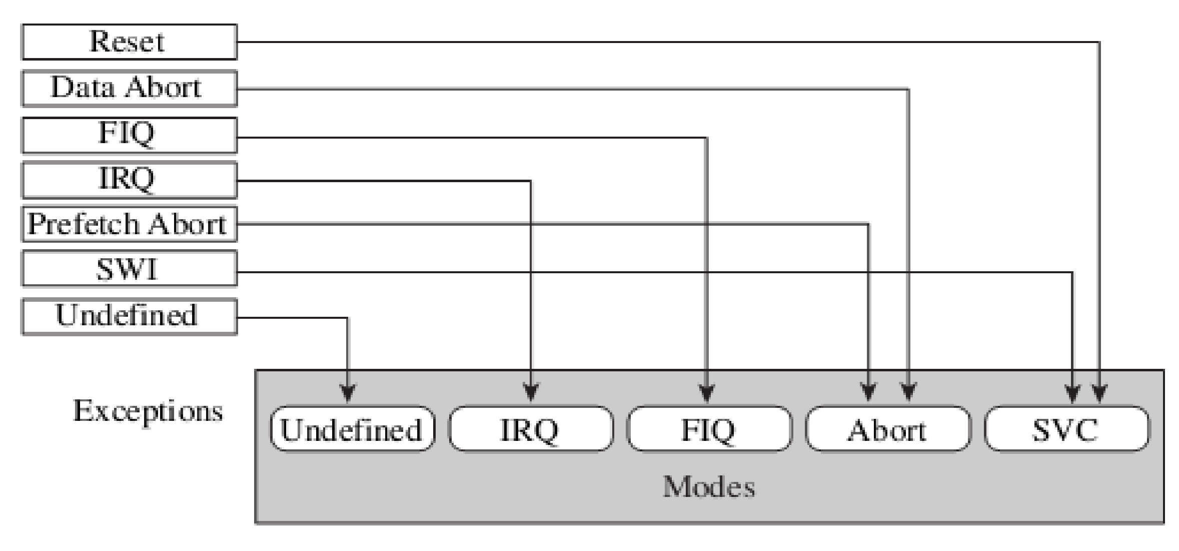
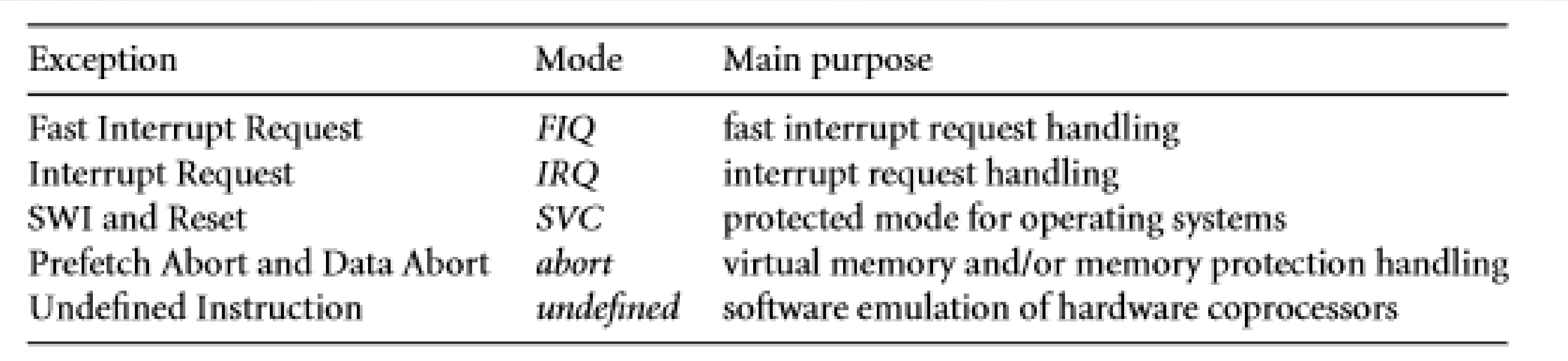
ARM Exceptions
&
ARM Interrupt Controller

Exception

Condition that needs to halt the normal sequential execution of instructions

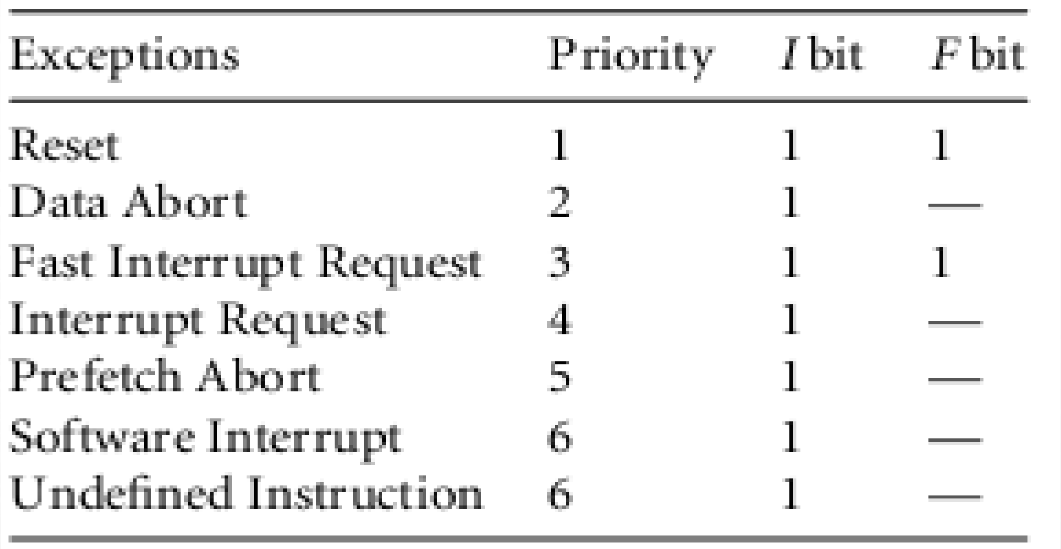
Mapping exceptions to modes





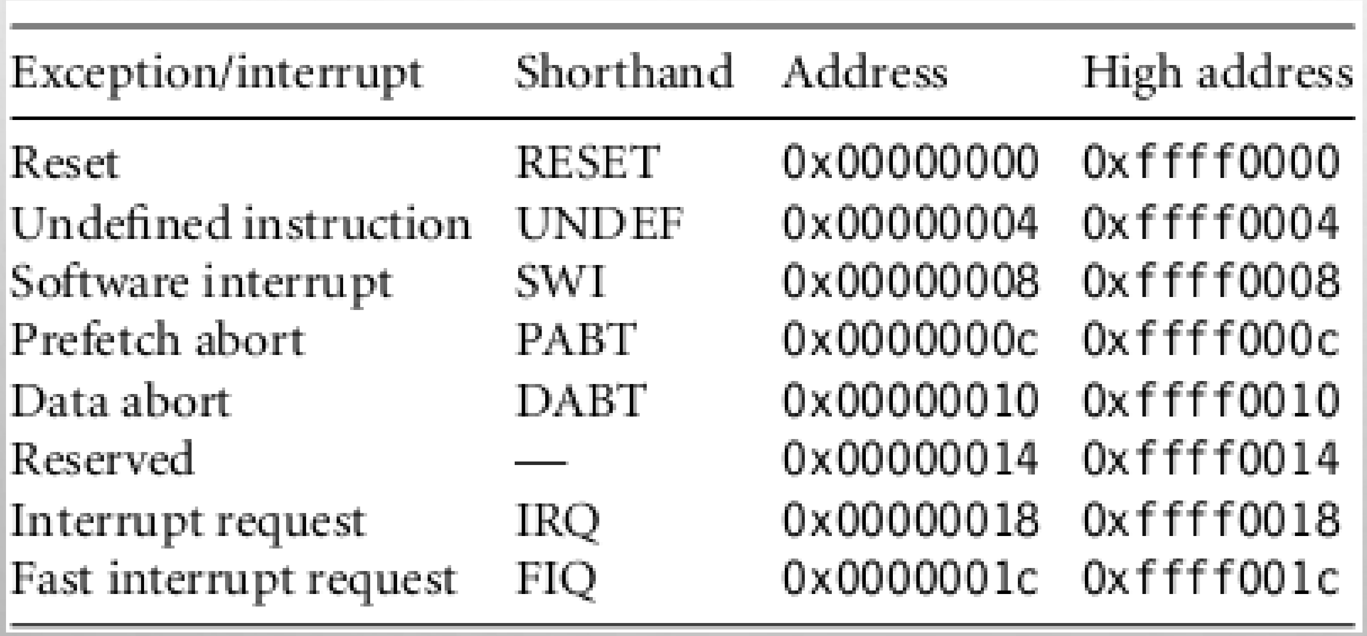
Exception	Mode	Main purpose
Fast Interrupt Request	*FIQ*	fast interrupt request handling
Interrupt Request	*IRQ*	interrupt request handling
SWI and Reset	*SVC*	protected mode for operating systems
Prefetch Abort and Data Abort	*abort*	virtual memory and/or memory protection handling
Undefined Instruction	*undefined*	software emulation of hardware coprocessors

Exception Priority:

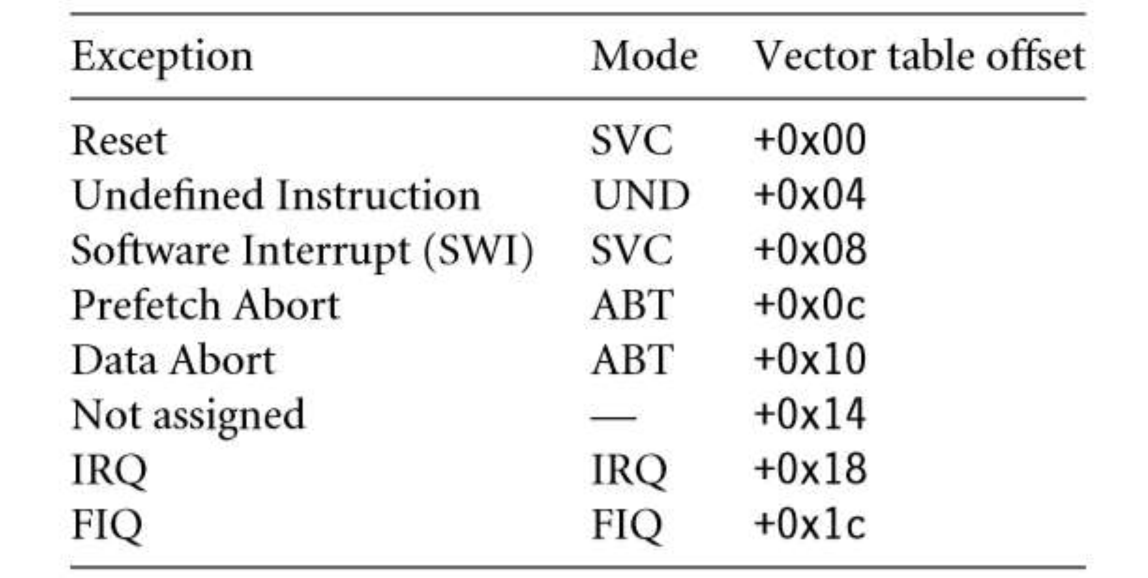


Exceptions	Priority	*I* bit	*F* bit
Reset	1	1	1
Data Abort	2	1	—
Fast Interrupt Request	3	1	1
Interrupt Request	4	1	—
Prefetch Abort	5	1	—
Software Interrupt	6	1	—
Undefined Instruction	6	1	—

Figure: ARM Vector table

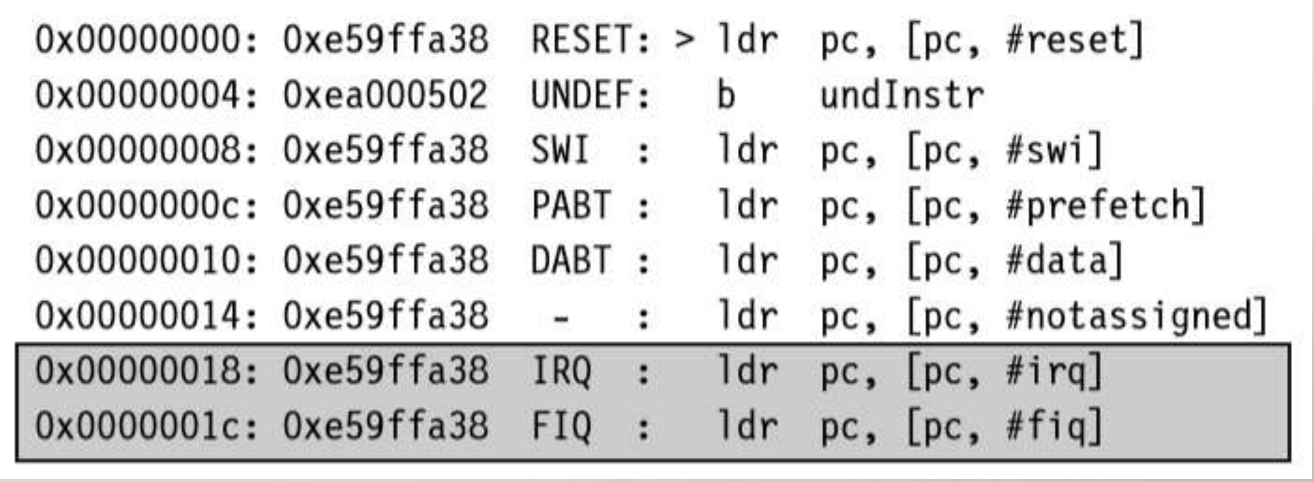


Exception/interrupt	Shorthand	Address	High address
Reset	RESET	0x00000000	0xffff0000
Undefined instruction	UNDEF	0x00000004	0xffff0004
Software interrupt	SWI	0x00000008	0xffff0008
Prefetch abort	PABT	0x0000000c	0xffff000c
Data abort	DABT	0x00000010	0xffff0010
Reserved	—	0x00000014	0xffff0014
Interrupt request	IRQ	0x00000018	0xffff0018
Fast interrupt request	FIQ	0x0000001c	0xffff001c



Exception	Mode	Vector table offset
Reset	SVC	+0x00
Undefined Instruction	UND	+0x04
Software Interrupt (SWI)	SVC	+0x08
Prefetch Abort	ABT	+0x0c
Data Abort	ABT	+0x10
Not assigned	—	+0x14
IRQ	IRQ	+0x18
FIQ	FIQ	+0x1c

- B<address> provides a branch relative from the pc.
- LDR pc, [pc, #offset]—This load register instruction loads the handler address from memory to the pc.
- LDR pc, [pc, #-0xff0]—This load register instruction loads a specific interrupt service routine address from address 0xfffff030 to the pc.
- MOV pc, #immediate—This move instruction copies an immediate value into the pc.



```
0x00000000: 0xe59ffa38  RESET: > ldr  pc, [pc, #reset]
0x00000004: 0xea000502  UNDEF:   b    undInstr
0x00000008: 0xe59ffa38  SWI  :   ldr  pc, [pc, #swi]
0x0000000c: 0xe59ffa38  PABT :   ldr  pc, [pc, #prefetch]
0x00000010: 0xe59ffa38  DABT :   ldr  pc, [pc, #data]
0x00000014: 0xe59ffa38   -   :   ldr  pc, [pc, #notassigned]
0x00000018: 0xe59ffa38  IRQ  :   ldr  pc, [pc, #irq]
0x0000001c: 0xe59ffa38  FIQ  :   ldr  pc, [pc, #fiq]
```

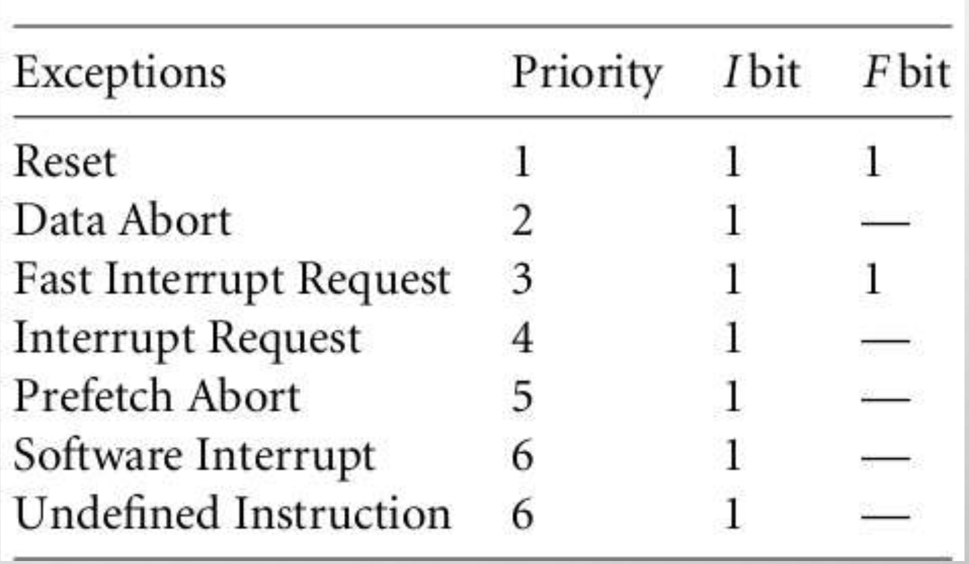
MOV pc, #immediate

RESET:

- ✓Happens when the processor powers up.
- ✓Initializes the system, sets up stacks for different processor modes.
- ✓Highest priority exception
- ✓Upon entry into the reset handler the CPSR is in SVC mode and both IRQ and FIQ bits are set to 1, masking any interrupts

DATA ABORT:

- ✓2nd highest priority.
- ✓ Happens when we try to read/write into an invalid address or access with the wrong access permission.
- ✓ Upon entry into the Data Abort Handler, IRQ's will be disabled (I-bit set 1), and FIQ will be enabled (F-bit set 0).



Exceptions	Priority	*I* bit	*F* bit
Reset	1	1	1
Data Abort	2	1	—
Fast Interrupt Request	3	1	1
Interrupt Request	4	1	—
Prefetch Abort	5	1	—
Software Interrupt	6	1	—
Undefined Instruction	6	1	—

FIQ:

- ✓Highest priority interrupt
- ✓ IRQ & FIQs are disabled till FIQ is handled.

Pre-Fetch Abort:

✓Similar to data abort, but happens on address fetch failure.

✓ On entry to the handler, IRQs are disabled, but FIQs remain enabled and can happen during a Pre-Fetch abort.

SWI:

✓A Software Interrupt (SWI) exception occurs when the SWI instruction is executed and none of the other higher-priority exceptions have been flagged.

IRQ:

✓2nd Highest priority interrupt

✓IRQ handler is entered only if there is no FIQ & Data Abort on-going.

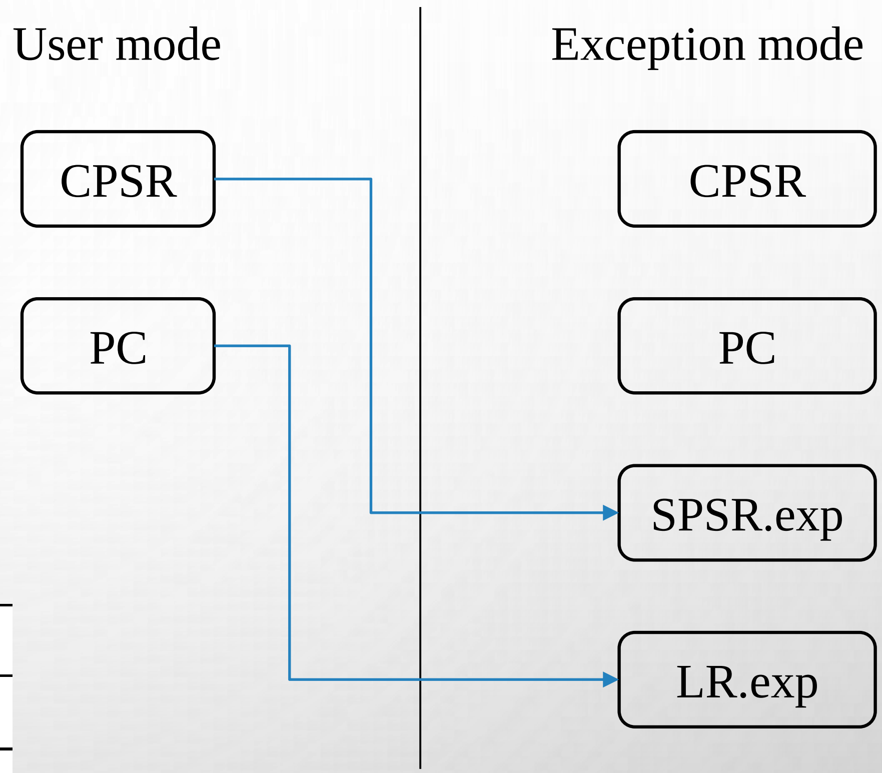
Undefined Instruction:

✓Undefined Instruction exception occurs when an instruction not in the ARM or Thumb instruction set reaches the execute stage of the pipeline and none of the other exceptions have been flagged

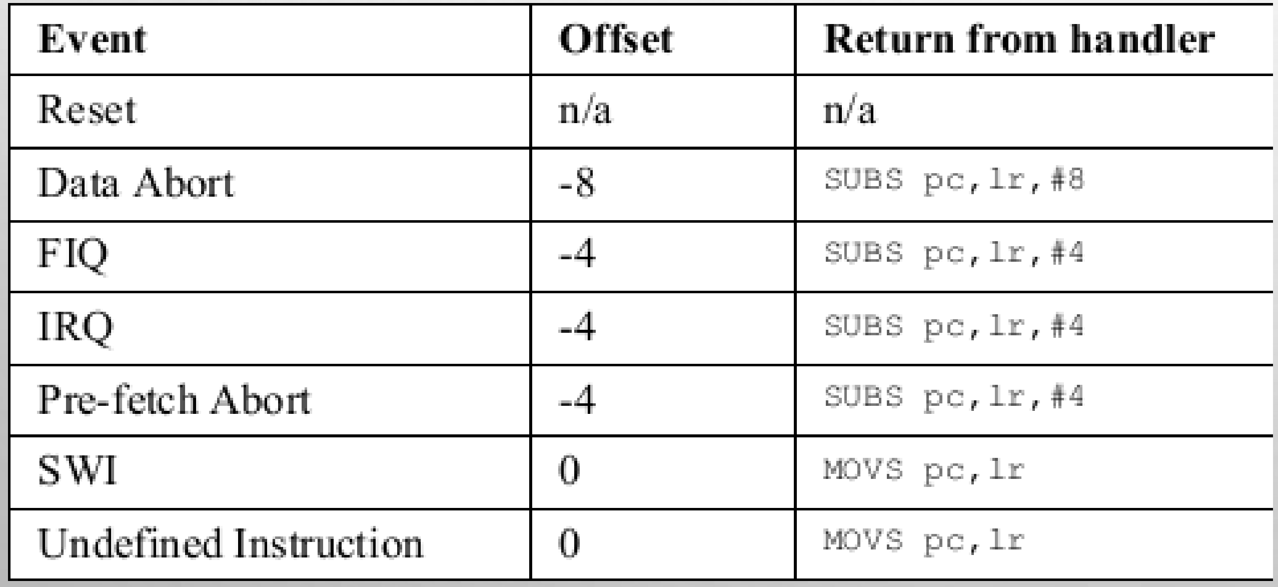
✓ Same priority as SWI as one can happen at a time. Meaning as the instruction being executed cannot both be an SWI instruction and an undefined instruction at the same time.

ARM Exception handling

- saves the cpsr to the spsr of the exception mode
- saves the pc to the lr of the exception mode
- sets the cpsr to the exception mode
- sets pc to the address of the exception handler

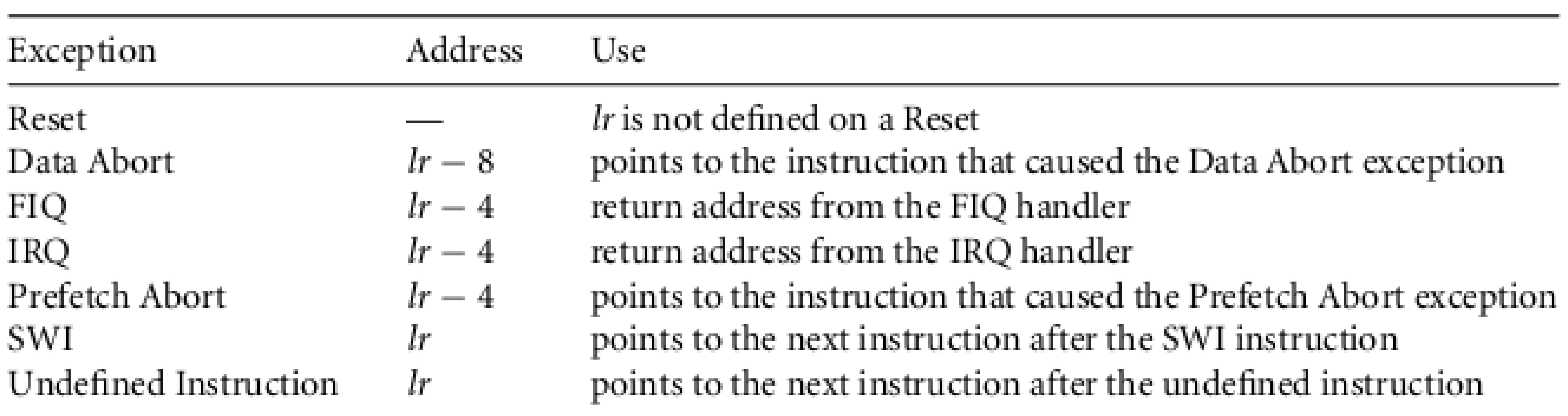


Return from Exception:



Event	Offset	Return from handler
Reset	n/a	n/a
Data Abort	-8	SUBS pc,lr,#8
FIQ	-4	SUBS pc,lr,#4
IRQ	-4	SUBS pc,lr,#4
Pre-fetch Abort	-4	SUBS pc,lr,#4
SWI	0	MOVS pc,lr
Undefined Instruction	0	MOVS pc,lr

Locating the offending instruction:



Exception	Address	Use
Reset	—	*lr* is not defined on a Reset
Data Abort	*lr* − 8	points to the instruction that caused the Data Abort exception
FIQ	*lr* − 4	return address from the FIQ handler
IRQ	*lr* − 4	return address from the IRQ handler
Prefetch Abort	*lr* − 4	points to the instruction that caused the Prefetch Abort exception
SWI	*lr*	points to the next instruction after the SWI instruction
Undefined Instruction	*lr*	points to the next instruction after the undefined instruction

This example shows that a typical method of returning from an IRQ and FIQ handler is to use a SUBS instruction:

```
handler
<handler code>
 ...
SUBS pc, r14, #4 ; pc=r14-4
```

Because there is an S at the end of the SUB instruction and the pc is the destination register, the cpsr is automatically restored from the spsr register.

This example shows another method that subtracts the offset from the link register r14 at the beginning of the handler.

```
handler
SUB r14, r14, #4 ; r14-=4
...
<handler code>
...
MOVS pc, r14 ; return
```

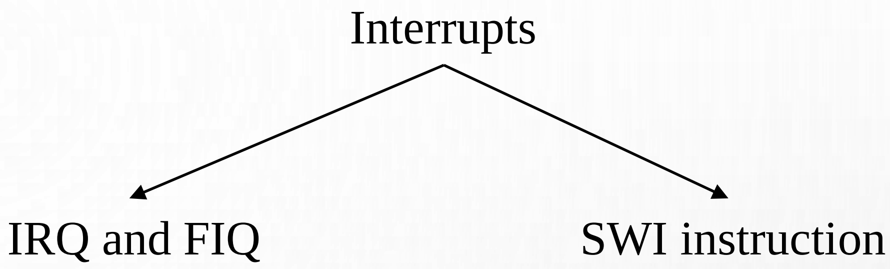
After servicing is complete, return to normal execution occurs by moving the link register r14 into the pc and restoring cpsr from the spsr.

The final example uses the interrupt stack to store the link register. This method first subtracts an offset from the link register and then stores it onto the interrupt stack.

```
handler
SUB r14, r14, #4 ; r14-=4
STMFD r13!,{r0-r3, r14} ; store context
...
<handler code>
...
LDMFD r13!,{r0-r3, pc}ˆ ; return
```

To return to normal execution, the LDM instruction is used to load the pc. The ˆ symbol in the instruction forces the cpsr to be restored from the spsr.

ARM Interrupt handling



Interrupts

IRQ and FIQ

SWI instruction

- Software Interrupts are normally reserved to call privileged operating system routines.

  Eg: Change a program running in user mode to a privileged mode.

- Interrupt Requests are normally assigned for general-purpose interrupts.

  IRQ exception has a lower priority and higher interrupt latency

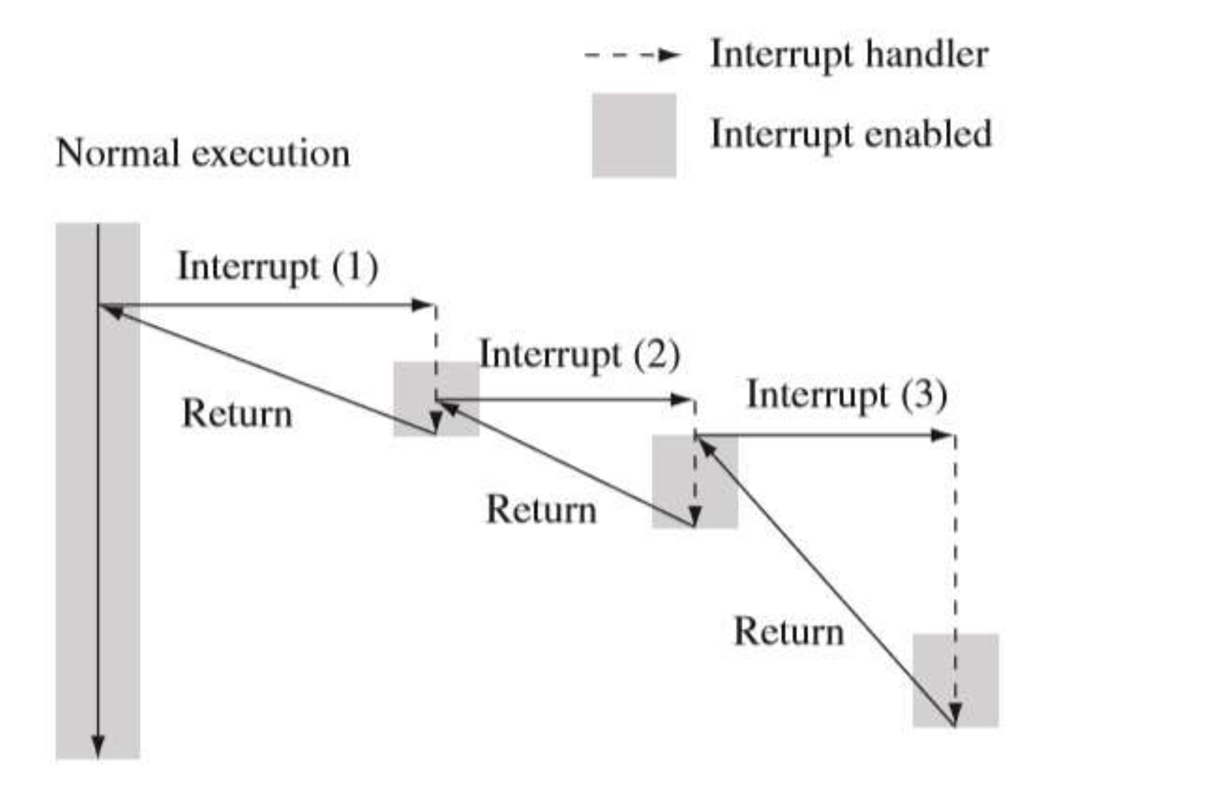
- Fast Interrupt Requests are normally reserved for a single interrupt source that requires a fast response time

Interrupt latency

The interval of time from an external interrupt request signal being raised to the first fetch of an instruction of a specific interrupt service routine (ISR).

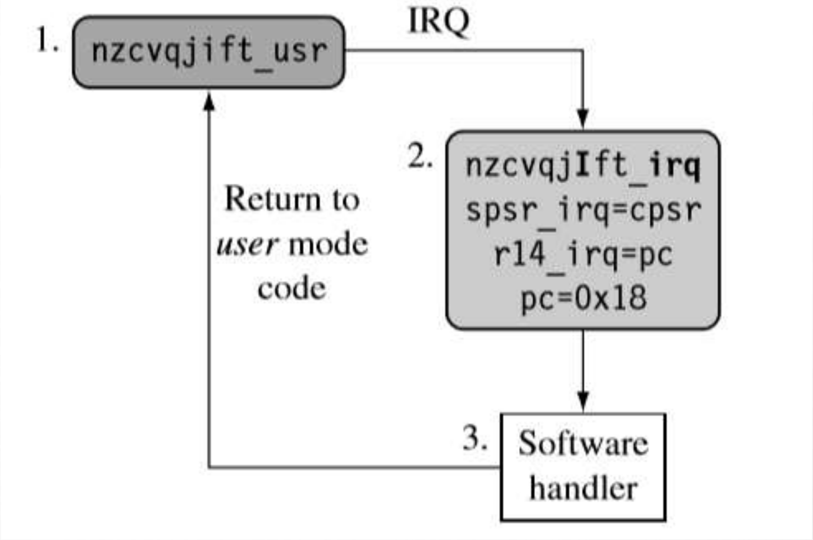
Two main methods to minimize interrupt latency

- **use a nested interrupt handler** - achieved by re-enabling the interrupts as soon as the interrupt source has been serviced but before the interrupt handling is complete.
- **Prioritization -** ignore interrupts of the same or lower priority than the interrupt you are handling, so only a higher-priority task can interrupt your handler.



Interrupt request IRQ

- When an IRQ occurs, the processor moves into state 2.
- Sets the IRQ =1, disabling any further IRQ exceptions.
- FIQ has a higher priority and therefore doesn't get disabled when a low-priority IRQ exception is raised.
- The cpsr processor mode changes to IRQ mode.
- spsr_irq = cpsr.
- PC = r14_irq.
- pc = IRQ entry +0x18 in the vector table.
- In state 3 the software handler takes over
- Upon completion, the processor mode reverts back to the original user mode code in state 1.



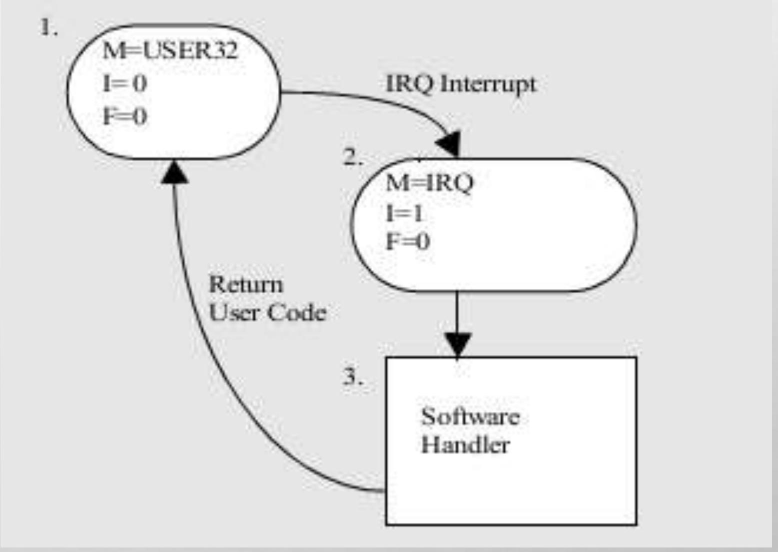
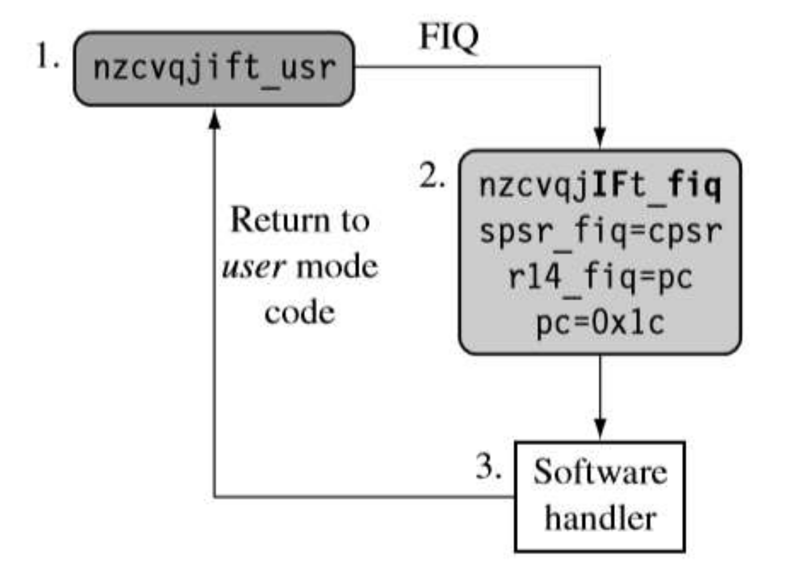
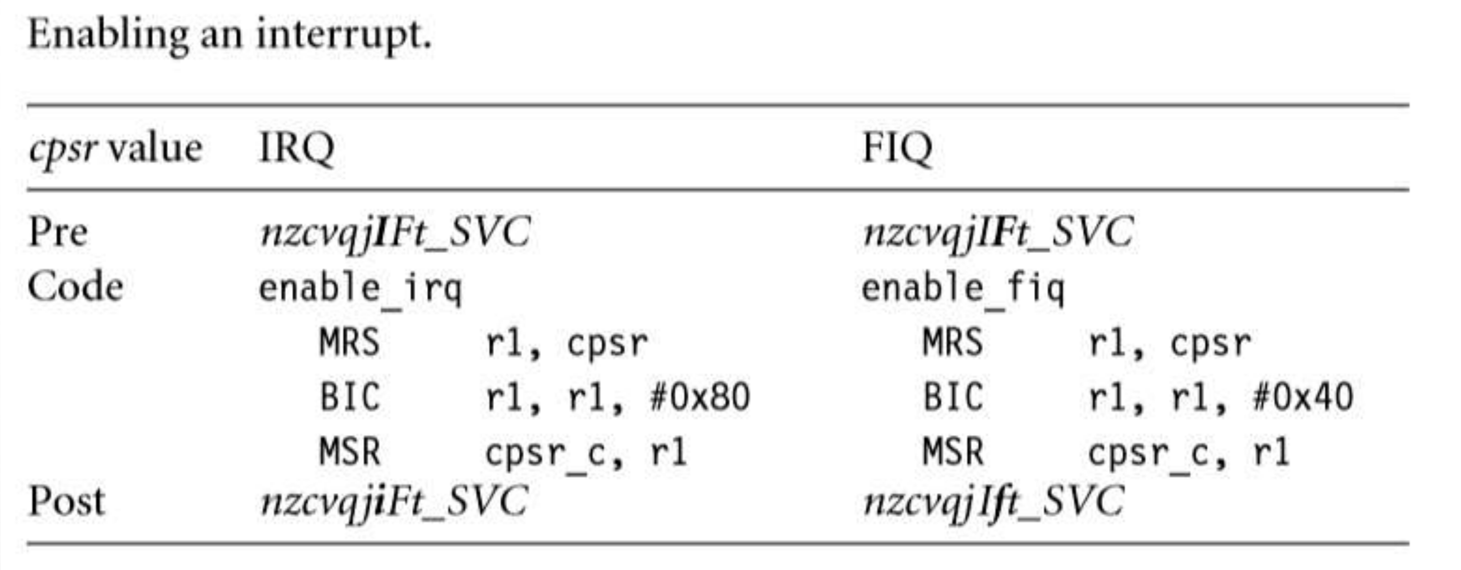


Figure: State-Machine for an IRQ

Interrupt request FIQ



- When an FIQ occurs, the processor moves into state 2.
- Sets the IRQ =1 and FIQ =1, disabling any further IRQ and FIQ exceptions.
- The cpsr processor mode changes to FIQ mode.
- No need to save registers r8 to r12
- spsr_fiq = cpsr.
- r14_fiq = pc.
- pc = FIQ entry +0x1C in the vector table.
- In state 3 the software handler takes over
- Upon completion, the processor mode reverts back to the original user mode code in state 1.
- FIQ is ideal for servicing a single-source, high-priority, low-latency interrupt.

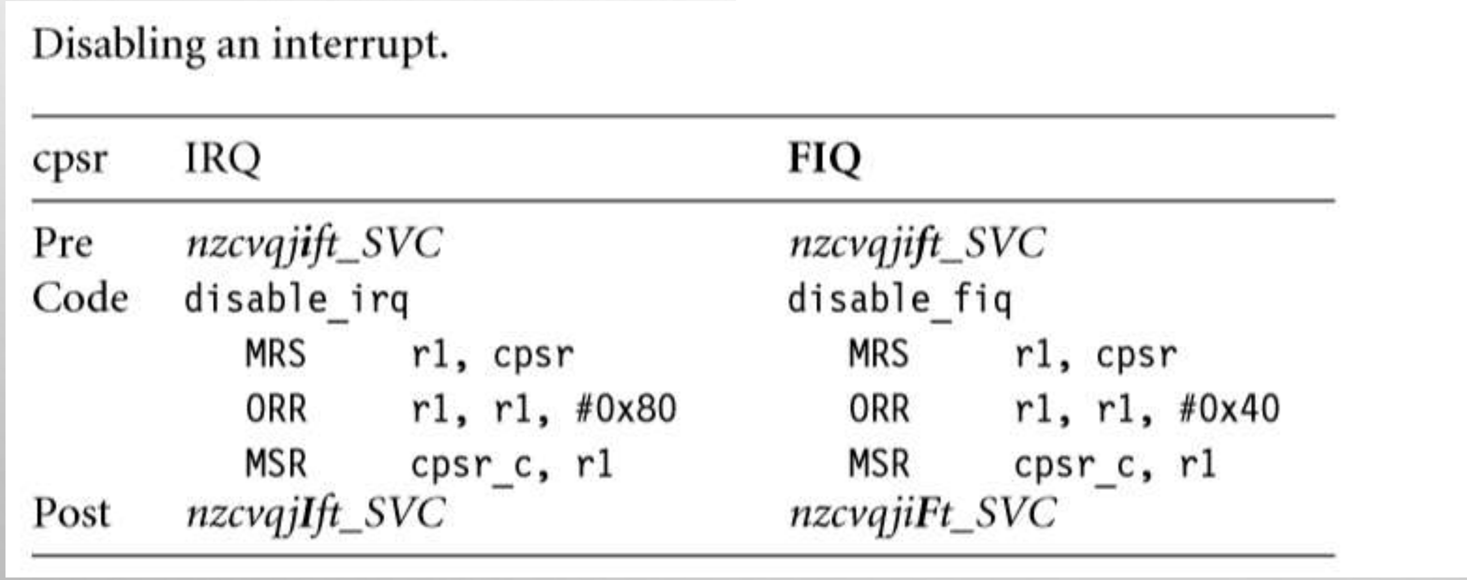


Enabling an interrupt.

cpsr value	IRQ	FIQ
Pre	*nzcvqj**I**Ft_SVC*	*nzcvqjI**F**t_SVC*

Code	`enable_irq`
	`MRS r1, cpsr`
	`BIC r1, r1, #0x80`
	`MSR cpsr_c, r1`
		`enable_fiq`
		`MRS r1, cpsr`
		`BIC r1, r1, #0x40`
		`MSR cpsr_c, r1`

Post	*nzcvqj**i**Ft_SVC*	*nzcvqjI**f**t_SVC*



Disabling an interrupt.

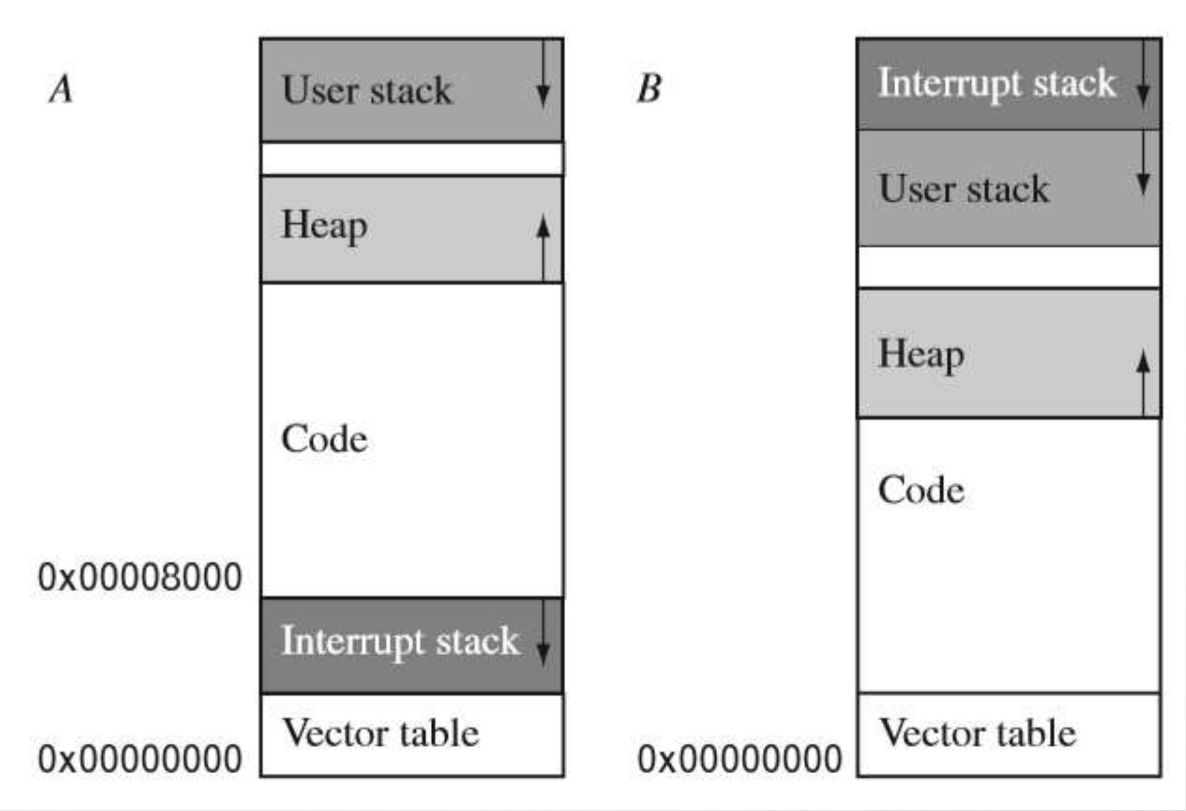
cpsr	IRQ	**FIQ**
Pre	*nzcvqj**i**ft_SVC*	*nzcvqji**f**t_SVC*

Code	`disable_irq`
	`MRS r1, cpsr`
	`ORR r1, r1, #0x80`
	`MSR cpsr_c, r1`
		`disable_fiq`
		`MRS r1, cpsr`
		`ORR r1, r1, #0x40`
		`MSR cpsr_c, r1`

Post	*nzcvqj**I**ft_SVC*	*nzcvqji**F**t_SVC*

The immediate value on the data processing BIC or ORR instruction has to be changed to 0xc0 to enable or disable both interrupts.

Stack memory layout



The main advantage of layout B over A is that B does not corrupt the vector table when a stack overflow occurs, and so the system has a chance to correct itself when an overflow has been identified.

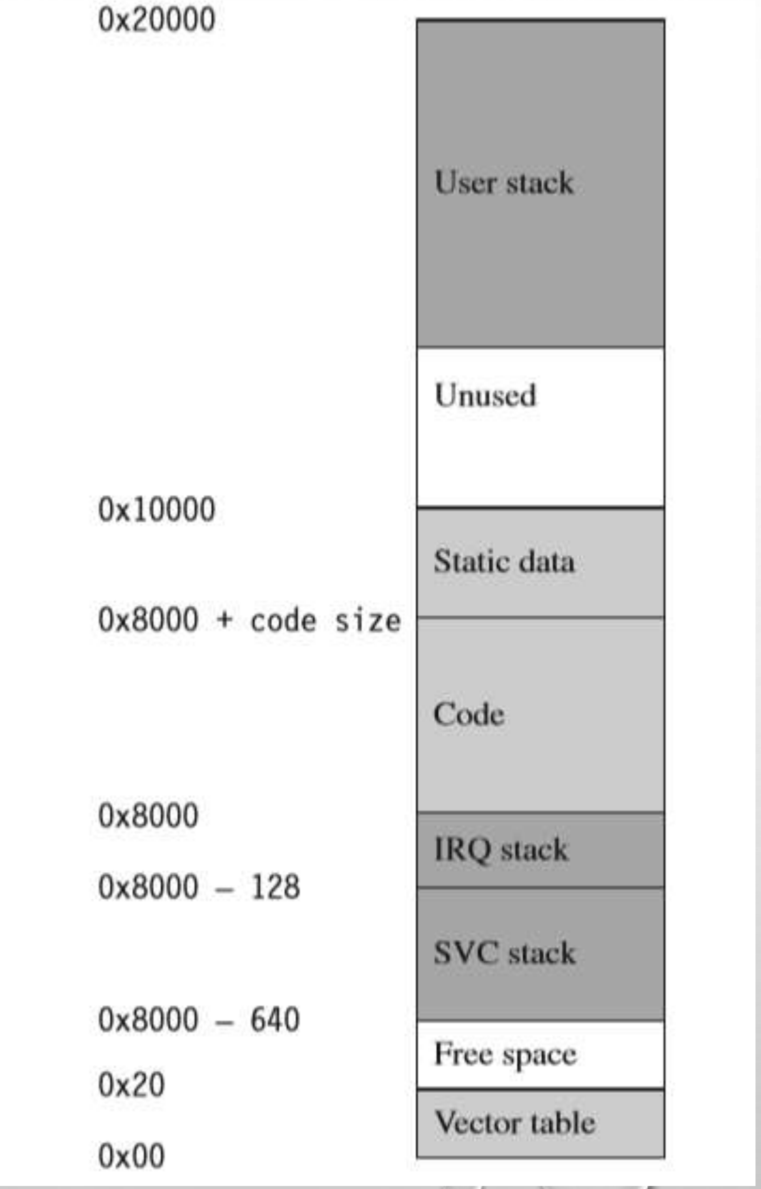
```
NoInt EQU 0xc0 ; Disable interrupt
```

Supervisor mode stack

```
LDR r13, SVC_NewStack ; r13_svc
...
SVC_NewStack
DCD SVC_Stack
```

IRQ mode stack

```
MOV r2, #NoInt|IRQ32md
MSR cpsr_c, r2
LDR r13, IRQ_NewStack ; r13_irq
...
IRQ_NewStack
DCD IRQ_Stack
```



User mode stack

```
MOV r2, #Sys32md
MSR cpsr_c, r2
LDR r13, USR_NewStack ; r13_usr
...
USR_NewStack
DCD USR_Stack
```

Non-nested Interrupt Handler

- The interrupts are disabled until control is returned back to the interrupted task or process.
- Can service only a single interrupt at a time.

1. Disable interrupt/s—When the IRQ exception is raised, the ARM processor will disable further IRQ exceptions from occurring.

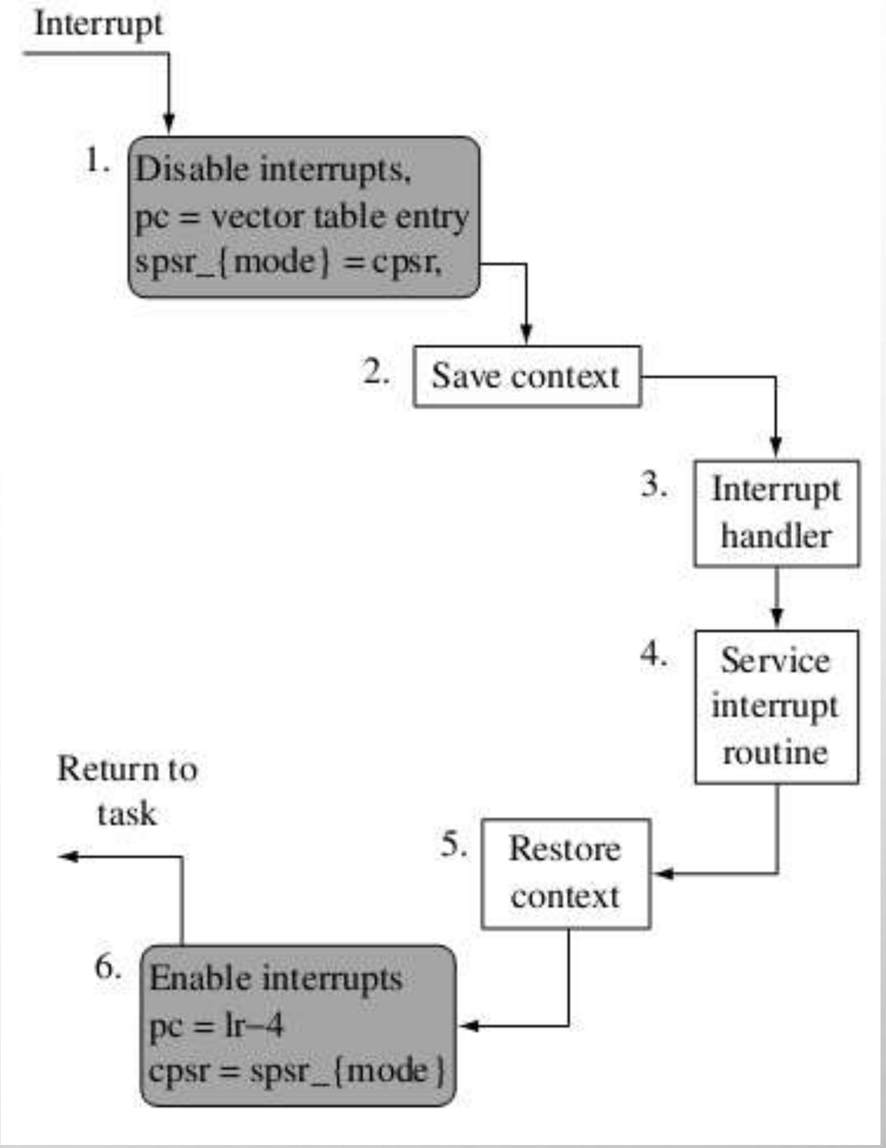
2. Save context—On entry the handler code saves a subset of the current processor mode non banked registers.

3. Interrupt handler—The handler then identifies the external interrupt source and executes the appropriate interrupt service routine (ISR).

4. Interrupt service routine—The ISR services the external interrupt source and resets the interrupt.

5. Restore context—The ISR returns back to the interrupt handler, which restores the context.

6. Enable interrupts— pc = lr−4 cpsr = spsr_{mode}



What happens when an exception happens?

1. Store the CPSR to the SPSR of the exception mode.

2. PC is stored in the LR of the exception mode.

3. Link register is set to a specific address based on the current instruction..

   For e.g. for ISR, LR = last executed instruction + 8

4. Update the CPSR about the exception

5. Set the PC to the address of the exception handler.