دانشکده مهندسی
کامپیوتر و فناوری اطلاعات

5/21/2021

Homework 5

Lec 13-18

MICROPROCESSOR
AND
ASSEMBLY LANGUAGE

Spring 2021

(1

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

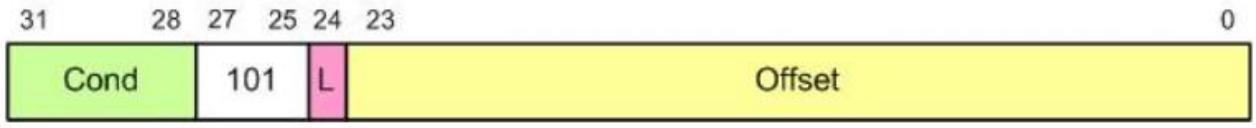
ب) مقدار نهایی رجیسترها در قطعه کد زیر چیست؟ (مرحله‌ها را توضیح دهید).

```
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 لکچر18 ، ما مثلا از branch در HERE B HERE استفاده میکردیم. به کمک اون میگفتیم بپر به اون آدرسی که لیبل HERE میگوید.

میگفتیم آدرس ها 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 32بیتی باید باشد ولی وقتی از branch استفاده میکنیم ، نمیتوانیم به هرجایی 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 مریزد.

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 نمیرود 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)

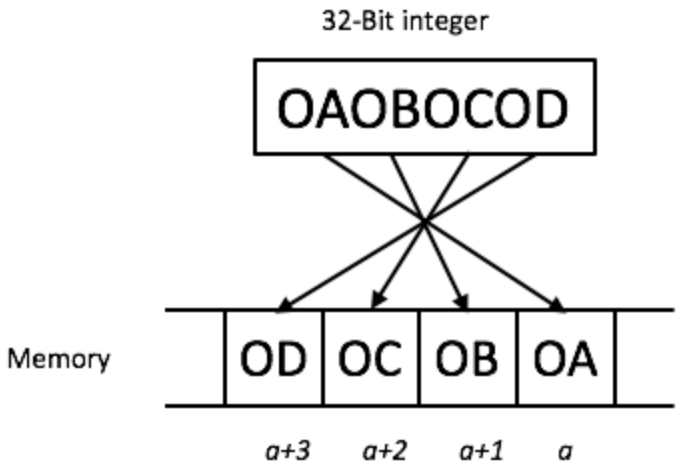
R2 میشود 1-(به صورت هگز میشود: 0xFFFFFFFF)

(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 big endian و little 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 و مقادیر کم ارزش تر داده(LSB) را در آدرس با شماره‌ی بیشتر ذخیره میکند.

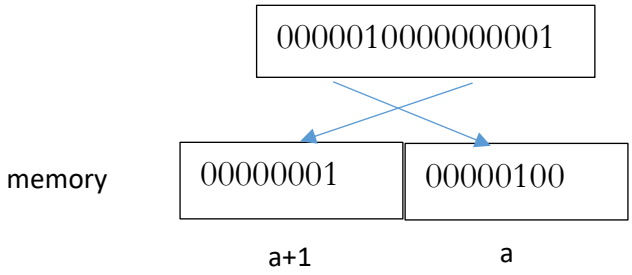
مثلا OA که MSB است در آدرس پایین تری قرار گرفته و OD که LSB است در آدرس بالاتری قرار گرفته است.

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

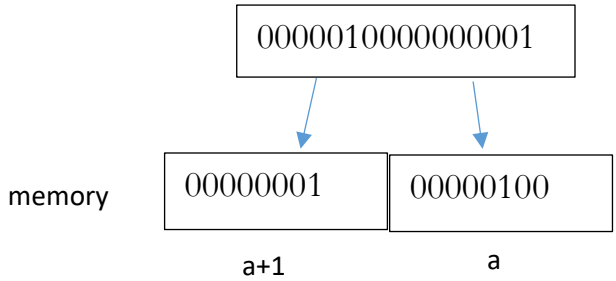
جواب قسمت ب)

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

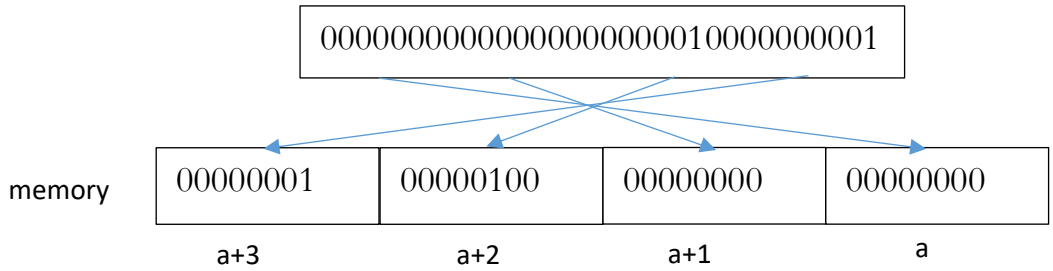
حال میدانیم که در نمایش big endian مقادیر و بایت پرارزش‌تر داده(MSB) را در آدرس با شماره‌ی کمتر و مقادیر و بایت کم ارزش تر داده(LSB) را در آدرس با شماره‌ی بیشتر ذخیره میکند. بنابراین عدد به صورت زیر إخیره میشود.(اینجا فرض شده یک دو بایتی از حافظه را خواستیم اشغال کنیم)



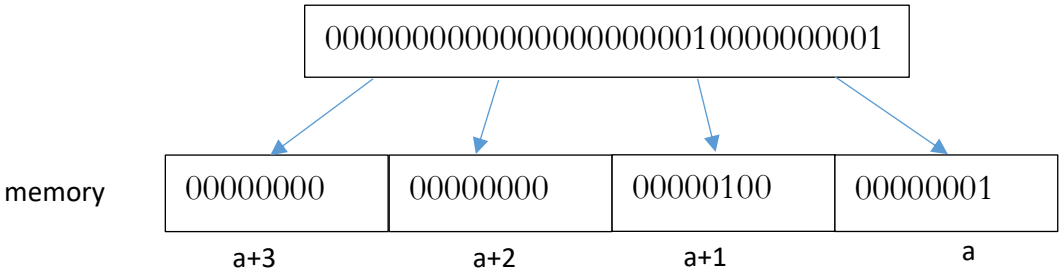
در نمایش little endian مقادیر و بایت پرارزش‌تر داده(MSB) را در آدرس با شماره‌ی بیشتر و مقادیر و بایت کم ارزش تر داده(LSB) را در آدرس با شماره‌ی کمتر ذخیره میکند.



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 همچین دستوری در سطح کاربر غیرمجاز میباشد و اجرا نخواهد شد. برای انجام این دستورات نیاز است که در مود Handler یا همان مود 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 مود کاربر هستیم نمیتوان به راحتی مود را به حالت Privileged تغییر داد و دستور مورد نظر را انجام داد. در ابتدا برفرض ما در مود کاربر هستیم و برنامه ی عادی ما در همان مود اجرا میشود تا اینکه یک اینتراپت یا exception بیاید و سپس state پراسسور به صورت اتوماتیک در stack ذخیره میشود و به مود Handler برویم. سپس اگر دستور را به عنوان ISR آن اینتراپت نوشته باشیم و همچنین اولویت اینتراپت 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 مود قبلی برمیگردیم و state ای که ذخیره کرده بودیم بازیابی میشود .

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

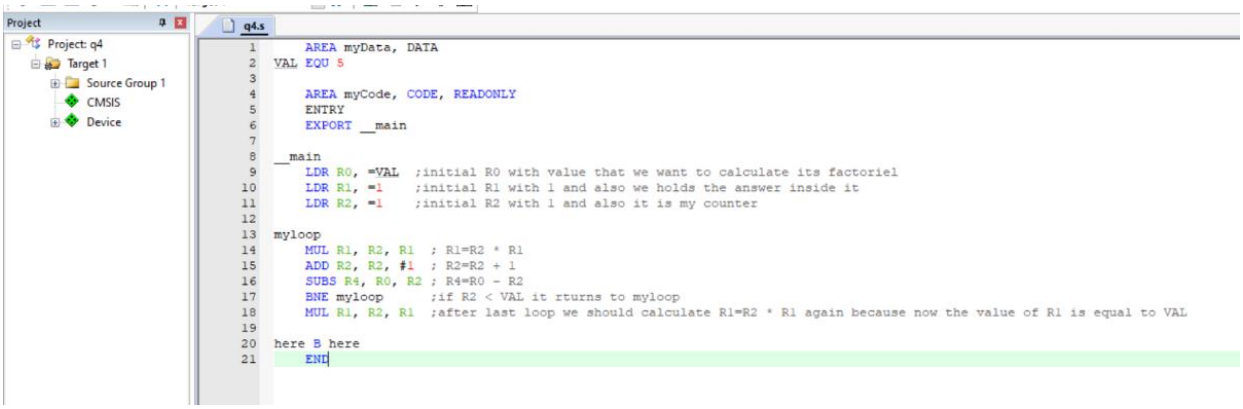
(4

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

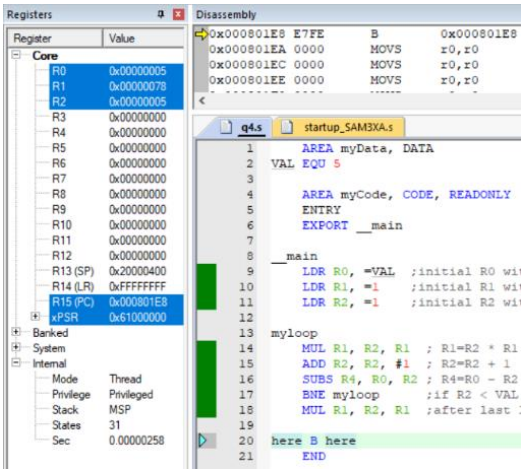
ب) به چند بایت از فضای پشته برای این کار نیاز است؟

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:



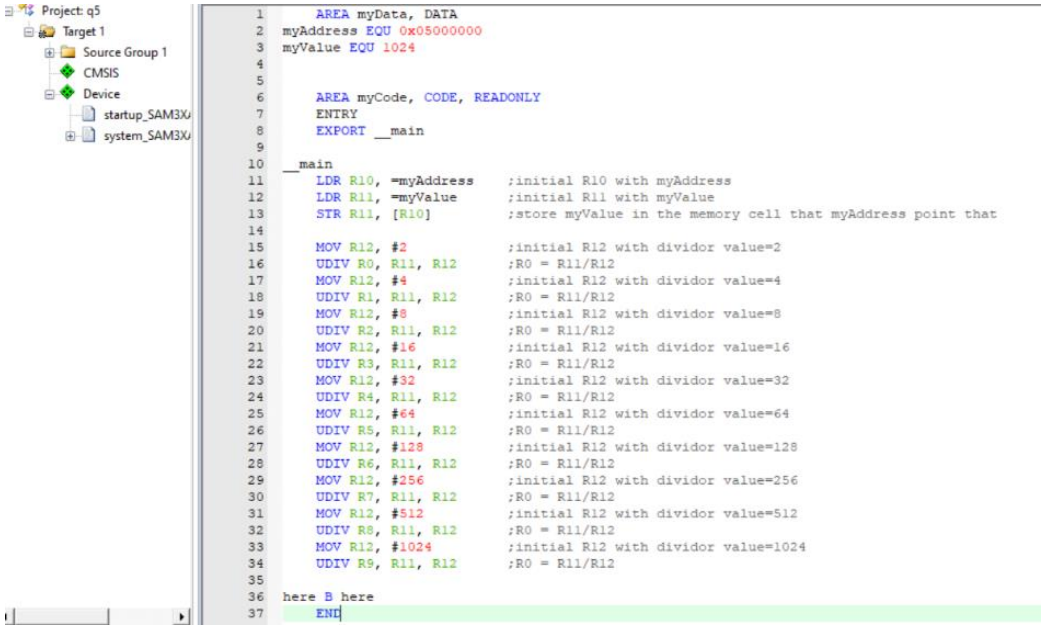
همچنین اگر شبیه ساز را اجرا کنیم مقداری که در رجیستر R1 داریم برابر حاصل مقدار R0 فاکتوریل میباشد.(ما مقدار اولیه R0 را برابر با 5 دادیم و درنهایت !5 که برابر با 120 در دسیمال و 78 در هگز میباشد را در R1 قرار دادیم)



جواب قسمت ب) طبق اطلاعیه کانال نیازی نیست حل شود.

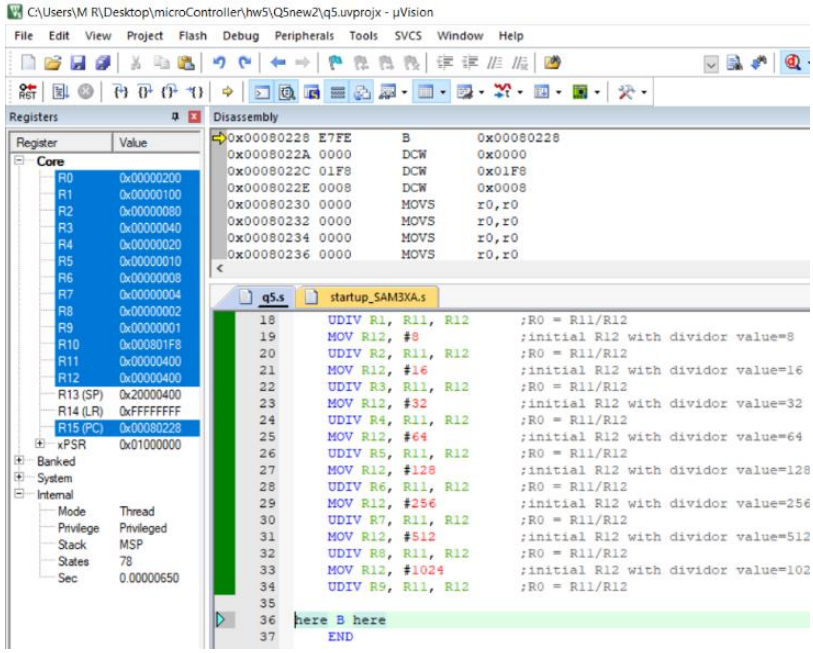
5) عددی را در خانه 0x05000000 ثبت کنید. برنامه‌ای بنویسید که آن را تقسیم بر توان‌های 2، از یک تا ده بکند و در ده رجیستر اول قرار دهد.

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




```
    AREA myData, DATA
myAddress EQU 0x05000000
myValue EQU 1024


    AREA myCode, CODE, READONLY
    ENTRY
    EXPORT __main

__main
    LDR R10, =myAddress     ;initial R10 with myAddress
    LDR R11, =myValue       ;initial R11 with myValue
    STR R11, [R10]          ;store myValue in the memory cell that myAddress point that

    MOV R12, #2             ;initial R12 with dividor value=2
    UDIV R0, R11, R12       ;R0 = R11/R12
    MOV R12, #4             ;initial R12 with dividor value=4
    UDIV R1, R11, R12       ;R0 = R11/R12
    MOV R12, #8             ;initial R12 with dividor value=8
    UDIV R2, R11, R12       ;R0 = R11/R12
    MOV R12, #16            ;initial R12 with dividor value=16
    UDIV R3, R11, R12       ;R0 = R11/R12
    MOV R12, #32            ;initial R12 with dividor value=32
    UDIV R4, R11, R12       ;R0 = R11/R12
    MOV R12, #64            ;initial R12 with dividor value=64
    UDIV R5, R11, R12       ;R0 = R11/R12
    MOV R12, #128           ;initial R12 with dividor value=128
    UDIV R6, R11, R12       ;R0 = R11/R12
    MOV R12, #256           ;initial R12 with dividor value=256
    UDIV R7, R11, R12       ;R0 = R11/R12
    MOV R12, #512           ;initial R12 with dividor value=512
    UDIV R8, R11, R12       ;R0 = R11/R12
    MOV R12, #1024          ;initial R12 with dividor value=1024
    UDIV R9, R11, R12       ;R0 = R11/R12

here B here
    END
```

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

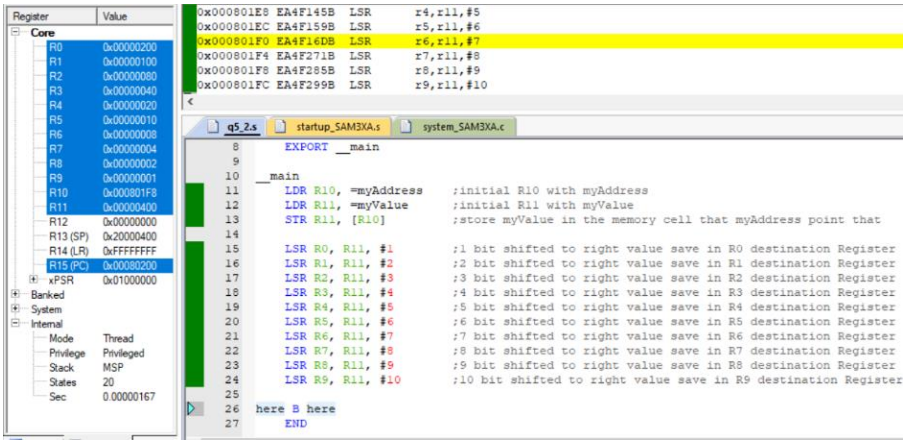


فقط نکته ای که هست اینه که چون آدرس 0x05000000 از حجم حافظه میکرو در سیستم بنده بیشتر بود به ارور برمیخوردم و مجبور شدم در حین شبیه سازی از آدرس 0x000801F8 استفاده کنم.

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

```
    AREA myData, DATA
myAddress EQU 0x05000000
myValue EQU 1024


    AREA myCode, CODE, READONLY
    ENTRY
    EXPORT __main

__main
    LDR R10, =myAddress     ;initial R10 with myAddress
    LDR R11, =myValue       ;initial R11 with myValue
    STR R11, [R10]          ;store myValue in the memory cell that myAddress point that

    LSR R0, R11, #1         ;1 bit shifted to right value save in R0 destination Register
    LSR R1, R11, #2         ;2 bit shifted to right value save in R1 destination Register
    LSR R2, R11, #3         ;3 bit shifted to right value save in R2 destination Register
    LSR R3, R11, #4         ;4 bit shifted to right value save in R3 destination Register
    LSR R4, R11, #5         ;5 bit shifted to right value save in R4 destination Register
    LSR R5, R11, #6         ;6 bit shifted to right value save in R5 destination Register
    LSR R6, R11, #7         ;7 bit shifted to right value save in R6 destination Register
    LSR R7, R11, #8         ;8 bit shifted to right value save in R7 destination Register
    LSR R8, R11, #9         ;9 bit shifted to right value save in R8 destination Register
    LSR R9, R11, #10        ;10 bit shifted to right value save in R9 destination Register

here B here
    END
```

در این حالت نیز نتیجه مانند قبل است:



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. درصورتی 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 درصورت 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 باهاش زیاد کار دارید، مثلا رجیستر یکی از پورت‌های IO که میدانیم این رجیسترها جدا از 16 تا رجیستر داخلی پردازنده میباشد و هرکدام یک آدرس 32 بیتی برای خودشون دارند. برای اینکه مجبور نباشیم این آدرس ها 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 از 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 هرجا 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.

مثلا 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 صفحه6 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
```

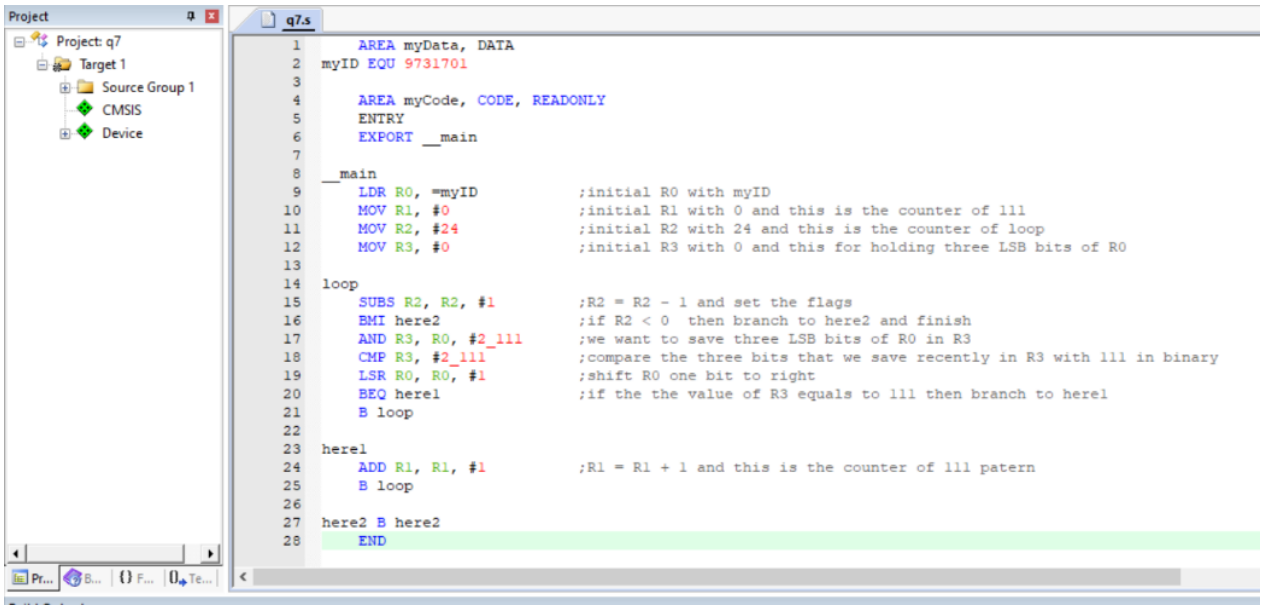
7) برنامه‌ای بنویسید که شماره دانشجویی شما را در R0 قرار دهد و تعداد دفعاتی که الگوی 111 در آن تکرار شده را در R1 بنویسد.

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.

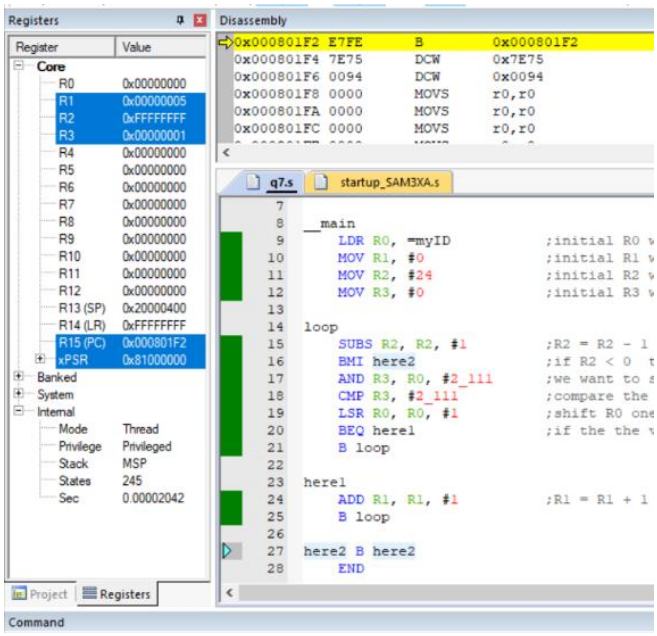
شماره دانشجویی من 9731701 است که به باینری برابر با 100101000111111001110101 میباشد که 24 بیت است. تعداد الگوهای 111 در آن برابر با 5 میباشد.

توضیحات: درابتدا شماره دانشجویی را در R0 ریختم. رجیسترهای دیگر نیز مقدار دهی اولیه شده اند و درواقع در R1 قرار است تعداد الگوهای111 را بشماریم پس مقدار اولیه 0 میخواهد. در R2 نیز تعداد ارقام شماره دانشجویی (به باینری)را نگه میداریم که 24 میباشد و حلقه ی ما باید به تعداد آن تکرار شود.

کد به صورت زیر میباشد:



همچنین مقادیر رجیسترها به صورت زیر میباشد:



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 می‌باشد.
- سوالات خود را می‌توانید 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 مودل بارگزاری کنید.
  - نمونه: HW5 -9731063
- فایل زیپ ارسال **نکنید**.