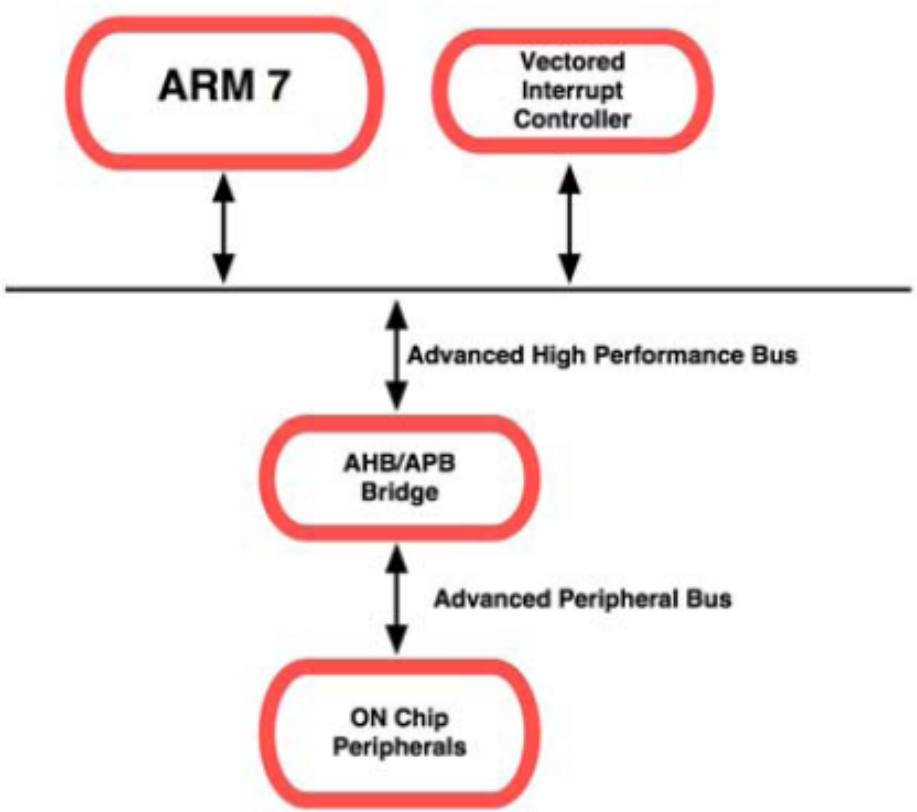
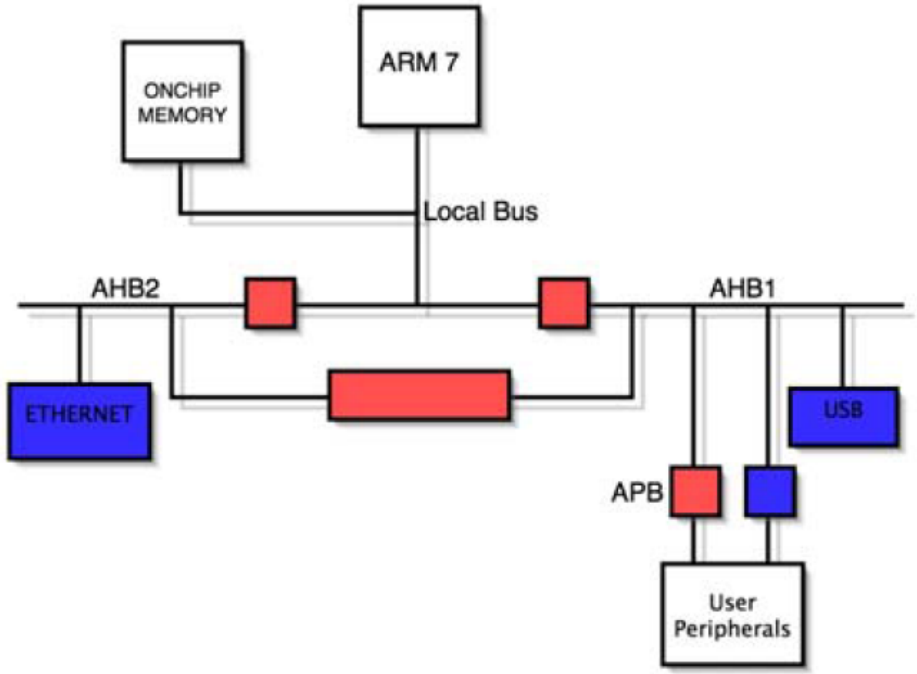
Operating systems and concurrency B02

David Kendall

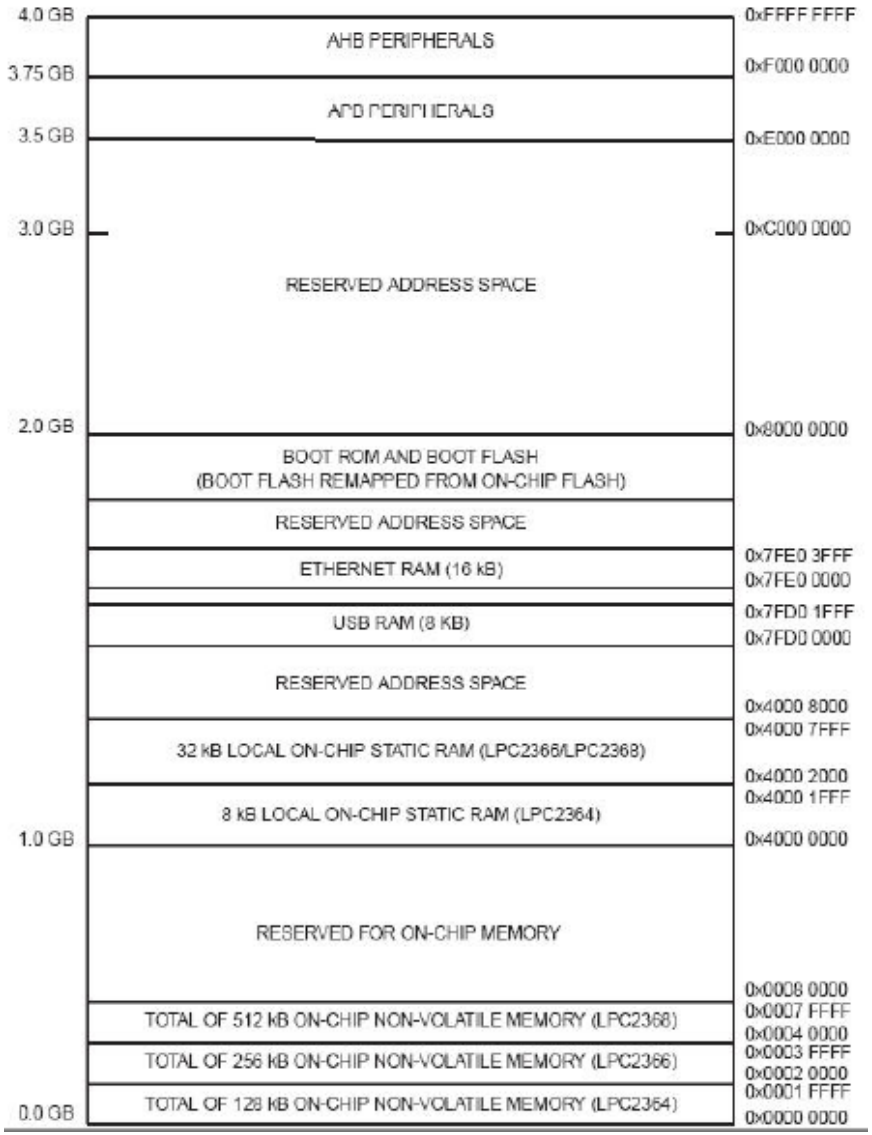
Northumbria University

Device handling - ARM7 bus structure





Device handling - LPC23xx memory map



- Devices on the LPC23xx are controlled by reading and writing to specific locations in the memory map of the microcontroller

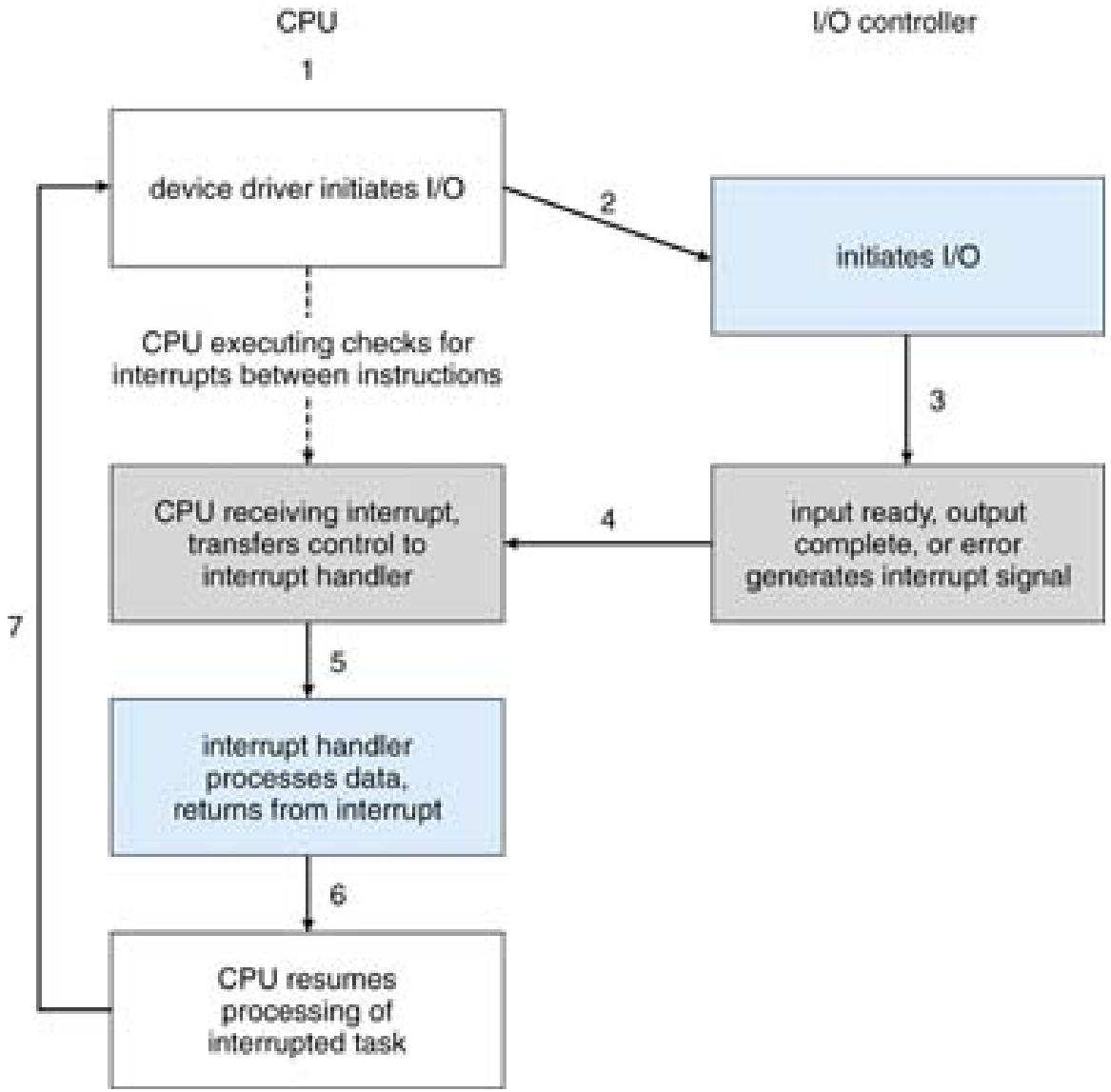
Device handling - Polling

- Typical I/O operation
  1. CPU repeatedly tests status register to see if device is busy
  2. When not busy, CPU writes a command into the command register
  3. CPU sets the command-ready bit
  4. When device see command-ready bit is set, it reads the command from the command register and sets the busy bit in the status register
  5. When device completes I/O operation, it sets a bit in the status register to indicate the command has been completed
  6. CPU repeatedly tests status register, waiting for command to be completed
- Problem: *busy-waiting* at steps 1 and 6

Device handling - Interrupt-driven

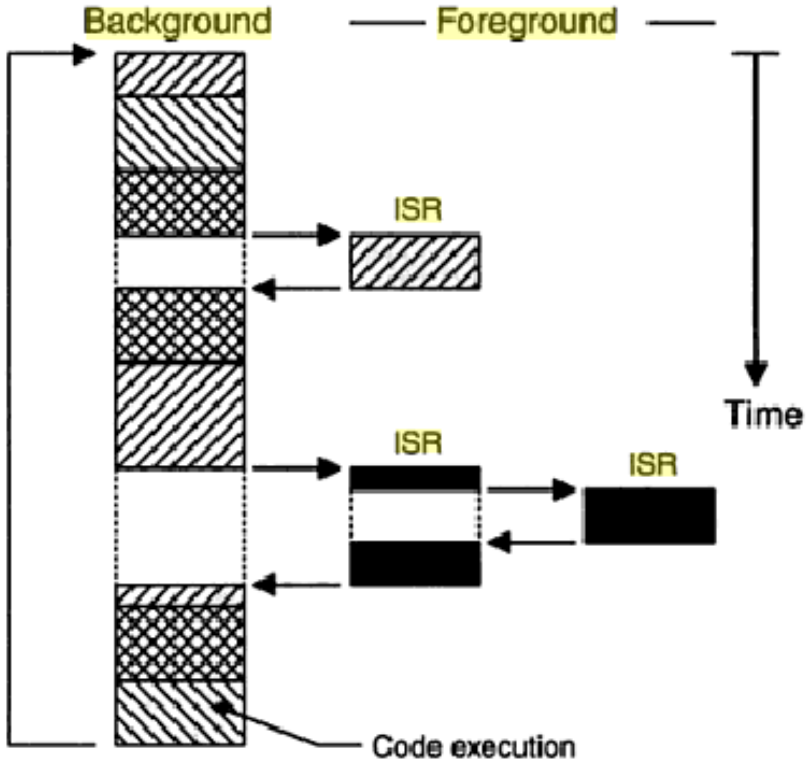
- Busy-waiting can be an inefficient use of the CPU
- CPU could be doing other useful computation instead of waiting
- E.g.
  - Assume: 10 ms for a disk I/O operation to complete; CPU clock speed of 60 MHz; average instruction requires 4 clock cycles – (rough estimates)
  - How many instructions could the CPU execute instead of waiting for the disk I/O?
- So, instead of waiting, CPU performs other useful work and allows the device to *interrupt* it, when the I/O operation has been completed

Interrupt-driven I/O cycle



Simple interrupt-driven program structure

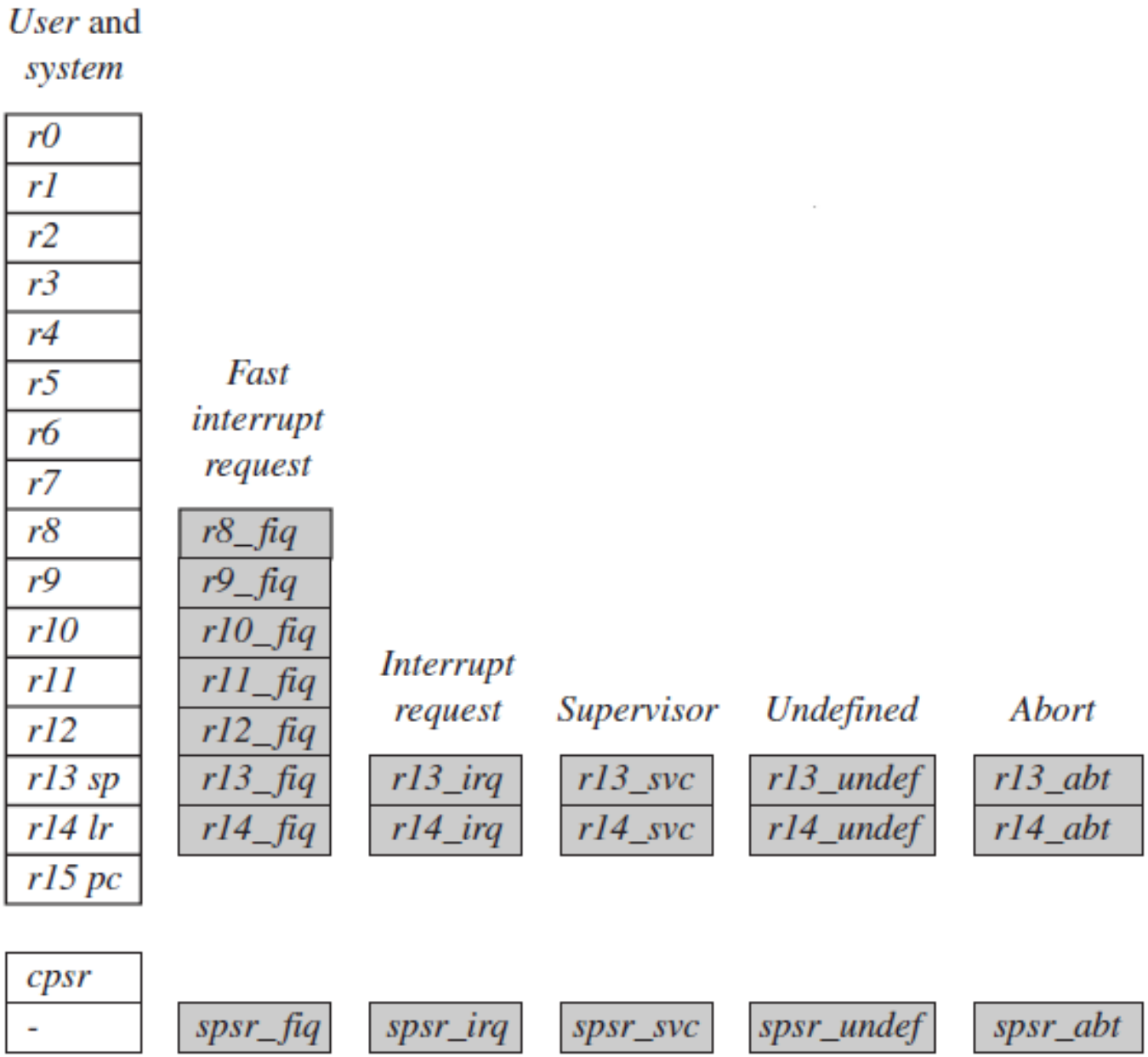
- Foreground / Background
- *Background*: Main (super) loop calls functions for computation
- *Foreground*: Interrupt service routines (ISRs) handle asynchronous events (interrupts)



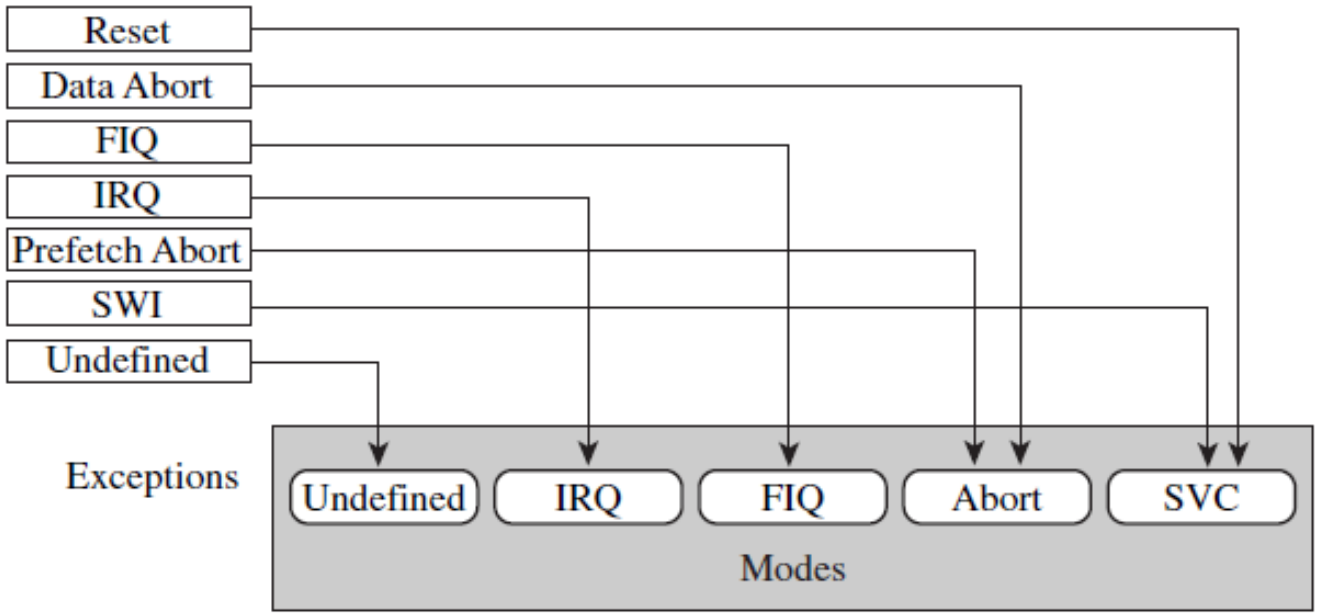
Context switch

- Notice that the state (*context*) of the background task must be restored on returning from servicing an interrupt
  - so that it can carry on its work, after the interrupt has been serviced, *as though it had not been interrupted*
- If the context is to be restored, it must first be saved
- What is the context of the background task?
  - . . . *the complete set of user-mode registers*

ARM7 Complete Register Set



ARM7 Exceptions and associated modes



Note:

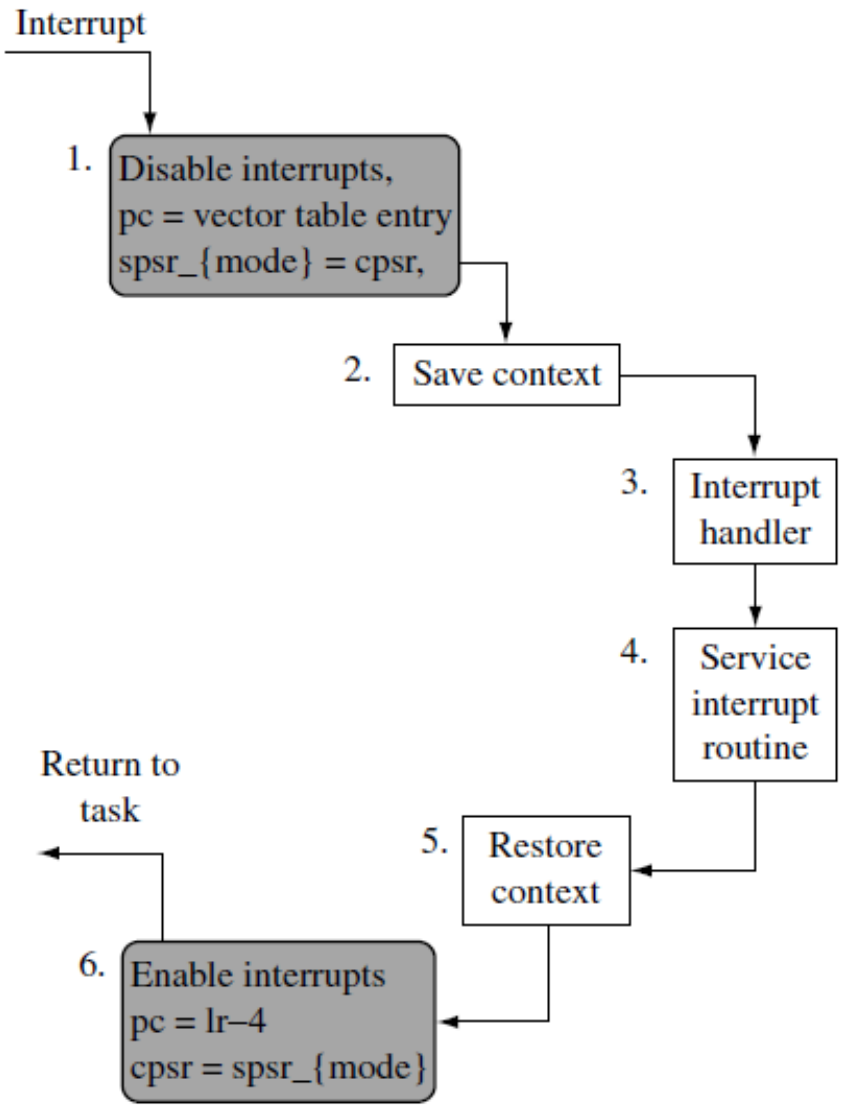
- IRQ and FIQ interrupts cause transfer to corresponding mode
- SWI instruction (software interrupt) causes transfer to *supervisor* mode

ARM7 Vector Table

Address	Exception
0x0000 0000	Reset
0x0000 0004	Undefined Instruction
0x0000 0008	Software Interrupt
0x0000 000C	Prefetch Abort (instruction fetch memory fault)
0x0000 0010	Data Abort (data access memory fault)
0x0000 0014	Reserved **Note:** Identified as reserved in ARM documentation, this location is used by the Boot Loader as the Valid User Program key. This is described in detail in Section 29–3.1.1 .
0x0000 0018	IRQ
0x0000 001C	FIQ

- When an exception occurs, processor automatically transfers control to the instruction at the appropriate entry in the vector table

Simple, non-nested interrupt handler



- *Non-nested*: interrupts remain disabled until control is returned to the interrupted task
- Interrupt handler tests the IRQ status to discover the *source* of the interrupt so that it can call the appropriate service routine.

Acknowledgements and Reading

Acknowledgements

- **[SGG09]** Silberschatz, A., Galvin, P. and Gagne, G., Operating systems concepts, John Wiley, 8th edition, 2009
- **[SSW04]** Sloss, A., Symes, D. and Wright, C., ARM System Developer's Guide, Morgan Kaufmann, 2004
- **[MAR07]** Martin, T., The Insider's Guide to the LPC2300/2400 based microcontrollers, Hitex, 2007

Reading

- SGG09, 13.1 – 13.3
- MAR07, 1.1 – 1.5