addr, =mem_addr_const;
12     LDR value_to_write, =256;
13
14     STR value_to_write, [mem_addr]; store to mem
15
16     MOV r0, value_to_write, LSR #1;
17     MOV r1, value_to_write, LSR #2;
18     MOV r2, value_to_write, LSR #3;
19     MOV r3, value_to_write, LSR #4;
20     MOV r4, value_to_write, LSR #5;
21     MOV r5, value_to_write, LSR #6;
22     MOV r6, value_to_write, LSR #7;
23     MOV r7, value_to_write, LSR #8;
24     MOV r8, value_to_write, LSR #9;
25     MOV r9, value_to_write, LSR #10;
26
27     HERE B HERE;
28     END
29
```



4) قطعه کد اسمبلی معادل با کد C زیر را بنویسید  
(الف)

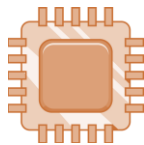
```
for (R0 = 0 ; R0 < 10; R0++){  
    if (R1 == 0) {  
        R2++;  
    }  
}
```

```
1      AREA myData, DATA  
2      number_const EQU 10  
3  
4      number RN r3;  
5      i      RN r4;  
6  
7      EXPORT __main  
8      AREA myCode, CODE, READONLY  
9      ENTRY  
10  
11     __main  
12     LDR number, =number_const  
13  
14     LDR i, =0;  
15  
16     loop  
17         CMP i, number  
18         BLO inside_loop  
19         B HERE  
20     inside_loop  
21  
22         CMP r1, #0;  
23         BNE out_loop  
24         ADD r2, r2, #1;  
25  
26     out_loop  
27         ADDS i, i, #1;  
28         B loop;  
29  
30     HERE B HERE;  
31     END
```



```
int *ptr;  
int sum = 0;  
for (int i = 0; i < 20; i++)  
    sum += *(ptr++);
```

```
1      AREA myData, DATA  
2  number_const EQU 20  
3  
4  ptr RN r0;  
5  sum RN r1;  
6  i   RN r2;  
7  mem_val RN r4;  
8  number  RN r5;  
9  
10     EXPORT __main  
11     AREA myCode, CODE, READONLY  
12     ENTRY  
13  
14     __main  
15     LDR i, =0;  
16     LDR sum, =0;  
17     LDR ptr, =0x05000000;  
18     LDR number, =number_const;  
19  
20     loop  
21         CMP i, number  
22         BLO inside_loop  
23         B HERE  
24     inside_loop  
25         ;because ++ of ptr is postfix, first  
26         ;read the value from memory then increment  
27         ; ptr by one. so ptr will point  
28         ;to the next byte of memory  
29         LDRB mem_val, [ptr];  
30         ADD ptr, ptr, #1;points to the next byte  
31  
32         ADD sum, sum, mem_val;  
33  
34         ADDS i, i, #1;  
35         B loop;  
36  
37     HERE B HERE;
```



5) در هر بخش آدرس خانه حافظه‌ای که به آن اشاره می‌شود را بدست آورید و رجیستری که با علامت سوال مشخص شده است را در هر مورد بنویسید. مراحل کار خود را توضیح دهید.

در همه موارد فرض کنید:  $R5=0x4000$ ,  $R4=0x20$   
همچنین هر جا نیاز به خواندن خانه‌ای از حافظه را داشتید مقدار آن را  $0xFF$  فرض کنید.

- a. `LDR R9, =0x11223344`  
`STRH R9, [R3, R4]`

a) مقدار در  $r9$  نوشته می‌شود. سپس به اندازه 2 بایت کم ارزش از  $r9$  در خانه ی حافظه نوشته می‌شود. آدرس خانه ی حافظه برابر با مجموع مقادیر  $r3, r4$  می‌باشد.

- b. `LDRB R8, [R3, R4, LSL #3]; R8 = ?`

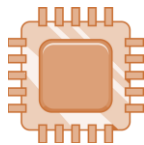
b) در اینجا مقدار مقدار  $r4$  با 3 بار شیفت به چپ با مقدار  $r3$  جمع می‌شود و سپس مقدار این خانه به اندازه یک بایت - کم ارزش تر- خوانده شده و در  $r8$  قرار می‌گیرد. در انتها مقدار  $r8$  برابر با  $0xFF$  می‌باشد. آدرس خانه ی حافظه برابر است با جمع  $r3$  و 8 برابر  $r4$ .

- c. `LDR R7, [R3], R4; R7 = ?, R3 = ?`

c) در اینجا مقدار خانه ی حافظه به آدرس  $r3$  خوانده می‌شود و در  $r7$  قرار می‌گیرد. سپس مقدار  $r3$  با  $r7$  جمع می‌شود و در  $r3$  قرار می‌گیرد.

- d. `LDR R6, =0x11223344`  
`STRB R6, [R3], R4, ASR #2, R3 = ?`

d) در اینجا مقدار رجیستر  $r6$  به اندازه ی یک بایت در خانه ی حافظه به آدرس  $r3$  قرار داده می‌شود. سپس مقدار  $r3$  با ( $r3$  تقسیم بر 4) جمع می‌شود و در  $R3$  قرار می‌گیرد.

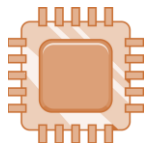


(6) 4 مورد از قوانین استاندارد AAPCS برای پیاده‌سازی توابع را نام ببرید.

**جواب:**

- 1- آرگومان های تابع باید از طریق رجیستر های r0 تا r3 ارسال شوند.
- 2- مقدار return شده باید در رجیستر r0 قرار بگیرد. اگر مقدار 64 بایتی باشد، 32 بیت دیگر باید در رجیستر r1 قرار گیرد.
- 3- تابع ها میتوانند از r4 تا r8 و r10, r11 برای اصطلاحاً متغیر لوکال خود استفاده نمایند. البته باید توجه داشت که تابع ها باید مقادیر این رجیستر ها را در صورتی که میخواهند از آنها استفاده کنند در ابتدا در استک ذخیره کنند و پس از تمام شدن کار آن ها با این رجیسترها، مقادیر اولیه ی این رجیستر ها را بازیابی کنند.
- 4- استک باید به شیوه ی full descending مورد استفاده قرار بگیرد. به این معنی که هنگام عملیات push مقدار استک پویتر کاهش می یابد و همچنین استک پویتر همواره به آخرین داده ی موجود در استک اشاره میکند.





- مهلت ارسال تمرین ساعت 23.55 **روز جمعه هفدهم دی ماه** است
- سوالات خود را می‌توانید از طریق تلگرام از تدریس‌یاران گروه خود بپرسید
- کد های اسمبلی را با استفاده از keil انجام دهید
- ارائه پاسخ تمرین بهتر است به روش‌های زیر انجام شود.
- 1) ارائه اسکرین شات از کد و نتیجه اجرای آن در یک فایل pdf
- 2) قرار دادن فایل کد و اسکرین شات از نتیجه اجرای کد. در صورت استفاده از این روش حتما هر سوال را در پوشه جداگانه قرار دهید.
- فایل پاسخ تمرین را تنها با قالب **HW6-9731\*\*\*.zip** یا **HW6-9731\*\*\*.pdf** در مدل بارگزاری کنید.
- نمونه HW6-9731097.pdf