توضیحات درمورد دستورات MOV ، STR و LDR :

MOV : به کمک دستور MOV R0,OP2 می‌توان یک داده 8 بیتی constant را در یک رجیستر ریخت (کپی کرد) یا اینکه داده داخل یک رجیستر را در یک رجیستر دیگر می‌توان برنیم.

LDR : به کمک دستور LDR Rd,[Rx] می‌توان محتوای که در آدرسی که Rx به آن اشاره می‌کند را در رجیستر Rd قرار داد. LDR به اندازه یک Word دیتا می‌خواند

حال LDR به صورت شبه instraction یا همان Pseudo instruction باشد که در این صورت ما می‌توانیم یک 32 بیتی که constant هست را در یک رجیستر بریزیم که ساختار آن به شکل زیر است:

LDR Rd ,=32-bit value

STR : به کمک دستور STR Rd,[Rx] می‌توانیم دیتایی که در آدرسی از رجیستر Rx قرار دارد (آدرس) را در رجیستر Rd (رجیستر مقصد) بریزیم.

- ایده‌ای برای پیاده‌سازی تابع تاخیر داشته باشید.

در کد زده شده همه‌شان داریم که تابع delay به صورت زیر می‌توان پیاده‌سازی نمود:

```
delay
    mov R4,#0          ; یک رجیستر R4 داریم که مقدار 0 را در آن قرار دادیم.
    LDR R5,=0x00B0000  ; در رجیستر R5 هم یک عدد بزرگی را قرار می‌دهیم
loop                   ; سپس یک loop هر بار یک واحد به R4
    ADD R4,R4,#1       ; اضافه می‌کنیم و با R5 مقایسه می‌کنیم که اگر
    cmp R4,R5          ; برابر نباشند دوباره لوپ را اجرا می‌کنیم. این روند
    BNE loop           ; تکرار می‌شود تا مقدار R4 با R5 برابر شود.
    BX LR
    END
```

• در رابطه با دستورات build، batch build، Stop build و translate توضیحاتی ارائه دهید.

- Build: فایل‌هایی که modify شده را ترجمه می‌کند و برنامه را ایجاد می‌کند. (F7)
- Batch Build: دستورات Build مربوط به target های انتخاب شده یا پروژه یا چند پروژه را اجرا می‌کند.
- Stop Build: روند Build را متوقف می‌کند.
- Translate: فایل Active را ترجمه می‌کند. (با استفاده از Ctrl+F7)

• راجع به دو بخش reset handler و Interrupt vector توضیح دهید.

- reset handler: بعد از ریست شدن CPU اجرا می‌شود و تابع System Init را صدا می‌زند.
- در Interrupt vector، تابع‌هایی قرار دارند که روش مشخصی را پیاده‌سازی می‌کنند، این روش‌ها به دستگاه‌هایی که CPU به آن‌ها متصل می‌شود بستگی دارد.

حال اگر بخواهیم فایل Startup را با device جدید منطبق کنیم => interrupt vector table را
کد داریم خاصیت ~ های مخصوص interrupt handler دستگاه را به روزرسانی کنیم.
← <Interrupt name>_IRQHandler

حال دیگر Start up پردازنده‌ای که داریم interrupt vector مربوط به وقفه‌های جانبی ما را به صورت زیر پیاده‌سازی می‌کند:

.long <Device Interrupt first>_Handler
⋮
.long <Device Interrupt Last>_Handler

← که به آن interrupt handler اضافه می‌شود.

Interrupt vector به شکل زیر پیاده‌سازی شده است.

__Vectors:

Stack ↲

```
.long    __INITIAL_SP      →   Stack Pointer را مقداردهی اولیه کند
.long    Reset_Handler
.long    NMI_Handler
.long    HARDFault_Handler
.long    memmanage_Handler
.long    BusFault_Handler
.long    UsageFault_Handler
.long    SecureFault_Handler
.long    0
.long    0
.long    0
.long    SVC_Handler
.long    Debugmon_Handler
.long    0
.long    PendSV_Handler
.long    SysTick_Handler
```

Scanned with CamScanner

قسمت مربوط به Reset Handler هم به شکل زیر پیاده سازی می شود:

Reset_Handler:

```
LDR   R0 = __INITIAL_SP
MSR   PSP, R0
LDR   R0 = __STACK_LIMIT
MSR   msplim, R0
MSR   psplim, R0
LDR   R0 = __STACK_SEAL
LDR   R1, = 0xFEF5EDA5U
STRD  R1, R1, [R0, #0]
BL    SystemInit
BL    __main
```

هر زمان که cpu ریست شود اجرا شود. در این بخش تابع SystemInit صدا زده شده و سپس __main که حاوی دستورات اصلی است اجرا شود.

حسین سادات هاشم پور - 9831106