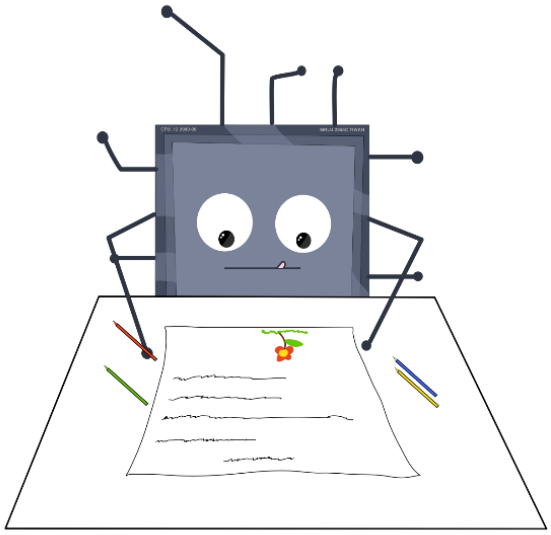
Department of Computer Engineering

Microprocessors and Assembly Language, Fall 2022, Dr. Farbeh

Homework 2 – Solutions

Lec 13-25



سوال ۱:

فرض کنید مقداری که r3 به آن اشاره دارد به این صورت است:

0x45
0xA5
0xC6
0x71

ثبات r3

و این ثبات در خانه 0x2000 0300 تا 0x2000 0303 حافظه قرار دارد.
ثبات r1 و r2 به ترتیب دارای شکل مشخص شده زیر هستند . همچنین ثبات r4 و r5 و r6 به ترتیب آدرس r1 و r2 و r3 را در خود دارند.

0xF5	0x34	0x21	0x56

ثبات r1

0x87	0xA4	0x5C	0x38

ثبات r2

به سوالات زیر پاسخ دهید (مقادیر اولیه ثبات ها و استک در هر بخش همان مقدار اولیه سوال است):
الف) مقدار r3 در پایان اجرای قطعه کد زیر چیست؟

```
strb r2 [r6];
ldrb r1 [r6];
ldrh r2 [r4];
ldr  r3 [r5];
strh r2 [r4];
ldrb r3 [r4];
```

ب) توضیح دهید تفاوت mov و ldr-str چیست و به نظر شما کدام از نظر کارایی سریع‌تر است؟ چرا؟

ج) فرض کنید یک دستور فرضی جدید به صورت ldrq داریم که ۳/۴ یک کلمه را می‌تواند جابه‌جا کند و کاملا در جای دیگر بریزد (به این معنی که ۳/۴ کلمه از خانه‌ای که برداشته می‌شود، خالی می‌گردد و در مکان جدید ریخته می‌شود.) چگونه می‌توان با حداقل دستورات ممکن و به کمک ldrq ، مقادیر r1 را با r2 جابه‌جا کرد؟ (استفاده از سایر دستورات مانند mov نیز جایز بوده به شرط آن‌که مسقیما برای خواسته اصلی سوال استفاده نشود.)

به طور مثال یک ثبات داریم با مقدار:
r=0x43F598C1
بعد از اجرای دستور ldrq به صورت ldrq r2 [address of r] مقدار جدید به این صورت می‌باشد:
r=0x43 و r2 = 0x87F598C1
و حال اگر برروی r، mov را اجرا کنیم مقدار 0x43 در خانه اول ثبات مقصد قرار میگیرد و سایر خانه‌ها دست نخورده باقی می‌ماند.

راهنمایی:از r3 برای جابه‌جایی کمک بگیرید.
توجه: در ثبات های r4 و r5 چیزی نباید بریزید چرا که ادرس ثبات های r2 و r1 در آن‌ها هستند.

د) با فرض وجود دستور ldrq با تعریف سوال قبل خروجی قطعه کد زیر را مشخص کنید:

```
str  r1 [r6];
ldrb r2 [r4];
strh r3 [r5];
ldrq r1 [r5];
strb r2 [r6];
ldrq r3 [r4];
```

پاسخ:

الف)

بعد از اجرا شدن خط اول:

R3: 0x45A5C638

بعد از اجرا شدن خط دوم :

R1: 0x38

R3: 0x45A5C638

بعد از اجرا شدن خط سوم:

R2: 0x38

R3: 0x45A5C638

بعد از اجرا شدن خط چهارم:

R3: 0x38

بعد از اجرا شدن خط پنجم:

R1: 0x38

R3: 0x38

بعد از اجرا شدن خط ششم:

R3: 0x38

ب)

- دستور MOV تنها می‌تواند مقادیر تا ۸ بیت(0x00 تا 0xff) یعنی همان ۰ تا ۲۵۵ را انتقال دهند؛ در حالی که دستورات LDR و STR تا ۳۲ بیت می‌توانند این عمل را انجام دهند.
- بنابراین دستور MOV سریع‌تر است.
- دستور LDR و STR می‌توانند با خانه‌های حافظه ارتباط داشته‌باشند و از آن‌ها برای انتقال داده استفاده کنند؛ در حالی که در دستور MOV انتقال داده بین دو رجیستر یا ریختن مقدار ۸ بیتی immediate در رجیستر امکان‌پذیر است.

ج)

MOV R3, R1 (R3= 0xF5342156)
LDRQ R1,[R5] (R1= 0xF5A45C38, R2=0x87)
MOV R1, R2 (R1= 0x87A45C38)
MOV R2, R3 (R2= 0xF5342156)

هر جواب صحیح مشابهی نیز پذیرفته خواهد شد.

د)

بعد از اجرای خط اول داریم:

R3: 0xF5342156

خط دوم:

R2: 0x56

خط سوم:

R2: 0x2156

خط چهارم:

R1: 0xF5002156

R2 = 0x0

خط پنجم:

R3: 0xF5342100

خط ششم:

R3: 0xF5002156

سوال ۲:

```
Area Code_Section, Readonly, Code
  LDR R3,  =Q_Data;
  MOV R4, #'2';
  SUB R4, R4, #0x29;
  ADD R3, R3, R4;
  LDRB R5, [R3];
HERE  B HERE

Area Data_Section, Data
Q_Data
  DCD 0x18
  DCB "HWDB"
  ALIGN 2
  DCB 0x36, 0x10
  END
```

با توجه به کد روبرو به سوالات زیر پاسخ دهید.

الف) پس از اجرای برنامه، محتوای نهایی رجیستر R5 را در حالتی که اعداد با متد Little Endian در رجیسترها ذخیره شوند به دست آورید.

ب) در صورتی که دستور ALIGN 2 را کامنت کنیم، نگاشت حافظه چه تغییری خواهد کرد؟ محتوای نهایی رجیستر R5 در این صورت چه خواهد بود؟

ج) آیا می‌توان شبه دستور LDR در خط دوم را با دستور دیگری جایگزین کرد؟

پاسخ:

الف)

مقدار موجود در رجیستر R3 در ابتدا آدرس اولین بایت داده Q_Data بوده که مقدار 0x18 در آن قرار گرفته است. در ادامه محتوای این رجیستر تغییر یافته و با محتوای رجیستر R4 یعنی 9 جمع خواهد شد. در نهایت آدرسی که رجیستر R3 به آن اشاره دارد با توجه به دستور Align یک بایت بعد از بایت 0x36 و یک بایت قبل از 0x10 می‌باشد که مقدار صفر دارد. پس محتوای نهایی رجیستر R5 صفر خواهد بود.

نگاشت حافظه با صرف نظر از حافظه اختصاص داده شده به دستورات به صورت زیر خواهد بود (بایت مشخص شده با رنگ یشمی(آدرس 0x00000019) بایتی است که در نهایت محتوای آن در رجیستر R5 نوشته می‌شود. هم‌چنین فرض کرده‌ایم آدرس اولین بایت داده Q_Data یا همان R3 برابر 0x00000010 است):

0x00000010	0x18	0	0	0
0x00000014	'H'	'W'	'D'	'B'
0x00000018	0x36	0	0x10	0

ب)

در این حالت داده‌های 0x36 و 0x10 در دو بایت پشت سرهم ذخیره شده و نگاشت حافظه به صورت زیر خواهد بود:

0x00000010	0x18	0	0	0
0x00000014	'H'	'W'	'D'	'B'
0x00000018	0x36	0x10	0	0

در این حالت مقدار 0x10 در رجیستر R5 ذخیره شده و محتوای این رجیستر به صورت زیر خواهد بود:

0x10	0	0	0

ج)

بله، می‌توانیم از شبه دستور ADR استفاده کنیم که دستور معادل به صورت زیر خواهد بود:

```
ADR R3, Q_Data
```

نکته در صورت استفاده از شبه دستور ADR نیازی به استفاده از = نخواهد بود.

سوال ۳:

فرض کنید مقدار اولیه رجیستر R10 معادل باینری شماره دانشجویی شما، R0 و R1 دو عدد دلخواه، R4 آدرس حافظه اولین گره در یک لیست پیوندی از اعداد صحیح و R5 آدرس اول لیست پیوندی می‌باشد.

برنامه‌ای بنویسید که مشخص کند چه تعداد الگوی "۱۰۱" در رجیستر R10 وجود دارد. (مثال: باینری ۹۸۳۱۰۹۰ مقدار ۱۰۰۱۰۱۱۰۰۰۰۰۰۱۰۱۰۱۱۰۰۱۰ است که الگوی ۱۰۱ در آن ۳ بار تکرار شده است.)

پاسخ:

```
1      AREA myData, DATA
2  PATTERN EQU     0x5 ; 101 in binary is equal to 5 in hex
3  _111    EQU     0x7 ;111 in hex
4  ITR     EQU     22 ; number of iterations needed
5  NUM     EQU     2_00000000000000010111100101
6
7      EXPORT __main
8      AREA myCode, CODE, READONLY
9      ENTRY
10
11 __main
12         MOV     R0, #0       ; number of patterns
13         MOV     R1, #ITR     ; number of iterations remained
14         MOV     R10, #NUM
15 FIND
16         AND     R2, R10, #_111  ; to get three most right bits
17         CMP     R2, #PATTERN    ; compare with 00...0101
18         ADDEQ   R0, R0, #1      ; count ++ if equals
19         MOV     R10, R10, ROR #1; rotate R10
20         SUBS    R1, R1, #1      ; ITR--
21         BNE     FIND
22 STOP    B       STOP
23         END
```

سوال ۴:

برنامه‌ای بنویسید که حاصل جمع دو آرایه از integer ها را با استفاده از subroutine پیدا کند. توجه داشته باشید که آرایه‌های ورودی signed و ۱۶ بیتی هستند. جواب آرایه اول را در POUT1&POUT2 و جواب آرایه دوم را در POUT3&POUT4 نشان دهید.

پاسخ:

```
; inputs:
; r0 = arr1[5]
; r1 = arr2[5]
; outputs:
; r0 = sum of arr1
; r1 = sum of arr2
sum_arrays:

    mov r4, #0          ; r4 = loop index (i)
    mov r5, #0          ; r5 = sum1 = 0
    mov r6, #0          ; r6 = sum2 = 0

    loop:
                        ; must use ldrsh because we're loading a signed half word
    ldrsh r2, [r0]      ; r2 = arr1[i]
    ldrsh r3, [r1]      ; r3 = arr2[i]

    add r5, r5, r2      ; sum1 += arr1[i]
    add r6, r6, r3      ; sum2 += arr2[i]

    add r4, r4, #1      ; i++
    add r0, r0, #2      ; arr1 += 2 (go 16 bits forward in array)
    add r1, r1, #2      ; arr2 += 2 (go 16 bits forward in array)

    cmp r4, #5          ; compare i with 5
    bne loop            ; repeat loop if i not equal to 5

    mov r0, r5          ; r0 = r5 = sum1
    mov r1, r6          ; r1 = r6 = sum2

    bx lr               ; return from subroutine

```

سوال ۵:

۴ مورد از قوانین استاندارد AAPCS را برای پیاده‌سازی توابع شرح دهید.

پاسخ:

- آرگومان‌های تابع باید از طریق رجیسترهای R0 تا R3 فرستاده شوند.
- مقدار بازگشتی باید درR0 (و R1 اگر مقدار ۶۴ بیتی است) قرار گیرد.
- توابع می‌توانند از رجیسترهای R4 تا R8 و R10 و R11 برای ذخیره اطلاعات موقت استفاده کنند. البته مقادیر این رجیسترها هنگام ورود باید ذخیره شود و قبل از بازگشت بازگردانی شوند.
- استک باید به صورت full ascending باشد.