SPRING 2017
CMPE 364

Microprocessor Based Design

Dr. Ryan Riley
(Slides adapted from Dr. Mohamed Al-Meer)

Modes, Exceptions & Interrupts

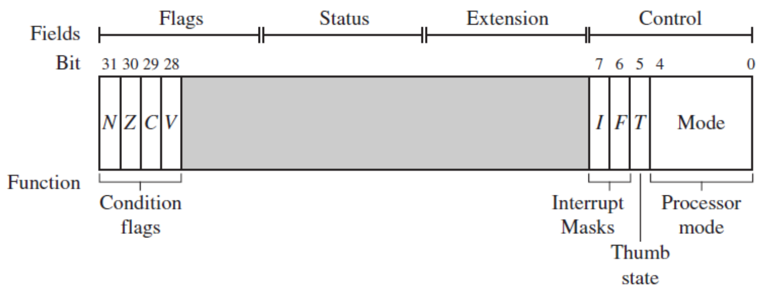
ARM Architecture

Lecture Expectations

- Expected to achieve:
  - Introduction
  - Modes
  - Handling Exceptions
  - Exceptions Details

ARM Processor Modes

- There are seven modes an ARM processor can be running in
- They are used by the processor while performing certain tasks or when certain events occur
- The current mode is stored in the CPSR



ARM Processor Modes

1. User
   - Used for normal program execution
   - Applications run in this mode
   - Limits access to memory and system registers
2. System
   - A special version of User mode used by certain OS designs
3. Supervisor
   - A privileged mode for the operating system
   - Able to access all memory, special register access, etc.

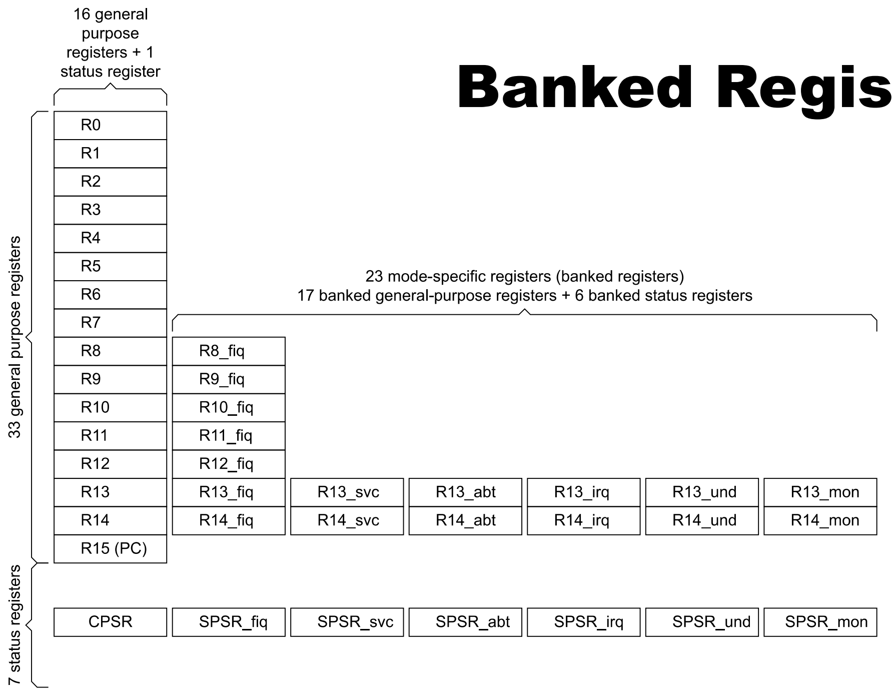
ARM Processor Modes

4. Interrupt (IRQ)
   - Used to handle general purpose interrupts
   - I/O is an example
5. Fast Interrupt (FIQ)
   - A special interrupt handling mode designed to allow for fast interrupt handling
   - Extra registers, faster entrance to the handler, etc.

ARM Processor Modes

6. Abort
   - Entered when a failed memory access occurs
7. Undefined
   - Entered when an unknown instruction is fetched and decoded

Banked Registers



Exceptions

- An exception is any condition that needs to halt the normal sequential execution of instructions.
- Exception handling is the method of processing next exceptions.
  - **when the ARM core is reset,**
  - **when an instruction fetch or memory access fails,**
  - **when an undefined instruction is encountered,**
  - **when a software interrupt instruction is executed, or**
  - **when an external interrupt has been raised.**

Types of Exceptions

There are seven types of exceptions

1. Reset
2. Data Abort
3. Prefetch Abort
4. FIQ
5. IRQ
6. Undefined Instruction
7. Software Interrupt (SWI)

Types of Exceptions

1. Data Abort
   - A memory access failed during a load/store instruction
   - Could be a privilege issue (read/writing somewhere you shouldn't)
   - The load/store that caused the exception will be partially complete, which is annoying
2. Prefetch Abort
   - A memory access failed during an instruction fetch
   - Usually a privilege issue

Types of Exceptions

3. FIQ
   - A fast IRQ was requested
4. IRQ
   - A normal IRQ was requested
5. Software Interrupt (SWI)
   - The program executed an SWI instruction

Types of Exceptions

6. Undefined Instruction
    - An undefined instruction was fetched and decoded
7. Reset
    - Processor is reset because the hardware reset pin is asserted
    - Used to bootstrap the system

Exception Handling

- When an exception occurs, the processor **suspends** the program and **transfers control** to an "exception handler"
    - A subroutine that addresses the cause of the exception
- Before transferring control to the handler, the hardware changes to one of the seven modes previously discussed

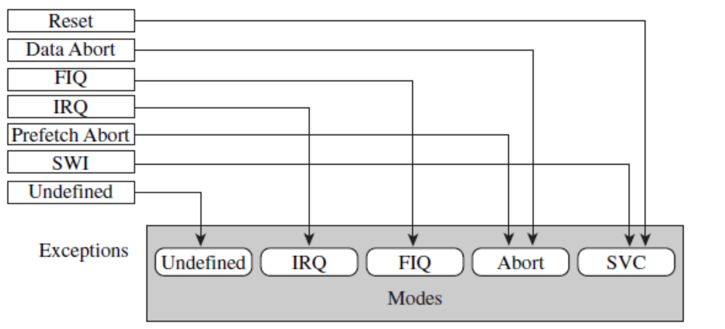
Exception Handling

- An **event** causes the processor to automatically (in **hardware**) to change the mode and transfer control to exception handler
  - Means B and BL are not used to transfer control
- When an exception causes a mode change, the core automatically:
  - ■ 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 correct mode

  Interrupts are disabled
  - ■ sets *pc* to the address of the exception handler

Exception to Mode Mapping

- Exceptions shift the processor into various modes



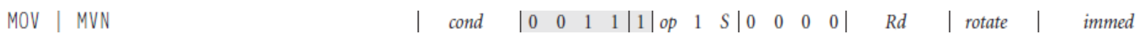
Exceptions and associated modes.

Exception Vector

- Exception vector is used to transfer control to the exception handler
- Vector table stores all the Exception Pointers for every type of exceptions
  - Comprises of a **single Branch Instruction** that branches to the specified handler.
- It always contains a branching instruction in one of the following forms:

Exception Vector

- **B <Add>**
  This instruction is used to make branching to the memory location with address "**Add**" relative to the current location of the pc.
- **LDR pc, [pc, #offset]**
  This instruction is used to load in the program counter register its old value + an offset value equal to "**offset**".
- **MOV pc, #immediate**
  Load in the program counter the value "immediate".

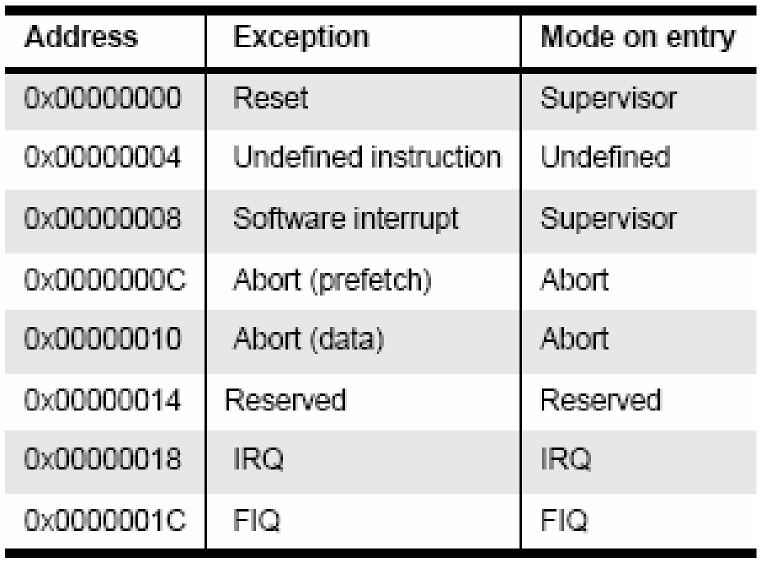


MOV | MVN	*cond*	0 0 1 1	1	*op*	1	*S*	0 0 0 0	*Rd*	*rotate*	*immed*

Exception Vector

- **LDR pc, [pc, #-0xff0]**
  This *load register instruction* loads a specific interrupt service routine address from address 0xfffff030 to the *pc*.

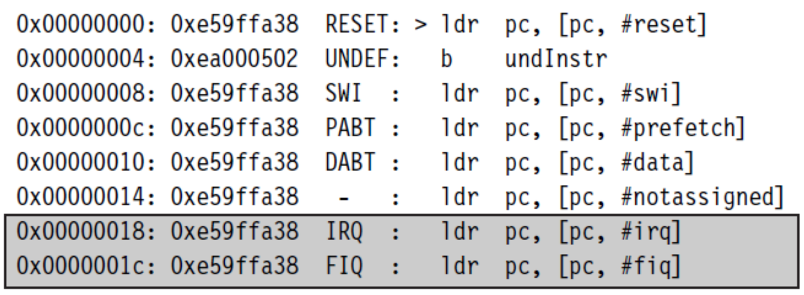
Vector table with Branching



Address	Exception	Mode on 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 Vector - Example

- Next Figure shows a typical vector table.
  - The Undefined Instruction entry is a branch instruction to jump to the undefined handler.
  - The other vectors use an indirect address jump wit the LDR load to *pc* instruction.



```
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]
```

Exception Priorities

- Exceptions can be generated from
    - **internal** or
    - **external** source.
  - **Internal source**: is a software interrupt while
  - **External source**: is a input or an output device trying to interrupt the processor
- What happens if multiple interrupts happen in the same time?
  - **The Exception with higher priority is served first**.
  - Next Table shows the various exceptions that occur on the ARM processor and their associated priority level.

Exception Priorities

Exception priority levels.

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	—

Entering an exception handler

- Preserve the address of the next instruction.
- Copy ***CPSR*** to the appropriate ***SPSR***, which is one of the banked registers for each mode of operation.
- Force the ***CPSR*** **mode** bits to a value depending on the raised exception.
- Force the ***PC*** to fetch the next instruction from the exception vector table.
  - Now the handler is running in the mode associated with the raised exception.
- When handler is done, the ***CPSR*** is restored from the saved ***SPSR.***
- ***PC*** is updated with the value of (***LR*** – **offset**) and the offset value depends on the type of the exception.

Returning from an exception handler

- Move the **Link Register *LR*** (**minus an offset**) to the ***PC***.
  - **The Reason?**: due to pipelining the address stored in LR may not be always the interrupted instruction.
  - **But how to tell the correct?** See next table to evaluate the ***offset***.
- Copy ***SPSR*** back to ***CPSR,*** this will automatically changes the mode back to the previous one.
- **Clear** the **interrupt disable flags** (if they were set).

Returning from an exception

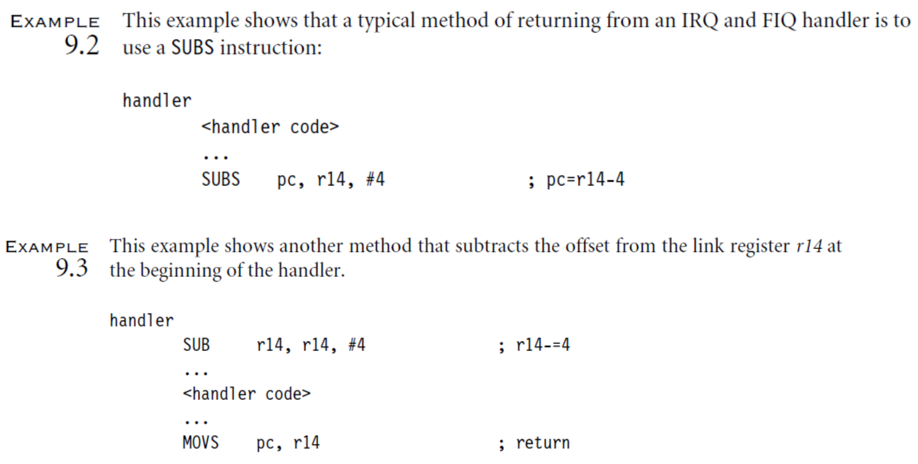
Table 9.4 Useful link-register-based addresses.

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

Returning from an exception

- Instructions used to return from exceptions:
- Use of **SUBS**
  - **SUBS pc, r14, #4 ; pc = r14 – 4** **-** (See Table 9.4)
- Use of **MOV**
  - **MOVS pc, r14 ; return**
- Use of **Load Multiple Stack**
  - **LDMFD r13!,{r0-r3, pc}ˆ ; return**

Returning from an exception



EXAMPLE 9.2 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
```

EXAMPLE 9.3 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
```

Returning from an exception

Example 9.4 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}^           ; return
```

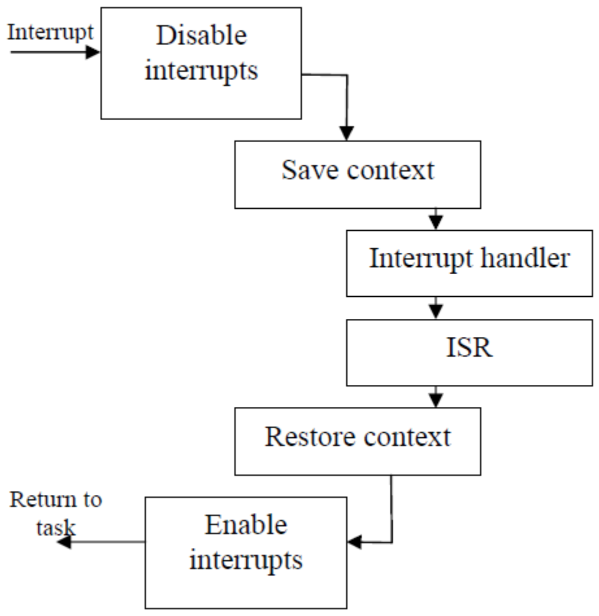
More on Interrupts

- **Two types**
  - caused by external events from **hardware peripherals**
  - The second type is the **SWI instruction**.
- **How are interrupts assigned?**
  - System designed (hardware and software)

More on Interrupts

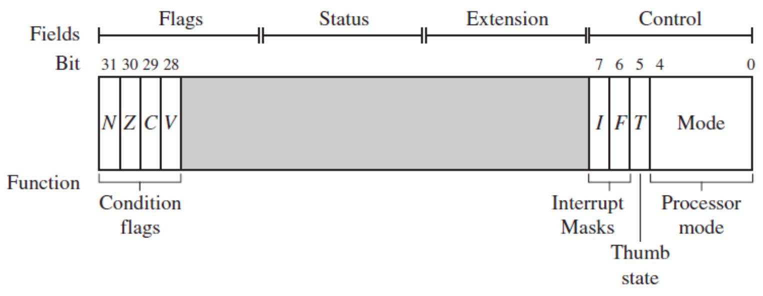
- There is a standard design for assigning interrupts adopted by system designers:
  - **SWIs** are normally used to call privileged operating system routines.
  - **IRQs** are normally assigned to general purpose interrupts like periodic timers.
  - **FIQ** is reserved for one single interrupt source that requires fast response time, like DMA or any time critical task that requires fast response.

Non-nested interrupt handling



Disabling interrupts?

- The CPSR has bits for enabling/disabling interrupts



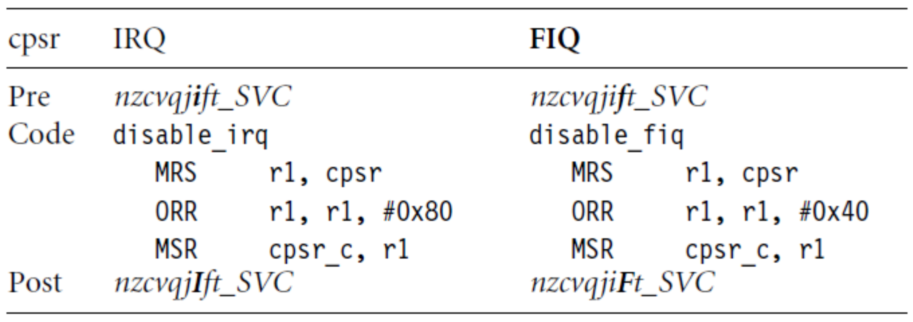
- If you are in a privileged mode, you can modify those bits

Enabling and Disabling FIQ and IRQ Exceptions

- The ARM processor core manually enable and disable interrupts.
- Table 9.5 shows how IRQ and FIQ interrupts are enabled. The procedure uses **three ARM instructions**.
  - **MRS** copies the contents of the ***cpsr* into register *r1***
  - The second instruction clears the IRQ or FIQ mask bit
  - The third instruction (**MSR**) then copies the **updated contents in register *r1* back into the *cpsr***., enabling the interrupt request

Disabling interrupts

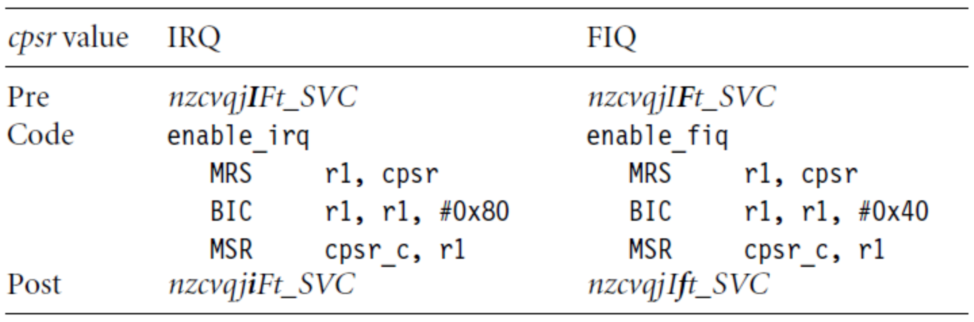
Disabling an interrupt.



cpsr	IRQ	**FIQ**
Pre	*nzcvqj**i**ft_SVC*	*nzcvqji**f**t_SVC*
Code	`disable_irq`	`disable_fiq`
	`MRS r1, cpsr`	`MRS r1, cpsr`
	`ORR r1, r1, #0x80`	`ORR r1, r1, #0x40`
	`MSR cpsr_c, r1`	`MSR cpsr_c, r1`
Post	*nzcvqj**I**ft_SVC*	*nzcvqji**F**t_SVC*

Enabling interrupts

Enabling an interrupt.



cpsr value	IRQ	FIQ
Pre	*nzcvqj**I**Ft_SVC*	*nzcvqjI**F**t_SVC*
Code	`enable_irq`	`enable_fiq`
	`MRS r1, cpsr`	`MRS r1, cpsr`
	`BIC r1, r1, #0x80`	`BIC r1, r1, #0x40`
	`MSR cpsr_c, r1`	`MSR cpsr_c, r1`
Post	*nzcvqj**i**Ft_SVC*	*nzcvqjI**f**t_SVC*