第14次實習課

學生：林培瑋

2024 Advanced Mixed-Operation System (AMOS) Lab.

Tamkang University
Department of Electrical and Computer Engineering
No.151, Yingzhuan Rd., Tamsui Dist., New Taipei City 25137, Taiwan (R.O.C.)

Priority

2024 Advanced Mixed-Operation System (AMOS) Lab.

Tamkang University
Department of Electrical and Computer Engineering
No.151, Yingzhuan Rd., Tamsui Dist., New Taipei City 25137, Taiwan (R.O.C.)

Priority Width Reading (using LSR with test pattern LSR)

```
        LDR     r0, =0xE000E484
        MOV     r2, #0xFF
        STRB    r2, [r0]

        LDRB    r2, [r0]
        MOV     r1, #0
        MOV     r3, #0x80
LOOP    CMP     r1, #8
        BEQ     OUT
        TST     r2, r3
        BEQ     OUT
        ADD     r1, #1
        LSR     r3, #1
        B       LOOP
OUT
```

Step1. 將最低8個位元存入1，un-implemented bits保持為0。

Step2. 將最低8個位元取出來計算寬度。

r1 = counter，初始為0，測試到的bit為1，則counter+1
r3 = test pattern

Step3. 若counter = 8，代表width最大，則跳出迴圈。

Step4. 從最高位元開始測試，若測試到的bit為0，則跳出迴圈。

Step5. 若測試到的bit為1，則counter+1。
Step6. 每次測試完，test pattern向右移一個bit。

Priority Width Reading (using LSL with test pattern LSL)

```
        LDR     r0, =0xE000E484
        MOV     r2, #0xFF
        STRB    r2, [r0]

        LDRB    r2, [r0]
        MOV     r2, #0xFC
        MOV     r1, #8
        MOV     r3, #1
LOOP    CMP     r1, #0
        BEQ     OUT
        TST     r2, r3
        BNE     OUT
        SUB     r1, #1
        LSL     r3, #1
        B       LOOP
OUT
```

Step1. 將最低8個位元存入1，un-implemented bits保持為0。

Step2. 將最低8個位元取出來計算寬度。

將r2 width設定成6，r2 = 0xFC

r1 = counter，初始為8，測試到的bit為0，則counter-1

r3 = test pattern

Step3. 若counter = 0，代表width最小，則跳出迴圈。

Step4. 從最低位元開始測試，若測試到的bit為1，則跳出迴圈。

Step5. 若測試到的bit為0，則counter-1。

Step6. 每次測試完，test pattern向左移一個bit。

Priority Width Reading (using LSR without test pattern LSR)

```
        LDR     r0, =0xE000E484
        MOV     r2, #0xFF
        STRB    r2, [r0]

        LDRB    r2, [r0]
        MOV     r2, #0xF0
        MOV     r1, #8
        MOV     r3, #1
LOOP    CMP     r1, #0
        BEQ     OUT
        TST     r2, r3
        BNE     OUT
        SUB     r1, #1
        LSR     r2, #1
        B       LOOP
OUT
```

Step1. 將最低8個位元存入1，un-implemented bits保持為0。

Step2. 將最低8個位元取出來計算寬度。

將r2 width設定成4，r2 = 0xF0

r1 = counter，初始為8，測試到的bit為0，則counter-1

r3 = test pattern

Step3. 若counter = 0，代表width最小，則跳出迴圈。

Step4. 從最低位元開始測試，若測試到的bit為1，則跳出迴圈。

Step5. 若測試到的bit為0，則counter-1。

Step6. 每次測試完，interrupt priority register向右移一個bit。

15.7 Interrupts

2024 Advanced Mixed-Operation System (AMOS) Lab.

Tamkang University

Department of Electrical and Computer Engineering

No.151, Yingzhuan Rd., Tamsui Dist., New Taipei City 25137, Taiwan (R.O.C.)

TABLE 15.1 vs. TABLE 15.5

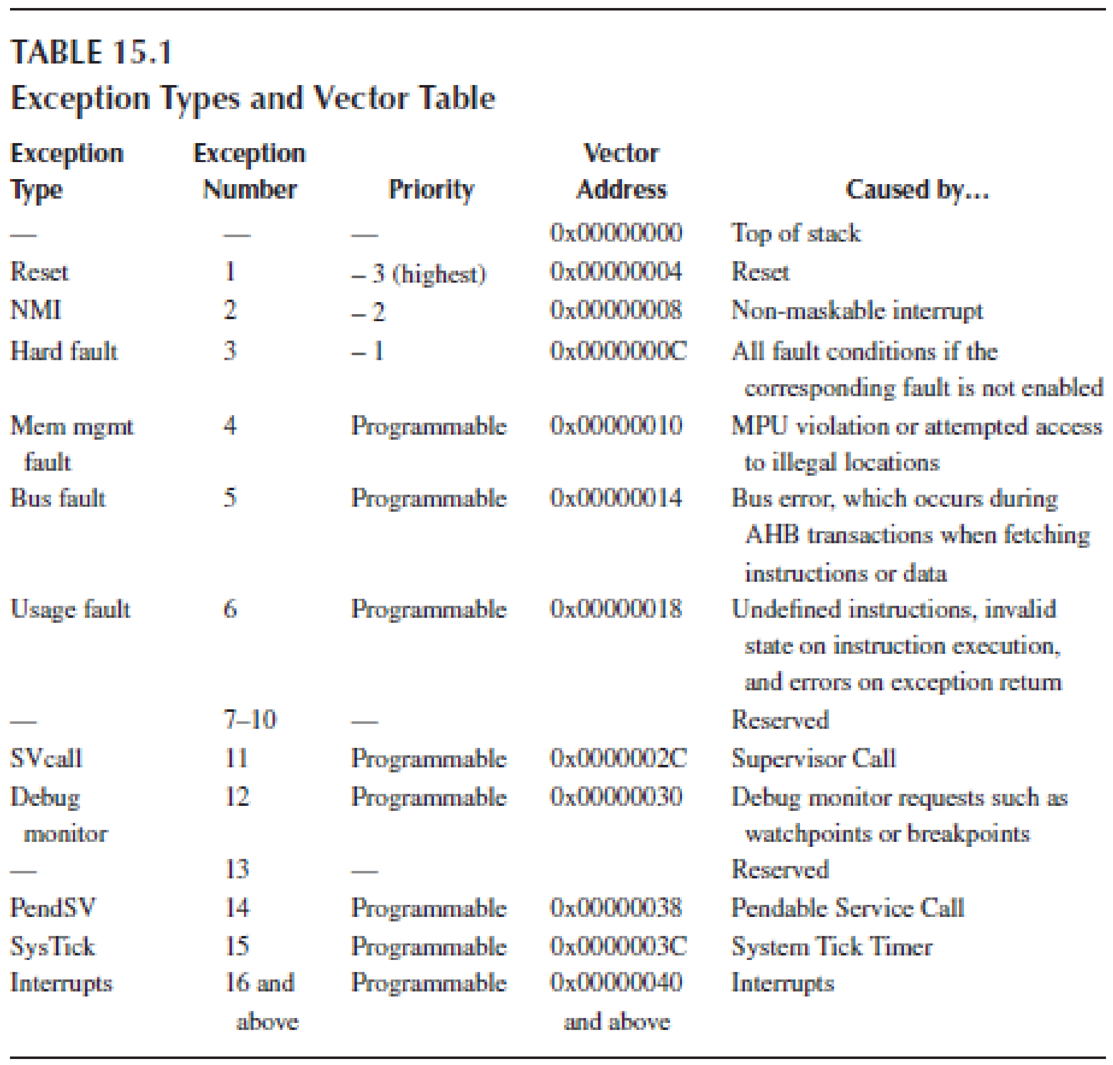


TABLE 15.1
Exception Types and Vector Table

Exception Type	Exception Number	Priority	Vector Address	Caused by...
—	—	—	0x00000000	Top of stack
Reset	1	– 3 (highest)	0x00000004	Reset
NMI	2	– 2	0x00000008	Non-maskable interrupt
Hard fault	3	– 1	0x0000000C	All fault conditions if the corresponding fault is not enabled
Mem mgmt fault	4	Programmable	0x00000010	MPU violation or attempted access to illegal locations
Bus fault	5	Programmable	0x00000014	Bus error, which occurs during AHB transactions when fetching instructions or data
Usage fault	6	Programmable	0x00000018	Undefined instructions, invalid state on instruction execution, and errors on exception return
—	7–10	—		Reserved
SVcall	11	Programmable	0x0000002C	Supervisor Call
Debug monitor	12	Programmable	0x00000030	Debug monitor requests such as watchpoints or breakpoints
—	13	—		Reserved
PendSV	14	Programmable	0x00000038	Pendable Service Call
SysTick	15	Programmable	0x0000003C	System Tick Timer
Interrupts	16 and above	Programmable	0x00000040 and above	Interrupts

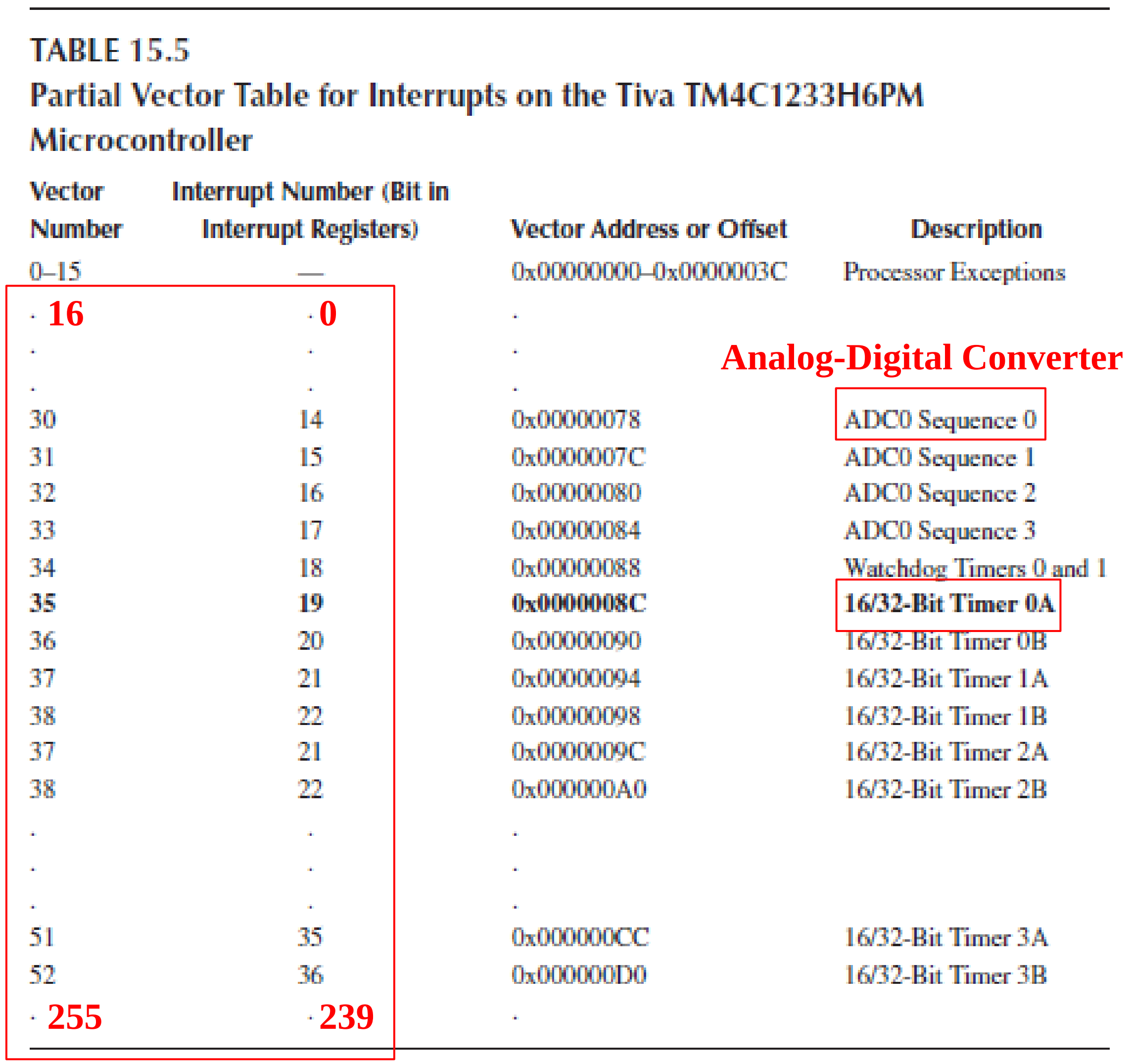


TABLE 15.5
Partial Vector Table for Interrupts on the Tiva TM4C1233H6PM Microcontroller

Vector Number	Interrupt Number (Bit in Interrupt Registers)	Vector Address or Offset	Description
0–15	—	0x00000000–0x0000003C	Processor Exceptions
. 16	. 0	.	
.	.	.	
.	.	.	
30	14	0x00000078	ADC0 Sequence 0
31	15	0x0000007C	ADC0 Sequence 1
32	16	0x00000080	ADC0 Sequence 2
33	17	0x00000084	ADC0 Sequence 3
34	18	0x00000088	Watchdog Timers 0 and 1
35	**19**	**0x0000008C**	**16/32-Bit Timer 0A**
36	20	0x00000090	16/32-Bit Timer 0B
37	21	0x00000094	16/32-Bit Timer 1A
38	22	0x00000098	16/32-Bit Timer 1B
37	21	0x0000009C	16/32-Bit Timer 2A
38	22	0x000000A0	16/32-Bit Timer 2B
.	.	.	
.	.	.	
.	.	.	
51	35	0x000000CC	16/32-Bit Timer 3A
52	36	0x000000D0	16/32-Bit Timer 3B
. 255	. 239	.	

Analog-Digital Converter

Hardware Interrupts

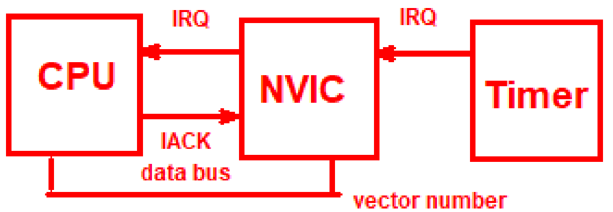
EXAMPLE 15.4

you need to program the same operation in assembly at least once! That's where we begin.

timer	=	counter
down timer		down counter
up timer		up counter

16-bit 0xFFFF $\longrightarrow$ 0

32-bit 0xFFFFFFFF $\longrightarrow$ 0



Exception Entry：
1. stacking
2. Table lookup
PC = vector
3. LR = EXC_RETURN

EXAMPLE 15.4

Let's look at an example of a relatively simple interrupt being caused by a timer counting down to zero. You can see from Table 15.5 that the twelve timers are all given their own vector number, interrupt number, and vector address. We'll set up one 16-bit timer, Timer 0A, to count down from 0xFFFF to 0, sending an interrupt to the NVIC, alerting the core that the timer expired. The processor will acknowledge the interrupt and jump to a handler routine. Once inside the interrupt handler, we'll put a value into a core register and then spin in an infinite loop so that we can see the process happen using a debugger. In this example, we'll use the Tiva Launchpad as a target. As a necessity, the timer must be configured as a memory-mapped peripheral, but this gives us a preview of Chapter 16.

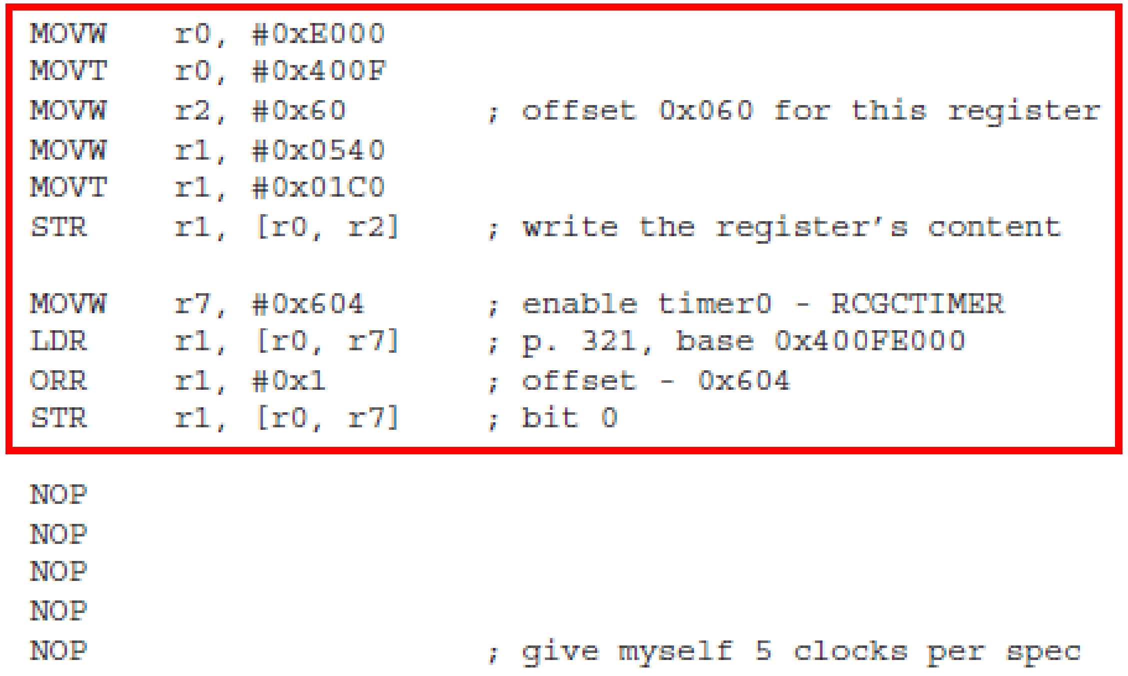
In order to configure the interrupts, the following must take place: 補：Real time programs

EXAMPLE 15.4 - 系統設定

2024 Advanced Mixed-Operation System (AMOS) Lab.

Tamkang University
Department of Electrical and Computer Engineering
No.151, Yingzhuan Rd., Tamsui Dist., New Taipei City 25137, Taiwan (R.O.C.)

EXAMPLE 15.4



```
MOVW    r0, #0xE000
MOVT    r0, #0x400F
MOVW    r2, #0x60           ; offset 0x060 for this register
MOVW    r1, #0x0540
MOVT    r1, #0x01C0
STR     r1, [r0, r2]        ; write the register's content

MOVW    r7, #0x604          ; enable timer0 - RCGCTIMER
LDR     r1, [r0, r7]        ; p. 321, base 0x400FE000
ORR     r1, #0x1            ; offset - 0x604
STR     r1, [r0, r7]        ; bit 0

NOP
NOP
NOP
NOP
NOP                         ; give myself 5 clocks per spec
```

系統設定

EXAMPLE 15.4 (The system clocks(16MHz) must be set up)

- The system clocks on processor must be set up, similar to the code we'll use in Chapter 16 for the GPIO example.

Using the Code Composer Studio tools, you can create a source code file with the following code:

LPC2104：LDR r0, =0x400FE000 ;literal pool

兩者不可對調

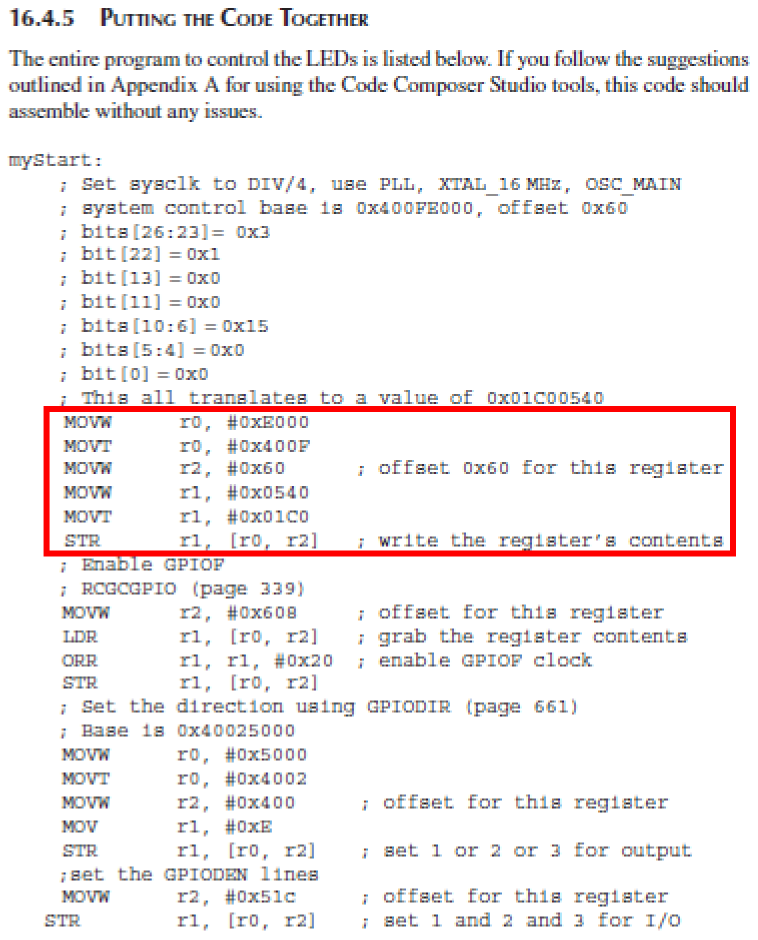
MOVT r0, #0x400F ; r0 = 0x400F0000
MOVW r0, #0xE000 ; r0 = 0x0000E000

Cortex獨有的指令

```
Word   MOVW    r0, #0xE000       ; 0x0000E000 (低位元存入，高位元清為0)
Top    MOVT    r0, #0x400F       ; 0x400FE000 (低位元不變，高位元存入)
       MOVW    r2, #0x60         ; offset 0x060 for this register
       MOVW    r1, #0x0540       System Control Registers
       MOVT    r1, #0x01C0
       STR     r1, [r0, r2]      ; write the register's content

       MOVW    r7, #0x604        ; enable timer0 - RCGCTIMER
       LDR     r1, [r0, r7]      ; p. 321, base 0x400FE000
       ORR     r1, #0x1          ; offset - 0x604
       STR     r1, [r0, r7]      ; bit 0
```

參考 P362 16.4.5

- The system clocks on processor must be set up, similar to the code we'll use in Chapter 16 for the GPIO example.



16.4.5 PUTTING THE CODE TOGETHER

The entire program to control the LEDs is listed below. If you follow the suggestions outlined in Appendix A for using the Code Composer Studio tools, this code should assemble without any issues.

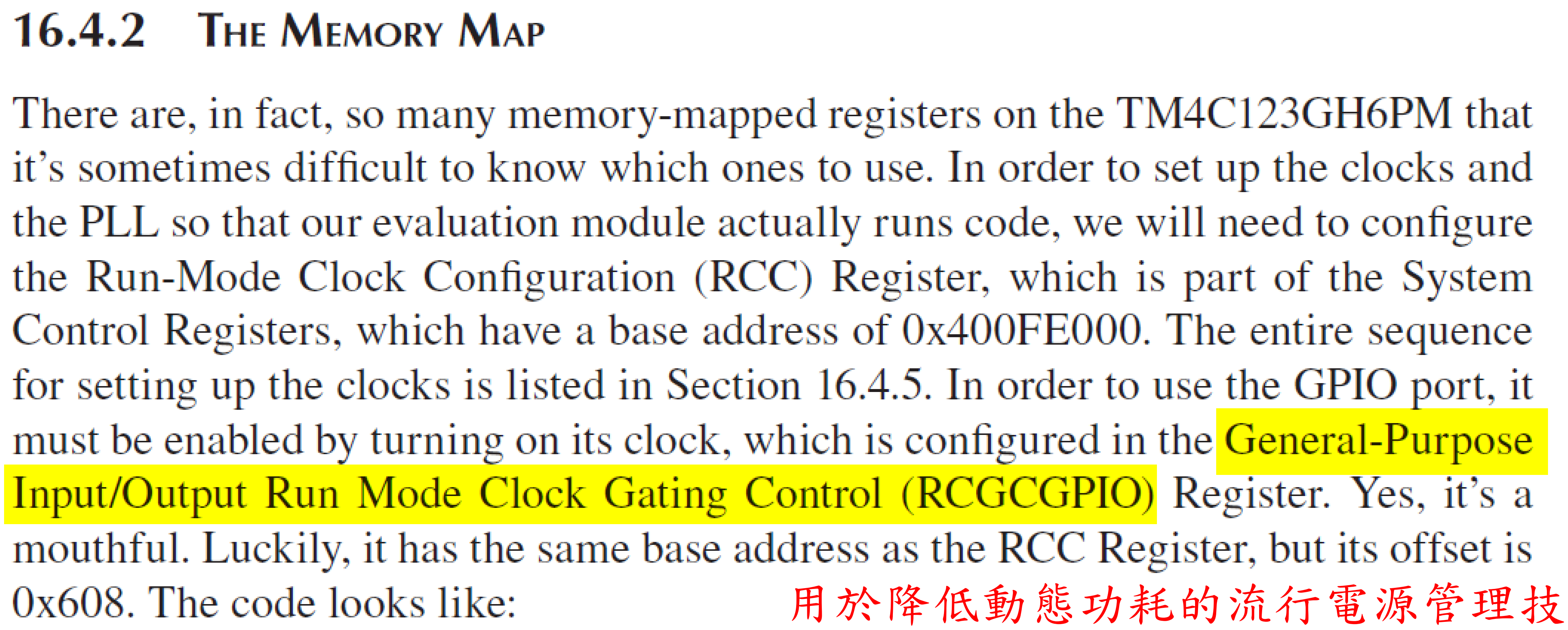
```
myStart:
    ; Set sysclk to DIV/4, use PLL, XTAL_16 MHz, OSC_MAIN
    ; system control base is 0x400FE000, offset 0x60
    ; bits[26:23]= 0x3
    ; bit[22] = 0x1
    ; bit[13] = 0x0
    ; bit[11] = 0x0
    ; bits[10:6] = 0x15
    ; bits[5:4] = 0x0
    ; bit[0] = 0x0
    ; This all translates to a value of 0x01C00540
    MOVW      r0, #0xE000
    MOVT      r0, #0x400F
    MOVW      r2, #0x60      ; offset 0x60 for this register
    MOVW      r1, #0x0540
    MOVT      r1, #0x01C0
    STR       r1, [r0, r2]   ; write the register's contents
    ; Enable GPIOF
    ; RCGCGPIO (page 339)
    MOVW      r2, #0x608     ; offset for this register
    LDR       r1, [r0, r2]   ; grab the register contents
    ORR       r1, r1, #0x20  ; enable GPIOF clock
    STR       r1, [r0, r2]
    ; Set the direction using GPIODIR (page 661)
    ; Base is 0x40025000
    MOVW      r0, #0x5000
    MOVT      r0, #0x4002
    MOVW      r2, #0x400      ; offset for this register
    MOV       r1, #0xE
    STR       r1, [r0, r2]    ; set 1 or 2 or 3 for output
    ;set the GPIODEN lines
    MOVW      r2, #0x51c      ; offset for this register
  STR         r1, [r0, r2]    ; set 1 and 2 and 3 for I/O
```

EXAMPLE 15.4 (Enable timer0 - RCGCTIMER)

- The clocks must be enabled to the interrupt block, specifically, using the RCGCTIMER register. Now that the timer is enabled, we can actually write to the memory-mapped registers within it. If the interrupt block is not enabled, any attempts to write to the memory-mapped registers results in a hard fault.

```
MOVW    r7, #0x604       ; enable timer0 - RCGCTIMER
LDR     r1, [r0, r7]     ; p. 321, base 0x400FE000
ORR     r1, #0x1         ; offset - 0x604
STR     r1, [r0, r7]     ; bit 0
```

參考 P359 16.4.2



16.4.2 The Memory Map

There are, in fact, so many memory-mapped registers on the TM4C123GH6PM that it's sometimes difficult to know which ones to use. In order to set up the clocks and the PLL so that our evaluation module actually runs code, we will need to configure the Run-Mode Clock Configuration (RCC) Register, which is part of the System Control Registers, which have a base address of 0x400FE000. The entire sequence for setting up the clocks is listed in Section 16.4.5. In order to use the GPIO port, it must be enabled by turning on its clock, which is configured in the General-Purpose Input/Output Run Mode Clock Gating Control (RCGCGPIO) Register. Yes, it's a mouthful. Luckily, it has the same base address as the RCC Register, but its offset is 0x608. The code looks like:

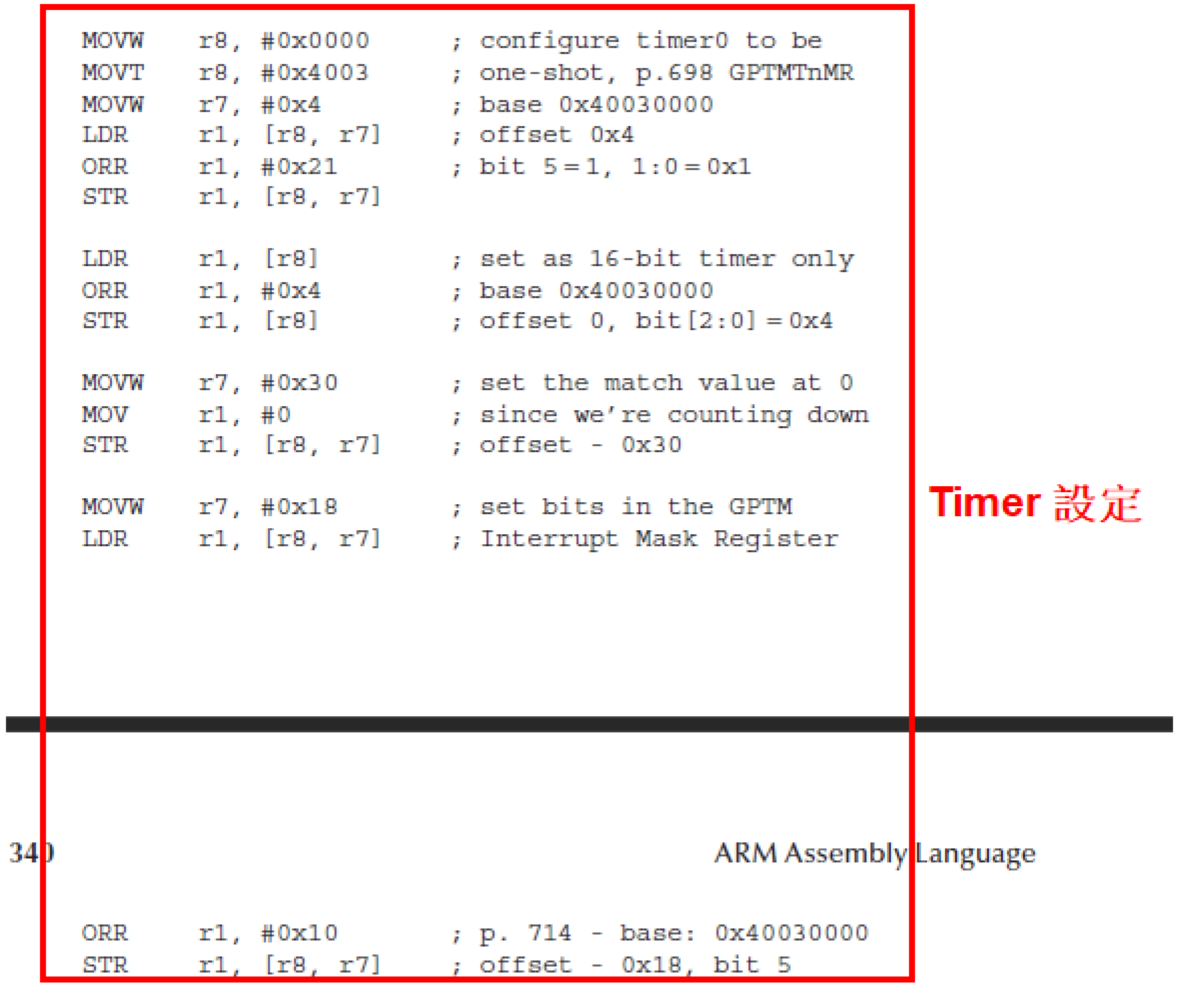
用於降低動態功耗的流行電源管理技術。它通過在電路或其子部分未使用或忽略時移除時鐘信號來實現。

EXAMPLE 15.4 - Timer設定

2024 Advanced Mixed-Operation System (AMOS) Lab.

Tamkang University
Department of Electrical and Computer Engineering
No.151, Yingzhuan Rd., Tamsui Dist., New Taipei City 25137, Taiwan (R.O.C.)

EXAMPLE 15.4



```
MOVW    r8, #0x0000       ; configure timer0 to be
MOVT    r8, #0x4003       ; one-shot, p.698 GPTMTnMR
MOVW    r7, #0x4          ; base 0x40030000
LDR     r1, [r8, r7]      ; offset 0x4
ORR     r1, #0x21         ; bit 5=1, 1:0=0x1
STR     r1, [r8, r7]

LDR     r1, [r8]          ; set as 16-bit timer only
ORR     r1, #0x4          ; base 0x40030000
STR     r1, [r8]          ; offset 0, bit[2:0]=0x4

MOVW    r7, #0x30         ; set the match value at 0
MOV     r1, #0            ; since we're counting down
STR     r1, [r8, r7]      ; offset - 0x30

MOVW    r7, #0x18         ; set bits in the GPTM
LDR     r1, [r8, r7]      ; Interrupt Mask Register
```

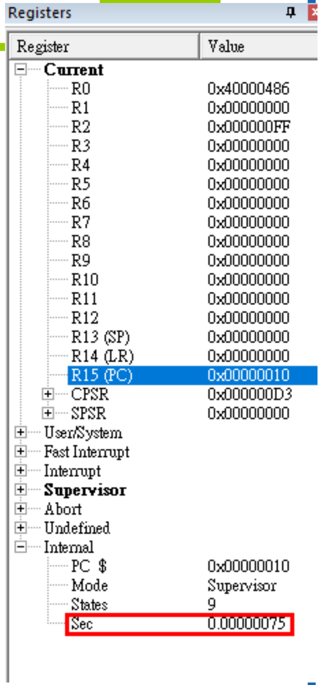
340   ARM Assembly Language

```
ORR     r1, #0x10         ; p. 714 - base: 0x40030000
STR     r1, [r8, r7]      ; offset - 0x18, bit 5
```

Timer 設定

EXAMPLE 15.4 (Configure timer0 to be one-shot)

- Rather than having a periodic timer, we will configure it to be a one-shot timer. To do this, we must configure the GPTMTnMR register.



```
Registers
Register          Value
⊟ Current
    R0            0x40000486
    R1            0x00000000
    R2            0x000000FF
    R3            0x00000000
    R4            0x00000000
    R5            0x00000000
    R6            0x00000000
    R7            0x00000000
    R8            0x00000000
    R9            0x00000000
    R10           0x00000000
    R11           0x00000000
    R12           0x00000000
    R13 (SP)      0x00000000
    R14 (LR)      0x00000000
    R15 (PC)      0x00000010
  ⊞ CPSR          0x000000D3
  ⊞ SPSR          0x00000000
⊞ User/System
⊞ Fast Interrupt
⊞ Interrupt
⊞ Supervisor
⊞ Abort
⊞ Undefined
⊟ Internal
    PC $          0x00000010
    Mode          Supervisor
    States        9
    Sec           0.00000075
```

可以由右側視窗中的Sec算出Sec/clock cycle

```
NOP    No operation
NOP
NOP
NOP
NOP                                    ; give myself 5 clocks per spec
```

→ 間隔5個Clock Cycles

EX：碼表(時間到停止)

```
MOVW    r8, #0x0000        ; configure timer0 to be
MOVT    r8, #0x4003        ; one-shot, p.698 GPTMTnMR
MOVW    r7, #0x4           ; base 0x40030000
LDR     r1, [r8, r7]       ; offset 0x4
ORR     r1, #0x21          ; bit 5=1, 1:0=0x1
STR     r1, [r8, r7]
```

Periodical (週期性的)
(時間到了，Reset後，繼續計數)

EXAMPLE 15.4 (Set as 16-bit timer only)

- The timer will be set up as a 16-bit timer, so the initial count will be (by a reset) set to 0xFFFF. By default, the timer will count down to zero, rather than up.

```
LDR    r1, [r8]          ; set as 16-bit timer only
ORR    r1, #0x4          ; base 0x40030000
STR    r1, [r8]          ; offset 0, bit[2:0]=0x4
```

EXAMPLE 15.4 (Set the match value)

- The timer will be set up as a 16-bit timer, so the initial count will be (by a reset) set to 0xFFFF. By default, the timer will count down to zero, rather than up.

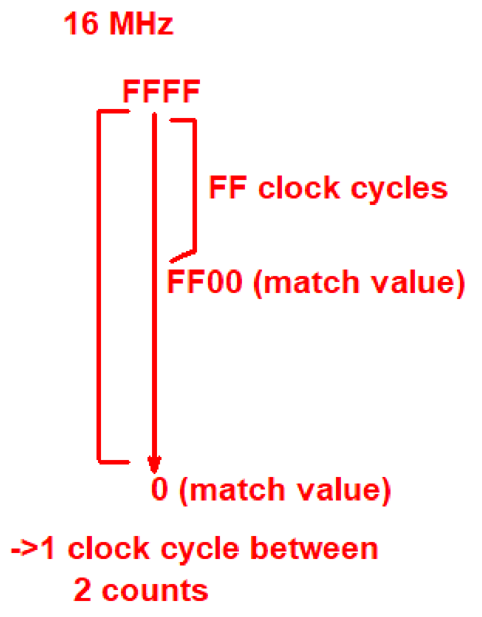
```
MOVW    r8, #0x0000       ; configure timer0 to be
MOVT    r8, #0x4003       ; one-shot, p.698 GPTMTnMR
MOVW    r7, #0x4          ; base 0x40030000
LDR     r1, [r8, r7]      ; offset 0x4
ORR     r1, #0x21         ; bit 5=1, 1:0=0x1
STR     r1, [r8, r7]

LDR     r1, [r8]          ; set as 16-bit timer only
ORR     r1, #0x4          ; base 0x40030000
STR     r1, [r8]          ; offset 0, bit[2:0]=0x4
```

設定match value，等同於設定倒數多久時間。

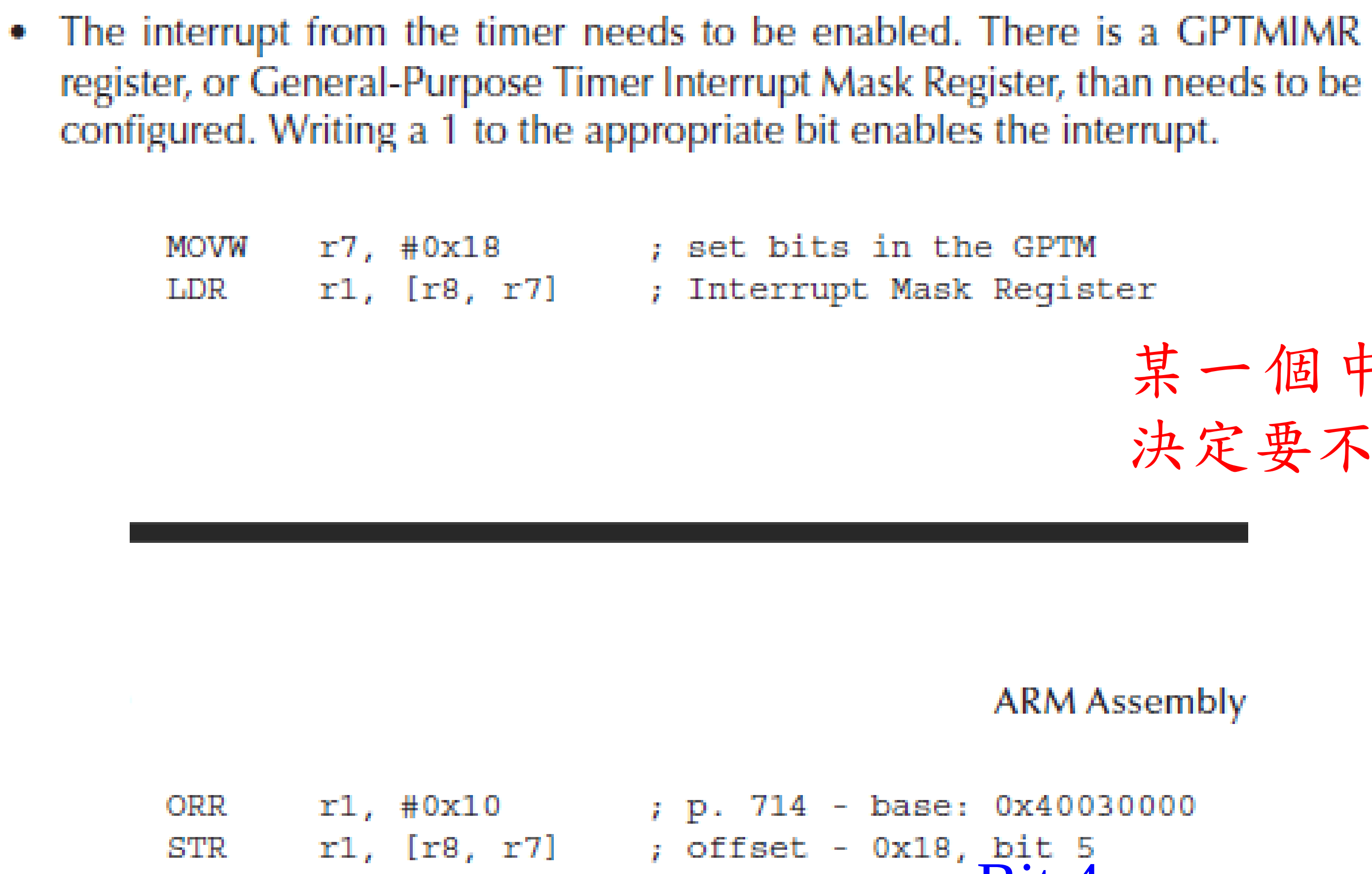
```
MOVW    r7, #0x30         ; set the match value at 0
MOV     r1, #0            ; since we're counting down
STR     r1, [r8, r7]      ; offset - 0x30
```

若match value設定成0，則需要倒數多少時間?(從Timer啟動後經過多少秒，產生中斷)

Ans：0xFFFF(clock cycles)/16MHz



EXAMPLE 15.4(Set bits in the GPTM Interrupt Mask Register)



- The interrupt from the timer needs to be enabled. There is a GPTMIMR register, or General-Purpose Timer Interrupt Mask Register, than needs to be configured. Writing a 1 to the appropriate bit enables the interrupt.

```
MOVW    r7, #0x18        ; set bits in the GPTM
LDR     r1, [r8, r7]     ; Interrupt Mask Register
```

ARM Assembly

```
ORR     r1, #0x10        ; p. 714 - base: 0x40030000
STR     r1, [r8, r7]     ; offset - 0x18, bit 5
```

某一個中斷產生時，CPU決定要不要忽略他。

Bit 4

設定Interrupt Mask Register，當Timer這個中斷產生時，讓CPU不要忽略它。

EXAMPLE 15.4 - NVIC設定

2024 Advanced Mixed-Operation System (AMOS) Lab.

Tamkang University
Department of Electrical and Computer Engineering
No.151, Yingzhuan Rd., Tamsui Dist., New Taipei City 25137, Taiwan (R.O.C.)

EXAMPLE 15.4 (Enable interrupt on timer0)

- Interrupts needs to be enabled from Timer 0A in the NVIC. Bit 19 of the Interrupt Set Enable register in the NVIC enables Timer 0A.

```
ORR     r1, #0x10          ; p. 714 - base: 0x40030000
STR     r1, [r8, r7]       ; offset - 0x18, bit 5
NVICBase = 0xE000E000
MOVW    r6, #0xE000        ; enable interrupt on timer0
MOVT    r6, #0xE000        ; p. 132, base 0xE000E000
MOVW    r7, #0x100         ; offset - 0x100, bit 19
MOV     r1, #(1<<19)       ; enable bit 19 for timer0
STR     ~~r1~~, [r6, r7]  (WRITE)
        r2
```

(READ)
LDR r2, [r6, r7] →

ORR r2, r1 →
(MODIFY)

EXAMPLE 15.4 - Timer設定

2024 Advanced Mixed-Operation System (AMOS) Lab.

EXAMPLE 15.4 (Start the timer)

- The timer needs to be started.

```
MOVW    r6, #0x0000      ; start the timer
MOVT    r6, #0x4003
MOVW    r7, #0xC
LDR     r1, [r6, r7]
ORR     r1, #0x1
STR     r1, [r6, r7]     ; go!!
```

Now that the NVIC, Timer 0A, and all of the control registers are programmed, we can write a very simple handler for the interrupt we are expecting:

```
IntDefaultHandler:
        MOVW    r10, #0xBEEF
        MOVT    r10, #0xDEAD
Spot
        B       Spot
```

Q&A

Thanks for your attention !!