



دانشکده مهندسی
کامپیوتر و فناوری اطلاعات

5/21/2021



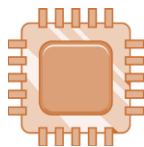
Homework 5

Lec 13-18



MICROPROCESSOR
AND
ASSEMBLY LANGUAGE

Spring 2021



الف) توضیح دهید آیا برای اجرای دستور branch از کل 4 گیگابایت فضا می‌توانیم استفاده کنیم؟

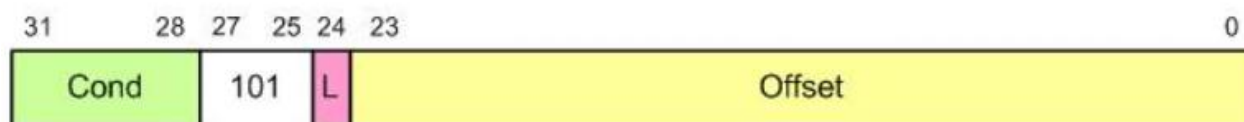
ب) مقدار نهایی رجیسترها در قطعه کد زیر چیست؟ (مرحله‌ها را توضیح دهید).

```
AREA myData, Data
L1  MOV R0, #0
    ADD R1,R0,#2
    B    L2
    ADD R2,R0,R1
L3  RSB R1,R2,#5
    B    L4
    ADD R1,R1,R0
L2  MOV R0,R1
    SUB R2,R0,#3
    B    L3
L4  B    L4
```

جواب قسمت الف)

خیر نمیتوانیم- طبق توضیحات استاد در صفحه 5 لکچر 18 ، ما مثلا از branch در HERE B HERE استفاده میکردیم. به کمک اون می‌گفتیم بپر به اون آدرسی که لیبل HERE می‌گوید.

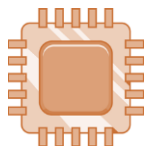
می‌گفتیم آدرس‌ها 32 بیتی اند. اون آدرسی که در دستور branch قرار می‌گیرد طبق این شکل یک آدرس 24 بیتی است نه یک آدرس 32 بیتی. این معنیش اینه که اگرچه آدرس یک 32 بیتی باید باشد ولی وقتی از branch استفاده می‌کنیم ، نمیتوانیم به هر جایی دلمون می‌خواهد branch کنیم و فقط به یک فضایی که با 24 بیت قابلیت آدرس دهی دارد، میتوانیم branch کنیم. بنابراین با branch نمیتوان از کل فضای 4 گیگ حافظه استفاده کرد.



جواب قسمت ب)

در ابتدای کار یک خط نوشته شده که جزء خطوط دستور نمیباشد. درواقع به کمک AREA اومدیم سکشن بندی کردیم. بعد گفتیم اسم این فضا myData میباشد و از جنس Data میباشد.

سپس به سراغ خط اول کد می‌رویم که به کمک دستور MOV مقدار constant صفر را در رجیستر R0 میریزد.



سپس به کمک دستور ADD مقدار ثابت 2 را با مقدار موجود در رجیستر R0 که برابر 0 بود جمع میکند و حاصل را در رجیستر R1 میریزد. بنابراین در R1 مقدار 2 داریم.

سپس branch میکند (بدون شرط) به L2.

در آنجا به کمک دستور MOV مقدار موجود در رجیستر R1 یعنی 2 را در رجیستر R0 میریزد. پس مقدار R0 اکنون 2 میباشد.

به کمک دستور SUB ، مقدار موجود در R0 را منهای 3 میکند و حاصل را در R2 میریزد. در اینجا مقدار 1- را در R2 میریزد. (طبق توضیحات استاد مقدار موجود در رجیسترها به صورت علامت دار و درواقع مکمل 2 میباشد).

سپس branch میکند (بدون شرط) به L3.

به کمک RSB که Reverse SUB میباشد مقدار ثابت 5 را منهای مقدار موجود در رجیستر R2 یعنی 1- میکند و حاصل که 6 هست را در رجیستر R1 میریزد.

سپس branch میکند (بدون شرط) به L4.

در آنجا هم دوباره به L4 برنچ میکند و درواقع در یک loop قرار میگیرد. و دیگر برنامه پیش نمیروود و fetch کردن دستورات تمام میشود.

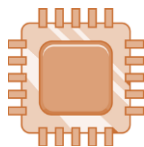
پس مقادیر نهایی رجیسترها :

R0 میشود 2 (به صورت هگز میشود: 0x00000002)

R1 میشود 6 (به صورت هگز میشود: 0x00000006)

R2 میشود 1- (به صورت هگز میشود: 0xFFFFFFFF)

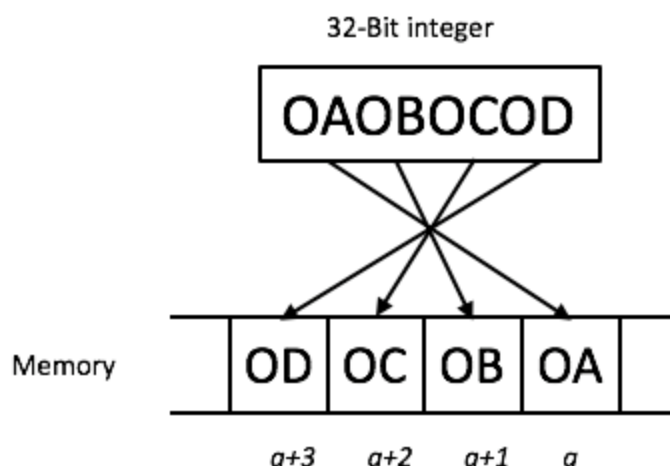
(همچنین لازم به ذکر است که مقادیر رجیسترها به صورت علامت دار two's complement میباشد)



(2)

الف) با ذکر دلیل بیان کنید تصویر زیر کدام یک از روش‌های ذخیره‌سازی در حافظه را نشان می‌دهد؟

ب) عدد 1025 را به دو روش little endian و big endian نمایش دهید.



جواب قسمت الف)

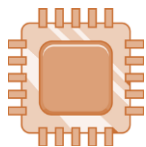
روش Big endian را نشان می‌دهد. زیرا همانطور که از تصویر پیداست مقادیر پرارزش‌تر داده (MSB) را در آدرس با شماره‌ی کمتر و مقادیر کم ارزش‌تر داده (LSB) را در آدرس با شماره‌ی بیشتر ذخیره می‌کند. مثلاً OA که MSB است در آدرس پایین‌تری قرار گرفته و OD که LSB است در آدرس بالاتری قرار گرفته است.

- **Big endian**

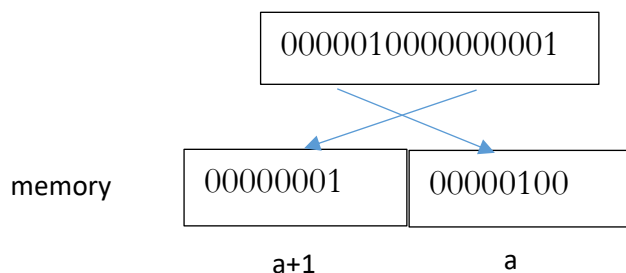
- Low byte goes to the high memory location
- High byte goes to the low memory address

جواب قسمت ب)

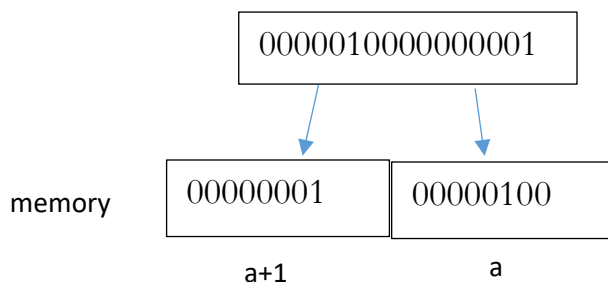
ابتدا باید عدد 1025 را به باینری تبدیل کنیم که مقدارش 10000000001 میشود. اگر به 16 بیت تبدیل کنیم چون در خانه‌های حافظه با بایت‌ها کار می‌کنیم میشود: 0000010000000001



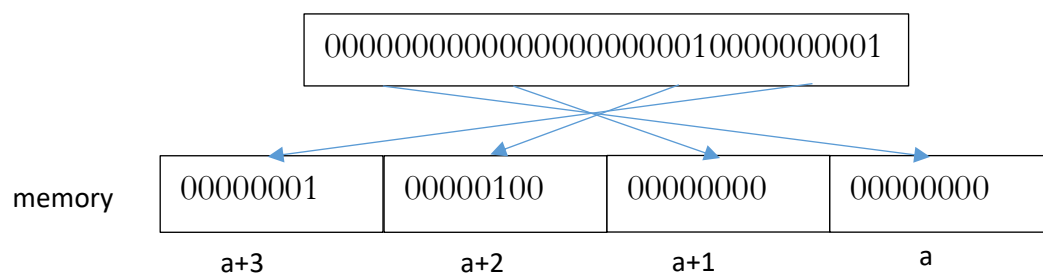
حال میدانیم که در نمایش big endian مقادیر و بایت پرارزش تر داده (MSB) را در آدرس با شمارهی کمتر و مقادیر و بایت کم ارزش تر داده (LSB) را در آدرس با شمارهی بیشتر ذخیره میکند. بنابراین عدد به صورت زیر ذخیره میشود. (اینجا فرض شده یک دو بایتی از حافظه را خواستیم اشغال کنیم)



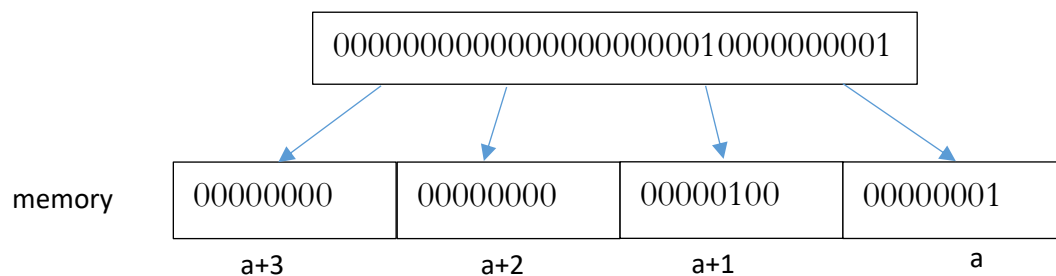
در نمایش little endian مقادیر و بایت پرارزش تر داده (MSB) را در آدرس با شمارهی بیشتر و مقادیر و بایت کم ارزش تر داده (LSB) را در آدرس با شمارهی کمتر ذخیره میکند.

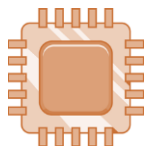


البته اگر میخواستیم فرض کنیم مثل قسمت الف عدد ما یک 32 بیتی است و درواقع قرار است یک 32 بیت از حافظه را به آن اختصاص دهیم، آنگاه نمایش big endian آن به صورت زیر میشود:



نمایش little endian آن نیز به این صورت خواهد بود:



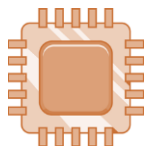


(3) اگر سطح دسترسی ما کاربر باشد اما دستوری را به پردازنده بدهیم که نیاز به سطح دسترسی `privilege` داشته باشد، چه اتفاقی رخ می‌دهد؟

اگر سطح دسترسی ما کاربر باشد نمیتواند دستورات `privileged` را در آن سطح اجرا کند و `exception` میدهد. در واقع همچنین دستوری در سطح کاربر غیرمجاز میباشد و اجرا نخواهد شد. برای انجام این دستورات نیاز است که در `Handler` یا همان `Privileged` باشیم.

اگر واقع نیاز داریم که این کد نوشته شده را که به سطح `privileged` نیاز دارد را اجرا کنیم؛ باید این دستور به عنوان `ISR` نوشته شود و وقتی در `Handler` هستیم نمیتوان به راحتی `Privileged` را به حالت `Privileged` تغییر داد و دستور مورد نظر را انجام داد. در ابتدا بفرض ما در `Handler` هستیم و برنامه ی عادی ما در همان `user` اجرا میشود تا اینکه یک اینترپت یا `exception` بیاید و سپس `state` پراسسور به صورت اتوماتیک در `stack` ذخیره میشود و به `Handler` برویم. سپس اگر دستور را به عنوان `ISR` آن اینترپت نوشته باشیم و همچنین اولویت اینترپت نسبت به کار قبلی بیشتر باشد و `Mask` هم نشود، آنگاه آن دستور ما که نیاز به این سطح از دسترسی داشت اجرا میشود. سپس پس از انجام آن به `user` برمیگردیم و `state` ای که ذخیره کرده بودیم بازیابی میشود.

به طور خلاصه چون این دستور غیرمجاز است وقفه نرم افزاری رخ میدهد.



الف) برنامه‌ای بنویسید که مقدار R0 فاکتوریل را در R1 قرار دهد.

ب) به چند بایت از فضای پشته برای این کار نیاز است؟

جواب قسمت الف)

این سوال در فیلمی که در کانال برای آموزش keil قرار داده شده بود حل شده است. با توجه به آن فیلم داریم:

```

1  AREA myData, DATA
2  VAL EQU 5
3
4  AREA myCode, CODE, READONLY
5  ENTRY
6  EXPORT __main
7
8  __main
9      LDR R0, =VAL ;initial R0 with value that we want to calculate its factorial
10     LDR R1, =1 ;initial R1 with 1 and also we holds the answer inside it
11     LDR R2, =1 ;initial R2 with 1 and also it is my counter
12
13 myloop
14     MUL R1, R2, R1 ; R1=R2 * R1
15     ADD R2, R2, #1 ; R2=R2 + 1
16     SUBS R4, R0, R2 ; R4=R0 - R2
17     BNE myloop ;if R2 < VAL it returns to myloop
18     MUL R1, R2, R1 ;after last loop we should calculate R1=R2 * R1 again because now the value of R1 is equal to VAL
19
20 here B here
21     END
    
```

همچنین اگر شبیه ساز را اجرا کنیم مقداری که در رجیستر R1 داریم برابر حاصل مقدار R0 فاکتوریل میباشد. (ما مقدار اولیه R0 را برابر با 5 دادیم و در نهایت 5! که برابر با 120 در دسیمال و 78 در هگز میباشد را در R1 قرار دادیم)

Registers

Register	Value
R0	0x00000005
R1	0x00000078
R2	0x00000005
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000400
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x000001E8
xPSR	0x61000000

Disassembly

```

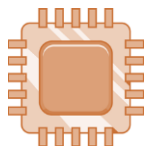
0x000001E8 E7FE B 0x000001E8
0x000001EA 0000 MOVN R0,R0
0x000001EC 0000 MOVN R0,R0
0x000001EE 0000 MOVN R0,R0
    
```

q4.s

```

1  AREA myData, DATA
2  VAL EQU 5
3
4  AREA myCode, CODE, READONLY
5  ENTRY
6  EXPORT __main
7
8  __main
9      LDR R0, =VAL ;initial R0 wi
10     LDR R1, =1 ;initial R1 wi
11     LDR R2, =1 ;initial R2 wi
12
13 myloop
14     MUL R1, R2, R1 ; R1=R2 * R1
15     ADD R2, R2, #1 ; R2=R2 + 1
16     SUBS R4, R0, R2 ; R4=R0 - R2
17     BNE myloop ;if R2 < VAL
18     MUL R1, R2, R1 ;after last :
19
20 here B here
21     END
    
```

جواب قسمت ب) طبق اطلاعاتی که در کانال برای این سوال قرار داده شده بود، پاسخ 31 بایت است.



MICROPROCESSOR AND ASSEMBLY LANGUAGE

Dr. Farbeh

Homework 5



دانشکده مهندسی
کامپیوتر و فناوری اطلاعات

5) عددی را در خانه 0x05000000 ثبت کنید. برنامه‌ای بنویسید که آن را تقسیم بر توان‌های 2، از یک تا ده بکند و در ده رجیستر اول قرار دهد.

دو راه برای این کار به نظرم رسید. راه اول:

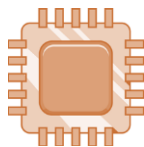
```

Project: q5
Target 1
Source Group 1
Device
startup_SAM3X
system_SAM3X

1 AREA myData, DATA
2 myAddress EQU 0x05000000
3 myValue EQU 1024
4
5
6 AREA myCode, CODE, READONLY
7 ENTRY
8 EXPORT __main
9
10 __main
11 LDR R10, =myAddress ;initial R10 with myAddress
12 LDR R11, =myValue ;initial R11 with myValue
13 STR R11, [R10] ;store myValue in the memory cell that myAddress point that
14
15 MOV R12, #2 ;initial R12 with divisor value=2
16 UDIV R0, R11, R12 ;R0 = R11/R12
17 MOV R12, #4 ;initial R12 with divisor value=4
18 UDIV R1, R11, R12 ;R0 = R11/R12
19 MOV R12, #8 ;initial R12 with divisor value=8
20 UDIV R2, R11, R12 ;R0 = R11/R12
21 MOV R12, #16 ;initial R12 with divisor value=16
22 UDIV R3, R11, R12 ;R0 = R11/R12
23 MOV R12, #32 ;initial R12 with divisor value=32
24 UDIV R4, R11, R12 ;R0 = R11/R12
25 MOV R12, #64 ;initial R12 with divisor value=64
26 UDIV R5, R11, R12 ;R0 = R11/R12
27 MOV R12, #128 ;initial R12 with divisor value=128
28 UDIV R6, R11, R12 ;R0 = R11/R12
29 MOV R12, #256 ;initial R12 with divisor value=256
30 UDIV R7, R11, R12 ;R0 = R11/R12
31 MOV R12, #512 ;initial R12 with divisor value=512
32 UDIV R8, R11, R12 ;R0 = R11/R12
33 MOV R12, #1024 ;initial R12 with divisor value=1024
34 UDIV R9, R11, R12 ;R0 = R11/R12
35
36 here B here
37 END

```

خروجی و درواقع مقدار ذخیره شده در رجیسترها (منظور سوال از R0 تا R9 میباشد) به صورت زیر میباشد:



MICROPROCESSOR AND ASSEMBLY LANGUAGE

Dr. Farbeh

Homework 5



دانشکده مهندسی
کامپیوتر و فناوری اطلاعات

فقط نکته ای که هست اینه که چون آدرس 0x05000000 از حجم حافظه میکرو در سیستم بنده بیشتر بود به ارور برمیخوردم و مجبور شدم در حین شبیه سازی از آدرس 0x000801F8 استفاده کنم.

راه دومی که به نظرم رسید استفاد از LSR بود که با شیفت به راست دادن میتوان عدد موردنظر را تقسیم بر توان های 2 کرد:

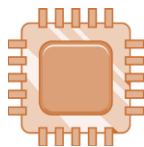
```
1 AREA myData, DATA
2 myAddress EQU 0x05000000
3 myValue EQU 1024
4
5
6 AREA myCode, CODE, READONLY
7 ENTRY
8 EXPORT __main
9
10 __main
11 LDR R10, =myAddress ;initial R10 with myAddress
12 LDR R11, =myValue ;initial R11 with myValue
13 STR R11, [R10] ;store myValue in the memory cell that myAddress point that
14
15 LSR R0, R11, #1 ;1 bit shifted to right value save in R0 destination Register
16 LSR R1, R11, #2 ;2 bit shifted to right value save in R1 destination Register
17 LSR R2, R11, #3 ;3 bit shifted to right value save in R2 destination Register
18 LSR R3, R11, #4 ;4 bit shifted to right value save in R3 destination Register
19 LSR R4, R11, #5 ;5 bit shifted to right value save in R4 destination Register
20 LSR R5, R11, #6 ;6 bit shifted to right value save in R5 destination Register
21 LSR R6, R11, #7 ;7 bit shifted to right value save in R6 destination Register
22 LSR R7, R11, #8 ;8 bit shifted to right value save in R7 destination Register
23 LSR R8, R11, #9 ;9 bit shifted to right value save in R8 destination Register
24 LSR R9, R11, #10 ;10 bit shifted to right value save in R9 destination Register
25
26 here B here
27 END
```

در این حالت نیز نتیجه مانند قبل است:

Register	Value
R0	0x00000200
R1	0x00000100
R2	0x00000080
R3	0x00000040
R4	0x00000020
R5	0x00000010
R6	0x00000008
R7	0x00000004
R8	0x00000002
R9	0x00000001
R10	0x000801F8
R11	0x00000400
R12	0x00000000
R13 (SP)	0x20000400
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x00000200
xPSR	0x01000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	Z0
Sec	0.00000167

Address	Instruction	Comment
0x000801E8	EA4F145B LSR r4,r11,#5	
0x000801EC	EA4F145B LSR r5,r11,#6	
0x000801F0	EA4F145B LSR r6,r11,#7	
0x000801F4	EA4F271B LSR r7,r11,#8	
0x000801F8	EA4F285B LSR r8,r11,#9	
0x000801FC	EA4F299B LSR r9,r11,#10	


```
8 EXPORT __main
9
10 __main
11 LDR R10, =myAddress ;initial R10 with myAddress
12 LDR R11, =myValue ;initial R11 with myValue
13 STR R11, [R10] ;store myValue in the memory cell that myAddress point that
14
15 LSR R0, R11, #1 ;1 bit shifted to right value save in R0 destination Register
16 LSR R1, R11, #2 ;2 bit shifted to right value save in R1 destination Register
17 LSR R2, R11, #3 ;3 bit shifted to right value save in R2 destination Register
18 LSR R3, R11, #4 ;4 bit shifted to right value save in R3 destination Register
19 LSR R4, R11, #5 ;5 bit shifted to right value save in R4 destination Register
20 LSR R5, R11, #6 ;6 bit shifted to right value save in R5 destination Register
21 LSR R6, R11, #7 ;7 bit shifted to right value save in R6 destination Register
22 LSR R7, R11, #8 ;8 bit shifted to right value save in R7 destination Register
23 LSR R8, R11, #9 ;9 bit shifted to right value save in R8 destination Register
24 LSR R9, R11, #10 ;10 bit shifted to right value save in R9 destination Register
25
26 here B here
27 END
```



6) چند کاربرد EQU Directive را نام برده و آن‌ها را شرح دهید.

در کل میتوان از directive ها استفاده کرد تا برنامه را منسجم تر و خوانا تر نوشت.

از آنجایی که ما در اسمبلی مثل زبان های سطح بالا نمیتوانیم متغیر تعریف کنیم، در اینجا به کمک EQU میتوانیم برای دیتاهایی که داریم سمبل تعریف کنیم. طبق صفحه 4 اسلاید 17 نیز: از EQU برای تعریف یک نام برای یک مقدار ثابت یا یک آدرس ثابت استفاده میشود. باعث میشود که برنامه آسانتر قابل خواندن باشد و باعث جلوگیری از سرچ کردن داخل کل یک برنامه برای یافتن تغییرات و اصلاحات آن ها میشود.

کاربردها:

Using EQU for fixed data assignment

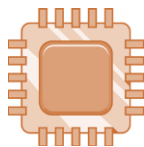
برای مثال به کمک EQU میتوان یک مقدار ثابت را بهش سمبل و اسم خاصی بدهیم و از این به بعد هرجایی در خط های مختلف برنامه با این مقدار ثابت خواستیم کاری انجام دهیم از سمبل آن استفاده میکنیم. به این صورت اگر یک بخواهیم این مقدار ثابت را تغییر دهیم و آپدیت کنیم، میرویم مقداری را که به اون label دادیم را فقط تغییر میدهیم. در صورتی که اگر از EQU استفاده میکردیم باید برای مثال در کل خطوط دنبال این مقدار ثابت میگشتیم و دستی و دونه دونه مقدار آن را در صورت نیاز عوض میکردیم. این مقدار ثابت میتواند یک عدد hex یا binary یا decimal و یا ASCII characters باشد:

DATA1 EQU 0x39 ; the way to define hex value

DATA2 EQU 2_00110101 ; the way to define binary value (35 in hex)

DATA3 EQU 39 ; decimal numbers (27 in hex)

DATA4 EQU '2' ; ASCII characters



Using EQU for special register address assignment

فرض کنید یک رجیستر که باهش زیاد کار دارید، مثلاً رجیستر یکی از پورت‌های IO که میدانیم این رجیسترها جدا از 16 تا رجیستر داخلی پردازنده میباشد و هر کدام یک آدرس 32 بیتی برای خودشون دارند. برای اینکه مجبور نباشیم این آدرس‌ها را حفظ کنیم، از یک label استفاده میکنیم و یک بار آن آدرس را به کمک EQU به آن label، assign میکنیم و در طول برنامه دیگر با آن label به جای آدرس استفاده میکنیم. برای مثال در کد زیر به کمک EQU برای آدرس 0x3FFFC058 یک label، FIO2SET0، قرار دادیم و جلوتر این آدرس را به کمک شبه دستور LDR داخل رجیستر R2 قرار دادیم.

FIO2SET0 EQU 0x3FFFC058 ; PORT2 output set register 0 address

MOV R6, #0x01 ; R6 = 0x01

LDR R2, =FIO2SET0 ; R2 = 0x3FFFC058

STRB R6, [R2] ; Write 0x01 to FIO2SET0

Using EQU for RAM address assignment

میتوان از EQU برای اینکه یک خانه‌ی حافظه را هم (یک قسمتی از SRAM که یک دیتای خاصی دارد) را هم آدرس دهی کنید و درواقع label گذاری کنید. و مثل قبل هر جا کاری با آن خانه از حافظه داشتیم مثل انتقال دیتا به آن خانه یا خواندن از آن، میتوان به جای آدرس از این label استفاده کرد.

مثلاً در این قطعه کد سمبل SUM اشاره میکند به خانه حافظه‌ی 0x40000120 که یک خانه از فضای SRAM میباشد.

SUM EQU 0x40000120 ; assign RAM location to SUM

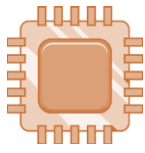
MOV R2, #5 ; load R2 with 5

MOV R1, #2 ; load R1 with 2

ADD R2, R2, R1 ; R2 = R2 + R1

LDR R3, =SUM ; load R3 with 0x40000120

STRB R2, [R3] ; store the result SUM



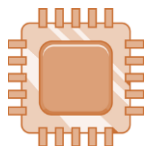
لازم به ذکر است که بگوییم که طبق مطالب ذکر شده در صفحه 6 اسلاید 17 ، یک ورژن دیگری نیز برای EQU وجود دارد که RN directive میباشد. از آن برای label زدن به رجیسترهای 16 گانه پردازنده و نیاز نیست که برای آن ها از EQU استفاده کنید. پس به نحوی RN همان EQU میباشد برای label زدن به 16 تا رجیستر داخل پردازنده. مثال آن در کد زیر آمده:

- **To give a CPU register a name**

VAL1 RN R1 ; define VAL1 as a name for R1

VAL2 RN R2 ; define VAL2 as a name for R2

SUM RN R3 ; define SUM as a name for R3



(7) برنامه‌ای بنویسید که شماره دانشجویی شما را در R0 قرار دهد و تعداد دفعاتی که الگوی 111 در آن تکرار شده را در R1 بنویسد.

برداشتی که بنده از سوال کردم این بوده که ما به صورت دسیمال شماره دانشجوییمان را در R0 میریزیم و در رجیسترها که مقدار باینری آن را داریم باید تعداد 111 ها را بیابیم.

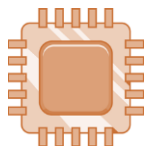
شماره دانشجویی من 9731701 است که به باینری برابر با 100101000111111001110101 می باشد که 24 بیت است. تعداد الگوهای 111 در آن برابر با 5 می باشد.

توضیحات: در ابتدا شماره دانشجویی را در R0 ریختم. رجیسترهای دیگر نیز مقدار دهی اولیه شده اند و در واقع در R1 قرار است تعداد الگوهای 111 را بشماریم پس مقدار اولیه 0 می خواهد. در R2 نیز تعداد ارقام شماره دانشجویی (به باینری) را نگه میداریم که 24 می باشد و حلقه ی ما باید به تعداد آن تکرار شود.

کد به صورت زیر می باشد:

```
Project: q7
Target 1
Source Group 1
CMSIS
Device

q7.s
1 AREA myData, DATA
2 myID EQU 9731701
3
4 AREA myCode, CODE, READONLY
5 ENTRY
6 EXPORT __main
7
8 __main
9 LDR R0, =myID ;initial R0 with myID
10 MOV R1, #0 ;initial R1 with 0 and this is the counter of 111
11 MOV R2, #24 ;initial R2 with 24 and this is the counter of loop
12 MOV R3, #0 ;initial R3 with 0 and this for holding three LSB bits of R0
13
14 loop
15 SUBS R2, R2, #1 ;R2 = R2 - 1 and set the flags
16 BMI here2 ;if R2 < 0 then branch to here2 and finish
17 AND R3, R0, #2_111 ;we want to save three LSB bits of R0 in R3
18 CMP R3, #2_111 ;compare the three bits that we save recently in R3 with 111 in binary
19 LSR R0, R0, #1 ;shift R0 one bit to right
20 BEQ here1 ;if the the value of R3 equals to 111 then branch to here1
21 B loop
22
23 here1
24 ADD R1, R1, #1 ;R1 = R1 + 1 and this is the counter of 111 patern
25 B loop
26
27 here2 B here2
28 END
```



MICROPROCESSOR AND ASSEMBLY LANGUAGE

Dr. Farbeh

Homework 5



دانشکده مهندسی
کامپیوتر و فناوری اطلاعات

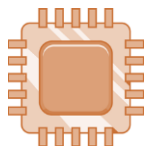
همچنین مقادیر رجیسترها به صورت زیر میباشد:

The screenshot shows a debugger interface with two main panels. The left panel, titled 'Registers', displays a list of registers (R0 to R15 and PSR) and their current values. The right panel, titled 'Disassembly', shows the assembly code being executed, including instructions like LDR, MOV, SUBS, BMI, AND, CMP, LSR, BEQ, and ADD, along with their addresses and comments.

Register	Value
R0	0x00000000
R1	0x00000005
R2	0xFFFFFFFF
R3	0x00000001
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000400
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x000801F2
PSR	0x81000000

```
7
8
9  _main
10  LDR R0, =myID           ;initial R0 %
11  MOV R1, #0              ;initial R1 %
12  MOV R2, #24             ;initial R2 %
13  MOV R3, #0              ;initial R3 %
14
15  loop
16  SUBS R2, R2, #1          ;R2 = R2 - 1
17  BMI here2               ;if R2 < 0 t
18  AND R3, R0, #2_111      ;we want to s
19  CMP R3, #2_111          ;compare the
20  LSR R0, R0, #1           ;shift R0 one
21  BEQ here1               ;if the the v
22
23  here1
24  ADD R1, R1, #1           ;R1 = R1 + 1
25  B loop
26
27  here2 B here2
28  END
```

همانطور که مشاهده میشود در R1 مقدار 5 که تعداد دفعات تکرار الگوی 111 در شماره دانشجویی بنده (به باینری) میباشد است.



- مهلت ارسال تمرین ساعت 23.55 روز سه شنبه 18 ام خرداد می باشد.
- سوالات خود را می توانید تنها از طریق ایمیل زیر بپرسید.
 - alirezasalehy@aut.ac.ir
- ارائه پاسخ تمرین بهتر است به دو روش زیر باشد:
 - (1) استفاده از فایل docx. تایپ پاسخها و ارائه فایل Pdf
 - (2) چاپ تمرین و پاسخ دهی به صورت دستنویس خوانا
- فایل پاسخ تمرین را تنها با قالب **HW5-9731***.pdf** در مدل بارگزاری کنید.
- نمونه: HW5 -9731063
- فایل زیپ ارسال نکنید.