به نام خدا

12/29/2021

گزارش آزمایش شماره 11 (اسمبلی 1)

آزمایشگاه ریزپردازنده و زبان اسمبلی

محمد جواد زندیه , ابوالفضل بکیاسای کیوی
دانشگاه صنعتی امیرکبیر دانشکده مهندسی کامپیوتر

سوال: توضیحات مختصری در مورد دستورات LDR, MOV, STR بدهید

دستور STR :

STR Rd, [Rx]

این دستور محتوای یک ثبات پردازنده به نام Rd را به خانه ای از حافظه با آدرس Rx می ریزد(Rx نیز خود یک ثبات پردازنده است که حاوی یک آدرس بین 0x00000000 و 0xFFFFFFFF می باشد.)

دستور LDR:

LDR Rd, [Rx]

محتوای خانه از حافظه با آدرس Rx را در ثباتی از پردازنده به نام Rd می ریزد

دستور MOV:

MOV Rn, Op2

مقدار Op2 را در ثبات Rn می ریزد(Op2 می تواند یک immediate باشد و اینکه یک ثبات مثلا Rm باشد که محتوای آنرا بخواهیم در Rn بریزیم)

*تمامی ثبات هایی که گفته می شود منظور ثبات های پردازنده می باشد.

سوال : ایده ای برای پیاده سازی تابع تاخیر ارائه دهید

می توان یک شمارش (counting) مثلا 0 تا 1000 انجام داد و این مدت که پردازنده مشغول شمارش است به عنوان یک delay در برنامه استفاده شود، در واقع پردازنده را به یک کار بیهوده مجبور کنیم تا مدت زمان مشخص شده بگذرد و هرچه شمارش طولانی تر باشد تاخیر کم تر می شود و بالعکس

سوال : به پرسش های زیر پاسخ دهید

بخش یک : بخش های stop build , batch build , rebuild , build , translate ابزار هایی که به هنگام ساختن کد مورد استفاده قرار می گیرند هستند پس از تغییر کد از دستور rebuild استفاده می کنیم. **با مطالعه در رابطه با بقیه دستور های گفته شده توضیحات مختصری ارائه دهید.**

دستور stop build برای توقف ترجمه کد اسمبلی ما به زبان ماشین می باشد، زمانی که اروری را متوجه شویم از آن استفاده می کنیم Stopping Build When Warnings Are Detected

دستور build برای ترجمه کد به زبان ماشین و تولید فایل های مشخص شده در تنظیمات از جمله فایل hex. می باشد Click the **Build** button to translate all source files and link the application.
دستور

دستور batch build فایل ها را به صورت دسته ای ترجمه می کند یعنی چندین فایل را که از قبل باید انتخاب کرده باشیم

دستور translate فایل های فعال کنونی را به زمان ماشین ترجمه می کند

ما نیاز به کد اسمبلی از پیش آماده ی startup داریم این کد به منظور تعریف کردن میکروکنترلر استفاده می شود و باید با طراحی سخت افزاری ما مطابقت داشته باشد. می توانید در رابطه با بخش های مختلف این کد با مراجعه به اینجا اطلاعات بیشتری پیدا کنید.
راجع به دو بخش reset-handler و interrupt vector توضیح مختصری ارائه دهید.

```
; Reset Handler

Reset_Handler   PROC
                EXPORT  Reset_Handler             [WEAK]
                IMPORT  SystemInit
                IMPORT  __main
                LDR     R0, =SystemInit
                BLX     R0
                LDR     R0, =__main
                BX      R0
                ENDP
```

The reset handler is normally **a short module coded in assembler that executes immediately on system startup**. As a minimum, your reset handler initializes stack pointers for the modes that your application is running in.

بخش reset handler زمانی که سیستم آغاز به کار می کند اجرا می شود. این تکه کد اسمبلی عملیات مقدار دهی اولیه به stack pointer برای مودی که میخواهید با آن شروع به کار کنید را می دهد.
در واقع وظیفه اصلی reset handler تنظیم کلاک سیستم و در نهایت پرش به تابع main می باشد.

```
; Vector Table Mapped to Address 0 at Reset

                AREA    RESET, DATA, READONLY
                EXPORT  __Vectors

__Vectors       DCD     __initial_sp              ;  0: Top of Stack
                DCD     Reset_Handler             ;  1: Reset Handler
                DCD     NMI_Handler               ;  2: NMI Handler
                DCD     HardFault_Handler         ;  3: Hard Fault Handler
                DCD     MemManage_Handler         ;  4: MPU Fault Handler
                DCD     BusFault_Handler          ;  5: Bus Fault Handler
                DCD     UsageFault_Handler        ;  6: Usage Fault Handler
                DCD     0                         ;  7: Reserved
                DCD     0                         ;  8: Reserved
                DCD     0                         ;  9: Reserved
                DCD     0                         ; 10: Reserved
                DCD     SVC_Handler               ; 11: SVCall Handler
                DCD     DebugMon_Handler          ; 12: Debug Monitor Handler
                DCD     0                         ; 13: Reserved
                DCD     PendSV_Handler            ; 14: PendSV Handler
                DCD     SysTick_Handler           ; 15: SysTick Handler
```

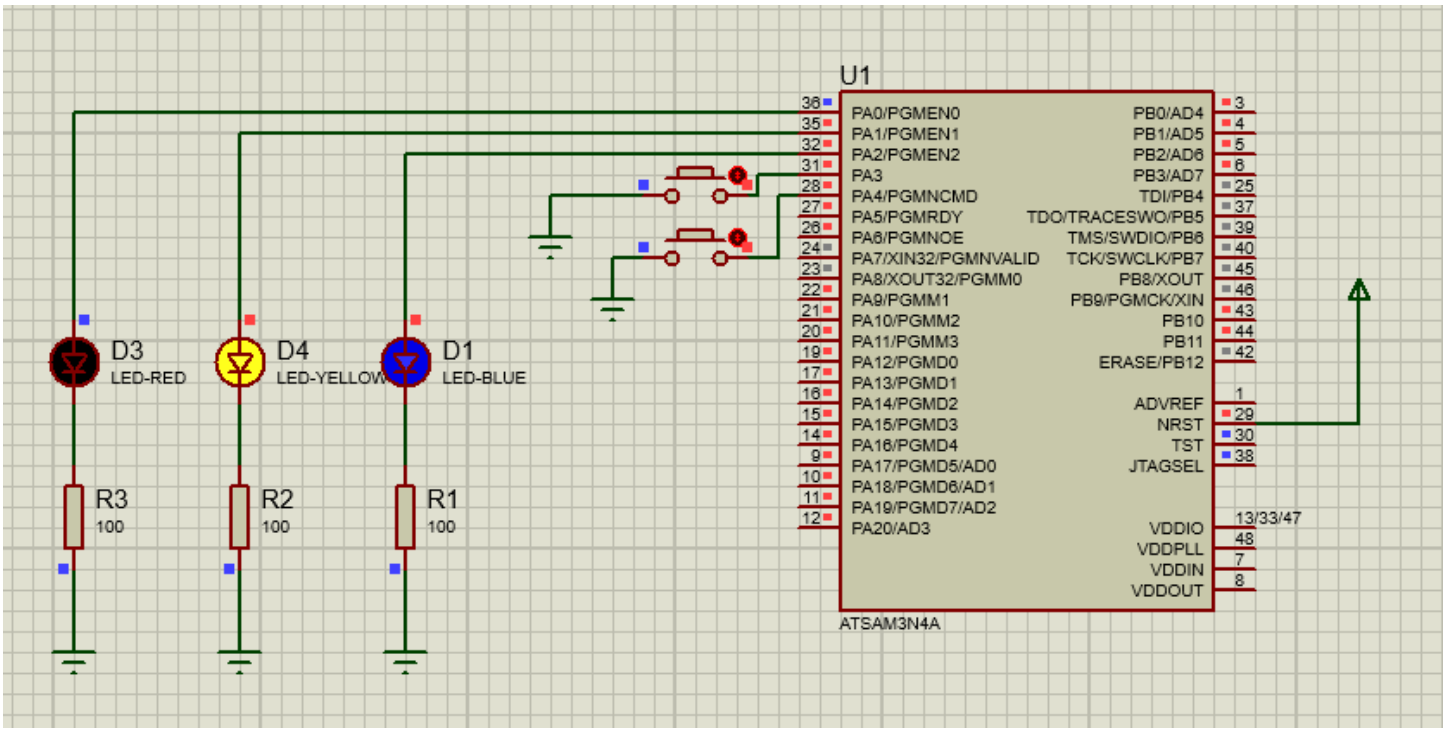
جدول interrupt ها که با نام interrupt vector شناخته می شود به منظور بررسی وقفه ها می باشد. در پردازنده arm موجود در درس دو مد کاری وجود دارد به نام های thread , exception. به صورت نرمال پردازنده در مد thread هست اما وقتی وقفه ای رخ می دهد با توجه به این جدول نوع وقفه و نحوه هندل کردن آن را متوجه شده و مد کاری پردازنده هم برای سرویس دهی به وقفه به مد exception تغییر می کند.

Role of Interrupt Vector Table in Interrupt Processing. ARM Cortex-M CPU has two modes of operation such as thread mode and exception. In normal execution, CPU runs in thread mode. But when an interrupt occurs the CPU transfers from thread mode to exception mode.

شرح کلی آزمایش:

شرح آزمایش:

این آزمایش شامل 3 عدد LED و 2 دکمه می باشد. در ابتدا LED ها خاموش می باشد. هنگامی که دکمه ی اول فشرده می شود. هر 3 LED شروع به چشمک زدن می کنند با این تفاوت که سرعت چشمک زدن LED ها از چپ به راست افزایش پیدا می کند. با فشردن دکمه ی دوم همه ی LED ها خاموش می شوند.



کد اسمبلی:

```
PIO_PER equ 0x400E0E00
PIO_OER equ 0x400E0E10
PIO_SODR equ 0x400E0E30
PIO_CODR equ 0x400E0E34

PIO_ISR equ 0x400E0E4C

    area myCode, code, readonly
    export __main
    entry

__main
    bl enable_pio

polling_interrupt_loop
    mov r4, #2_1000
    ldr r6, =PIO_ISR
    ldr r5, [r6]
    cmp r4, r5
    BEQ loop
    bl polling_interrupt_loop

button_check2
    mov r4, #2_10000
    ldr r6, =PIO_ISR
    ldr r5, [r6]
    cmp r4, r5
    beq polling_interrupt_loop
    bx lr


loop
    ; part 1
    bl led_on
    bl delay

    ; part 2
    mov r4, #2_001
    bl led_off
    bl delay

    ; part 3
    bl led_on
    mov r4, #2_010
    bl led_off
    bl delay

    ; part 4
    mov r4, #2_011
    bl led_off
    bl delay

    ; part 5
    bl led_on
    mov r4, #2_100
    bl led_off
    bl delay
```

```
57
58      ; part 6
59      mov r4, #2_101
60      bl led_off
61      bl delay
62
63      ; part 7
64      bl led_on
65      mov r4, #2_110
66      bl led_off
67      bl delay
68
69      ; part 8
70      mov r4, #2_111
71      bl led_off
72      bl delay
73
74
75      bl button_check2
76
77      bl loop
78
79  enable_pio
80      mov r4, #2_111
81
82      ldr r5, =PIO_PER
83      str r4, [r5]
84
85      ldr r5, =PIO_OER
86      str r4, [r5]
87
```

```
87
88      bx lr
89
90  led_on
91      mov r4, #2_111
92
93      ldr r5, =PIO_SODR
94      str r4, [r5]
95      bx lr
96
97  led_off
98      ;mov r4, #2_111
99
100     ldr r5, =PIO_CODR
101     str r4, [r5]
102
103     bx lr
104
105 delay
106     mov r4, #0
107     ldr r5, =0x00A00A
108
109 delay_loop
110     add r4, r4, #1
111
112     cmp r4, r5
113     bne delay_loop
114     bx lr
115
116     end
```