CEG2400 - Microcomputer Systems

Lecture 7: Interrupts

Philip Leong

ARM Exceptions

Exceptions and Modes

- Exceptions arise whenever the normal flow of a program has to be halted temporarily
  - For example to service an interrupt from a peripheral.
- ARM supports 7 types of exception and has a privileged processor mode for each type of exception.
- ARM Exception vectors

Address	Exception	Mode in Entry
0x00000000	Reset	Supervisor
0x00000004	Undefined instruction	Undefined
0x00000008	Software Interrupt	Supervisor
0x0000000C	Abort (prefetch)	Abort
0x00000010	Abort (data)	Abort
0x00000014	Reserved	Reserved
0x00000018	IRQ	IRQ
0x0000001C	FIQ	FIQ

Exception Entry

- When handling an exception, the ARM7TDMI:
  - Preserves the address of the next instruction in the appropriate Link Register
  - Copies the CPSR into the appropriate SPSR
  - Forces the CPSR mode bits to a value which depends on the exception
  - Forces the PC to fetch the next instruction from the relevant exception vector
  - It may also set the interrupt disable flags to prevent otherwise unmanageable nestings of exceptions.
  - If the processor is in THUMB state when an exception occurs, it will automatically switch into ARM state when the PC is loaded with the exception vector address.

Entry/Exit Actions

- Reset
  - When the processor's Reset input is asserted
    - CPSR ⇐ Supervisor + I + F
    - PC ⇐ 0x00000000
- Undefined Instruction
  - If an attempt is made to execute an instruction that is undefined
    - LR_undef ⇐ Undefined Instruction Address + #4
    - PC ⇐ 0x00000004, CPSR ⇐ Undefined + I
    - Return with : MOVS pc, lr
- Prefetch Abort
  - Instruction fetch memory abort, invalid fetched instruction
    - LR_abt ⇐ Aborted Instruction Address + #4, SPSR_abt ⇐ CPSR
    - PC ⇐ 0x0000000C, CPSR ⇐ Abort + I
    - Return with : SUBS pc, lr, #4

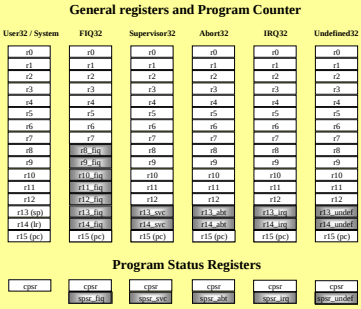
Entry/Exit Actions

- Data Abort
  - Data access memory abort, invalid data
    - LR_abt ⇐ Aborted Instruction + #8, SPSR_abt ⇐ CPSR
    - PC ⇐ 0x00000010, CPSR ⇐ Abort + I
    - Return with : SUBS pc, lr, #4 or SUBS pc, lr, #8
- Software Interrupt
  - Enters Supervisor mode
    - LR_svc ⇐ SWI Address + #4, SPSR_svc ⇐ CPSR
    - PC ⇐ 0x00000008, CPSR ⇐ Supervisor + I
    - Return with : MOV pc, lr

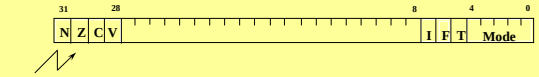
Entry/Exit Actions

- Interrupt Request
  - Externally generated by asserting the processor's IRQ input
    - LR_irq ⇐ PC - #4, SPSR_irq ⇐ CPSR
    - PC ⇐ 0x00000018, CPSR ⇐ Interrupt + I
    - Return with : SUBS pc, lr, #4
- Fast Interrupt Request
  - Externally generated by asserting the processor's FIQ input
    - LR_fiq ⇐ PC - #4, SPSR_fiq ⇐ CPSR
    - PC ⇐ 0x0000001C, CPSR ⇐ Fast Interrupt + I + F
    - Return with : SUBS pc, lr, #4
    - Handler @0x1C speeds up the response time

Recall registers



Recall program status regs



Copies of the ALU status flags (latched if the instruction has the "S" bit set).

* **Condition Code Flags**
N = **N**egative result from ALU flag.
Z = **Z**ero result from ALU flag.
C = ALU operation **C**arried out
V = ALU operation o**V**erflowed

* **Interrupt Disable bits.**
I = 1, disables the IRQ.
F = 1, disables the FIQ.

* **T Bit (Architecture v4T only)**
T = 0, Processor in ARM state
T = 1, Processor in Thumb state

Mode bits

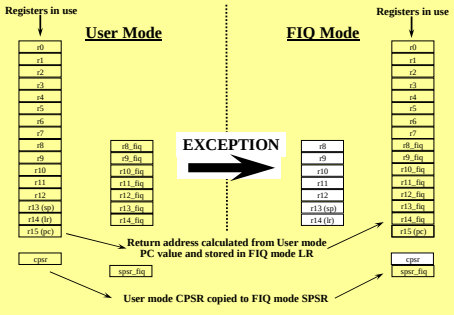
M[4:0]	Mode	Visible THUMB state registers	Visible ARM state registers
10000	User	R7..R0, LR, SP PC, CPSR	R14..R0, PC, CPSR
10001	FIQ	R7..R0, LR_fiq, SP_fiq PC, CPSR, SPSR_fiq	R7..R0, R14_fiq..R8_fiq, PC, CPSR, SPSR_fiq
10010	IRQ	R7..R0, LR_irq, SP_irq PC, CPSR, SPSR_irq	R12..R0, R14_irq..R13_irq, PC, CPSR, SPSR_irq
10011	Supervisor	R7..R0, LR_svc, SP_svc, PC, CPSR, SPSR_svc	R12..R0, R14_svc..R13_svc, PC, CPSR, SPSR_svc
10111	Abort	R7..R0, LR_abt, SP_abt, PC, CPSR, SPSR_abt	R12..R0, R14_abt..R13_abt, PC, CPSR, SPSR_abt
11011	Undefined	R7..R0 LR_und, SP_und, PC, CPSR, SPSR_und	R12..R0, R14_und..R13_und, PC, CPSR
11111	System	R7..R0, LR, SP PC, CPSR	R14..R0, PC, CPSR

Table 3-1: PSR mode bit values

Exception Handler

- On completion, the exception handler:
  - Restores any saved registers
  - Copies the SPSR back to the CPSR
  - Moves the LR, minus an offset where appropriate, to the PC
- How do we do (2) and (3)?
  - If we restore CPSR first, banked LR no longer accessible
  - If we restore PC first, execution returns to the instruction before exception
  - Entry/exit actions slides earlier give a special form of the instruction that does both (2) and (3)

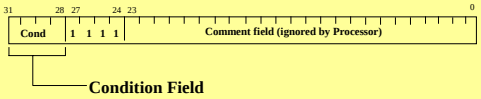
User to FIQ mode



FIQ vs IRQ latency

- FIQ
  - Fast interrupt that has a latency of 12 cycles
  - Note that it is the last entry in the exception vector table: the interrupt handler can be directly placed at 0x1c (or a jump can be made as for the other entries)
- IRQ
  - Latency of 25 cycles

Recall software interrupt

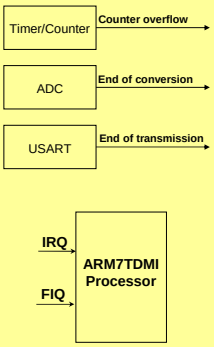


Condition Field

- In effect, a SWI is a user-defined instruction.
- It causes an exception trap to the SWI hardware vector (thus causing a change to supervisor mode, plus the associated state saving), thus causing the SWI exception handler to be called.
- The handler can then examine the comment field of the instruction to decide what operation has been requested.
- By making use of the SWI mechansim, an operating system can implement a set of privileged operations which applications running in user mode can request.
- See Exception Handling Module for further details.

Hardware Interrupts

- Internal: originate from on-chip peripherals or software
- ARM has fast and normal interrupt inputs



Source: http://www.at91.com/selftraining/ppt%20files/ARM7TDMI-based/AT91%20Interrupt%20Handling.ppt

Model Summary

- Our interrupt model is that we receive either an FIQ or IRQ and the processor saves the PC & CPSR to the appropriate LR and SPSR and jumps to the exception vector
- The ISR processes the interrupt
  - Clear the interrupt source
  - Do some work e.g. update counters, save data etc
  - Return from the interrupt

Using Exceptions

Polling

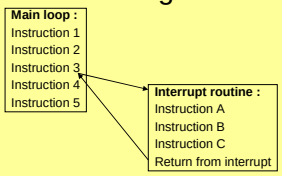
- Up till now, we have controlled I/O devices by polling

```
while (device_addr & 0x50)
        /* do nothing */ ;
send_data();
```

- Polling does not allow for processor to do other things while it tests the device's status register
  - Could be doing other things

Interrupts

- More efficient scheme: jump to an interrupt service routine (otherwise known as an exception) when a device changes status
- Other times can be doing other things



Source: http://www.at91.com/selftraining/ppt%20files/ARM7TDMI-based/AT91%20Interrupt%20Handling.ppt

IRQ_Handler

- An interrupt service routine (ISR) or IRQ_Handler must
  - Save registers that it will destroy
  - Do its work
  - Clear interrupt source
    - i.e. notify hardware that the interrupt has been serviced
  - Return from interrupt
- ISRs are very easy to get wrong and symptoms are normally non-deterministic!

MRS/MSR

- MRS and MSR read and write the CPSR & SPSRs
- Commonly used to change processor mode and to enable/disable interrupts, when the core is in a privileged mode (i.e. not User mode).
- Status registers are split into four 8-bit fields that can be individually written:
  Control (c) bits 0-7 5 processor mode bits, I & F interrupt disable bits, and the T bit on ARMv4T.
  Extension (x) bits 8-15 Reserved for future use (unused in Arch 3, 4 & 4T)
  Status (s) bits 16-23 Reserved for future use (unused in Arch 3, 4 & 4T)
  Flags (f) bits 24-31 NZCV flags (28-31) and 4 bits (24-27) reserved for future use
- MRS(cond) Rd, CPSR
  MRS(cond) Rd, SPSR
  MSR(cond) CPSR_fields, Rm
  MSR(cond) SPSR_fields, Rm
  MSR(cond) CPSR_fields, #immediate
  MSR(cond) SPSR_fields, #immediate

where 'fields' can be any combination of "cxsf".

- e.g. MSR CPSR_c, #0x1F ; go to System mode, IRQ & FIQ enabled

Enable Interrupts

- Enable interrupts by clearing FIQ and IRQ bits in the CPSR. To enable interrupts and enter supervisor mode
  - MSR cpsr_c,#0x13;
- Setting up the SP for various modes can be done in several ways. Set SP for IRQ mode (SP_irq). After setting the SP_irq, the mode is switched back to Supervisor Mode.

```
MRS R0, CPSR      ; current mode in R0
BIC R1, R0,#0x1F  ; clear mode bits
ORR R1, R1,#0x12  ; make it IRQ mode
MSR cpsr_c, R1    ; change to IRQ mode
LDR SP, =0x4.....  ; set the stack pointer
MSR cpsr_c, R0    ; back to previous mode (supervisor)
```


VIC

- ARM only allows two interrupt sources (FIQ and IRQ)
  - Most applications require more
- The vectored interrupt controller (VIC) is a peripheral that expands this to
  - 32 interrupt request inputs
  - 16 vectored IRQ interrupts
  - 16 priority levels dynamically assigned to interrupt requests
  - Software interrupt generation
  - Described in Ch 5 of http://www.nxp.com/acrobat_download/usermanuals/UM10120_1.pdf

VIC Categories

- Interrupt sources can be assigned one of 3 categories:
  - FIQ (best possible latency if only one source for this)
  - vectored IRQ only 16 of the 32 can be of this category. An ISR address can be stored in the VIC
  - non-vectored IRQ
- The above categories are in decreasing priority

Internal Peripherals

Table 55: Connection of interrupt sources to the Vectored Interrupt Controller (VIC)

Block	Flag(s)	VIC Channel # and Hex Mask	
WDT	Watchdog Interrupt (WDINT)	0	0x0000 0001
-	Reserved for Software Interrupts only	1	0x0000 0002
ARM Core	Embedded ICE, DbgCommRx	2	0x0000 0004
ARM Core	Embedded ICE, DbgCommTx	3	0x0000 0008
TIMER0	Match 0 - 3 (MR0, MR1, MR2, MR3) Capture 0 - 3 (CR0, CR1, CR2, CR3)	4	0x0000 0010
TIMER1	Match 0 - 3 (MR0, MR1, MR2, MR3) Capture 0 - 3 (CR0, CR1, CR2, CR3)	5	0x0000 0020
UART0	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI)	6	0x0000 0040
UART1	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) Modem Status Interrupt (MSI) [2]	7	0x0000 0080
PWM0	Match 0 - 6 (MR0, MR1, MR2, MR3, MR4, MR5, MR6)	8	0x0000 0100
I²C0	SI (state change)	9	0x0000 0200
SPI0	SPI Interrupt Flag (SPIF) Mode Fault (MODF)	10	0x0000 0400

VIC Registers

1 means true

0=IRQ 1 = FIQ

Table 33: VIC register map

Name	Description	Access	Reset value[1]	Address
VICIRQStatus	IRQ Status Register. This register reads out the state of those interrupt requests that are enabled and classified as IRQ.	RO	0	0xFFFF F000
VICFIQStatus	FIQ Status Requests. This register reads out the state of those interrupt requests that are enabled and classified as FIQ.	RO	0	0xFFFF F004
VICRawIntr	Raw Interrupt Status Register. This register reads out the state of the 32 interrupt requests / software interrupts, regardless of enabling or classification.	RO	0	0xFFFF F008
VICIntSelect	Interrupt Select Register. This register classifies each of the 32 interrupt requests as contributing to FIQ or IRQ.	R/W	0	0xFFFF F00C
VICIntEnable	Interrupt Enable Register. This register controls which of the 32 interrupt requests and software interrupts are enabled to contribute to FIQ or IRQ.	R/W	0	0xFFFF F010
VICIntEnClr	Interrupt Enable Clear Register. This register allows software to clear one or more bits in the Interrupt Enable register.	WO	0	0xFFFF F014
VICSoftInt	Software Interrupt Register. The contents of this register are ORed with the 32 interrupt requests from various peripheral functions.	R/W	0	0xFFFF F018
VICSoftIntClear	Software Interrupt Clear Register. This register allows software to clear one or more bits in the Software Interrupt register.	WO	0	0xFFFF F01C
VICProtection	Protection enable register. This register allows limiting	R/W	0	0xFFFF F020

VIC Registers

VICProtection	Protection enable register. This register allows limiting access to the VIC registers by software running in privileged mode.	R/W	0	0xFFFF F020
VICVectAddr	Vector Address Register. When an IRQ interrupt occurs, the IRQ service routine can read this register and jump to the value read.	R/W	0	0xFFFF F030
VICDefVectAddr	Default Vector Address Register. This register holds the address of the Interrupt Service routine (ISR) for non-vectored IRQs.	R/W	0	0xFFFF F034
VICVectAddr0	Vector address 0 register. Vector Address Registers 0-15 hold the addresses of the Interrupt Service routines (ISRs) for the 16 vectored IRQ slots.	R/W	0	0xFFFF F100
VICVectAddr1	Vector address 1 register.	R/W	0	0xFFFF F104
VICVectAddr2	Vector address 2 register.	R/W	0	0xFFFF F108
VICVectAddr3	Vector address 3 register.	R/W	0	0xFFFF F10C
VICVectAddr4	Vector address 4 register.	R/W	0	0xFFFF F110
VICVectAddr5	Vector address 5 register.	R/W	0	0xFFFF F114
VICVectAddr6	Vector address 6 register.	R/W	0	0xFFFF F118
VICVectAddr7	Vector address 7 register.	R/W	0	0xFFFF F11C
VICVectAddr8	Vector address 8 register.	R/W	0	0xFFFF F120
VICVectAddr9	Vector address 9 register.	R/W	0	0xFFFF F124
VICVectAddr10	Vector address 10 register.	R/W	0	0xFFFF F128
VICVectAddr11	Vector address 11 register.	R/W	0	0xFFFF F12C

VIC Registers

VICVectAddr12	Vector address 12 register.	R/W	0	0xFFFF F130
VICVectAddr13	Vector address 13 register.	R/W	0	0xFFFF F134
VICVectAddr14	Vector address 14 register.	R/W	0	0xFFFF F138
VICVectAddr15	Vector address 15 register.	R/W	0	0xFFFF F13C
VICVectCntl0	Vector control 0 register. Vector Control Registers 0-15 each control one of the 16 vectored IRQ slots. Slot 0 has the highest priority and slot 15 the lowest.	R/W	0	0xFFFF F200
VICVectCntl1	Vector control 1 register.	R/W	0	0xFFFF F204
VICVectCntl2	Vector control 2 register.	R/W	0	0xFFFF F208
VICVectCntl3	Vector control 3 register.	R/W	0	0xFFFF F20C
VICVectCntl4	Vector control 4 register.	R/W	0	0xFFFF F210
VICVectCntl5	Vector control 5 register.	R/W	0	0xFFFF F214
VICVectCntl6	Vector control 6 register.	R/W	0	0xFFFF F218
VICVectCntl7	Vector control 7 register.	R/W	0	0xFFFF F21C
VICVectCntl8	Vector control 8 register.	R/W	0	0xFFFF F220
VICVectCntl9	Vector control 9 register.	R/W	0	0xFFFF F224
VICVectCntl10	Vector control 10 register.	R/W	0	0xFFFF F228
VICVectCntl11	Vector control 11 register.	R/W	0	0xFFFF F22C
VICVectCntl12	Vector control 12 register.	R/W	0	0xFFFF F230
VICVectCntl13	Vector control 13 register.	R/W	0	0xFFFF F234
VICVectCntl14	Vector control 14 register.	R/W	0	0xFFFF F238
VICVectCntl15	Vector control 15 register.	R/W	0	0xFFFF F23C

VICSoftInt

Bit	23	22	21	20	19	18	17	16
Symbol	-	-	AD1	BOD	I2C1	AD0	EINT3	EINT2
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Bit	15	14	13	12	11	10	9	8
Symbol	EINT1	EINT0	RTC	PLL	SPI1/SSP	SPI0	I2C0	PWM0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Bit	7	6	5	4	3	2	1	0
Symbol	UART1	UART0	TIMER1	TIMER0	ARMCore1	ARMCore0	-	WDT
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Table 35: Software Interrupt register (VICSoftInt - address 0xFFFF F018) bit description

Bit	Symbol	Value	Description	Reset value
31:0	See VICSoftInt bit allocation table.	0	Do not force the interrupt request with this bit number. Writing zeroes to bits in VICSoftInt has no effect, see VICSoftIntClear (Section 5.4.2).	0

Ex: Setting type and Enabling Interrupts

- Program UART0 and SPI0 as vectored IRQs (UART0 being on the higher level than SPI0), while UART1 and I2C are generating non-vectored IRQs
- First set which are IRQ/FIQ

; SPI0, I2C, UART1 and UART0 are IRQ => bit10, bit9, bit7 and bit6=0
VICIntSelect = 0x0000 0440

- Enable them

; SPI0, I2C, UART1 and UART0 are enabled interrupts => bit10, bit9, bit 7 and bit6=1
VICIntEnable = 0x0000 06C0

Ex: Setting Vectors

; holds address at what routine for servicing non-vectored IRQs (i.e. UART1 and I2C) starts

VICDefVectAddr = 0x...

VICVectAddr0 = 0x... ; holds address where UART0 IRQ service routine starts

VICVectAddr1 = 0x... ; holds address where SPI0 IRQ service routine starts

Vector Control Reg

5.4.9 Vector Control registers 0-15 (VICVectCntl0-15 - 0xFFFF F200-23C)

These are a read/write accessible registers. Each of these registers controls one of the 16 vectored IRQ slots. Slot 0 has the highest priority and slot 15 the lowest. Note that disabling a vectored IRQ slot in one of the VICVectCntl registers does not disable the interrupt itself, the interrupt is simply changed to the non-vectored form.

Table 50: Vector Control registers 0-15 (VICVectCntl0-15 - 0xFFFF F200-23C) bit description

Bit	Symbol	Description	Reset value
4:0	int_request/ sw_int_assig	The number of the interrupt request or software interrupt assigned to this vectored IRQ slot. As a matter of good programming practice, software should not assign the same interrupt number to more than one enabled vectored IRQ slot. But if this does occur, the lower-numbered slot will be used when the interrupt request or software interrupt is enabled, classified as IRQ, and asserted.	0
5	IRQslot_en	When 1, this vectored IRQ slot is enabled, and can produce a unique ISR address when its assigned interrupt request or software interrupt is enabled, classified as IRQ, and asserted.	0
31:6	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Ex: Setting Priority

- VICVectCntlx assigns

; interrupt source with index 6 (UART0) is enabled as

; the one with priority 0 (the highest)

VICVectCntl0 = 0x0000 0026

; interrupt source with index 10 (SPI0) is enabled as the one with priority 1

VICVectCntl1 = 0x0000 002A

Ex: Execution vector

- After the programming, any IRQ on UART0, SPI0, UART1 or I2C will cause jump to IRQ vector (0x18)
  - Could put **LDR pc, [pc,#-0xFF0]** instruction there
  - This instruction loads PC with the address that is present in VICVectAddr (**0xFFFFF030)** register!
  - Because pc=0x18+8=0x20 and -0xff0= 0xFFFFF00F+1=0xFFFFF010)
- This single instruction handles all 32 possible interrupt sources!

Updating VIC Hardware

- Interrupts are prioritized by the VIC. Need to allow it to update at the end of the ISR

5.4.12 Vector Address register (VICVectAddr - 0xFFFF F030)

This is a read/write accessible register. When an IRQ interrupt occurs, the IRQ service routine can read this register and jump to the value read.

Table 53: Vector Address register (VICVectAddr - address 0xFFFF F030) bit description

Bit	Symbol	Description	Reset value
31:0	IRQ_vector	If any of the interrupt requests or software interrupts that are assigned to a vectored IRQ slot is (are) enabled, classified as IRQ, and asserted, reading from this register returns the address in the Vector Address Register for the highest-priority such slot (lowest-numbered) such slot. Otherwise it returns the address in the Default Vector Address Register. Writing to this register does not set the value for future reads from it. Rather, this register should be written near the end of an ISR, to update the priority hardware.	0x0000 0000

Interrupt Example

- http://www.nxp.com/acrobat_download/applicationnotes/AN10254_2.pdf
- Timer1 is configured to trigger an IRQ interrupt and the code runs from Flash

Interrupt vector table

This code should be linked to 0x0. Here the interrupt vectors are provided for the ARM core.

```
; ------------------------------------------------------------
; Assembler Directives
; ------------------------------------------------------------
AREA IVT, CODE ; New Code section
CODE32 ; ARM code
IMPORT start ; start symbol not
; defined in this
; section
Entry ; Defines entry point
; ------------------------------------------------------------
LDR PC, =start
LDR PC, Undefined_Addr
LDR PC, SWI_Addr
LDR PC, Prefetch_Addr
LDR PC, Abort_Addr
; At 0x14 the user should insert a signature
    (checksum).
DCD .
LDR PC, [PC, #-0xFF0]
LDR PC, FIQ_Addr

Undefined_Addr DCD Undefined_Handler
SWI_Addr DCD SWI_Handler
Prefetch_Addr DCD Prefetch_Handler
Abort_Addr DCD Abort_Handler
FIQ_Addr DCD FIQ_Handler

; ------------------------------------------------------------
; Exception Handlers
; ------------------------------------------------------------
; The following dummy handlers do not do
    anything useful in
; this example.They are set up here for
    completeness.
Undefined_Handler
B Undefined_Handler
SWI_Handler
B SWI_Handler
Prefetch_Handler
B Prefetch_Handler
Abort_Handler
B Abort_Handler
FIQ_Handler
B FIQ_Handler
END
```


Startup code

```
; ------------------------------------------------------------
; Assembler Directives
; ------------------------------------------------------------
AREA asm_code, CODE ; New Code section
CODE32 ; ARM code
IMPORT __main ; main not defined
; in this section
EXPORT start ; global symbol
; referenced in
; ivt.s
; ------------------------------------------------------------
start
; Enable interrupts
MSR cpsr_c,#0x13

; Set SP for Supervisor mode. Depending upon
; the stack the application needs this value
; needs to be set.
LDR SP,=0x4 ..
; Setting up SP for IRQ mode. Change mode to
; IRQ before setting SP_irq and then
; move back again to Supervisor mode
MRS R0, CPSR
BIC R1, R0,#0x1F
ORR R1, R1,#0x12
MSR cpsr_c, R1
LDR SP, =0x4..
MSR cpsr_c, R0
; Jump to C code
LDR lr, =__main
MOV pc, lr
END
```


Initialize (in C)

```
/* Function declarations */
__irq void IRQHandler(void);
void feed(void);
void Initialize(void);

#include"LPC2___.h"

int main()
{
    /* Initialize the system */
    Initialize();
    /* Start timer */
    T1_TCR=0x1;
    while(1)
    {
    }
}

/*********************************************************
    ****
Initialize
*********************************************************
    ***/
void Initialize()
{
    /* Setting Multiplier and divider values */
    PLLCFG=0x25;
    feed();
    /* Enabling the PLL */
    PLLCON=0x1;
    feed();
    /* Wait for the PLL to lock to set frequency */
    while(!(PLLSTAT & PLOCK)){}
    /* Connect the PLL as the clock source */
    PLLCON=0x3;
    feed();
    /* Enabling MAM and set number of clocks
    used for Flash memory fetch */
    MAMCR=0x2;
    MAMTIM=0x4;
```


```
    /*
    * Setting peripheral Clock (pclk)
    * to System Clock (cclk)
    */
    VPBDIV=0x1;
    /* Initialize GPIO */
    IODIR=0xFFFF;
    IOSET=0xFFFF;
    /* Initialize Timer 1 */
    T1_TCR=0x0; T1_TC=0x0;
    T1_PR=0x0;
    T1_PC=0x0;
    /* End user has to fill in the match value */
    T1_MR0=0x.....;
    /* Reset and interrupt on match */
    T1_MCR=0x3;
    /* Initialize VIC */
    VICINTSEL=0x0; /* Timer 1 selected as IRQ */
    VICINTEN= 0x20;/* Timer 1 interrupt enabled */
    VICCNTL0= 0x25;
    /* Address of the ISR */
    VICVADDR0=(unsigned long)IRQHandler;
}

void feed()
{
    PLLFEED=0xAA;
    PLLFEED=0x55;
}
```


Actual ISR

```
/*************************************************************
Timer 1 ISR
*************************************************************/
__irq void IRQHandler()
{
    /*
    * The ISR code. The
    * interrupt cleared in Timer1, a write must
    * be performed on the VIC Vector Address Register to
    * update VIC priority hardware. Here the user could
    * blink a few LED's or toggle some port pins as an
    * indication of being in the ISR
    */
    T1_IR=0x1;
    VICVADDR=0xff;
}
```


ISR in assembly

```
STMDB    SP!,{R0-R1}          ; save R0 and R1
; T1IR=0x1;
MOV      R0,#0x00000001
LDR      R1,[PC,#0x001C]
STR      R0,[R1]
; VICVectAddr=0xff;
MOV      R0,#0x000000FF
MOV      R1,#0x00000000
STR      R0,[R1,#-0x0FD0]
LDMIA    SP!,{R0-R1}
SUBS     PC,LR,#0x00000004   ; return from subroutine
```