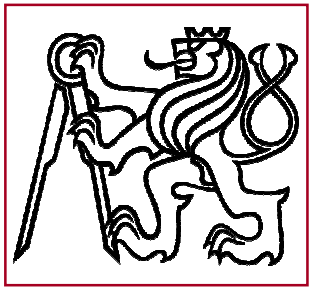
Y35PES

Programming for Embedded Systems

Ondřej Špinka, Pavel Němeček
spinkao@fel.cvut.cz nemecp1@fel.cvut.cz
http://dce.felk.cvut.cz/pes



Atmel Advanced Interrupt Controller (1)

Note – the AIC is not a standard ARM core peripheral, it is a proprietary Atmel device.

The **Advanced Interrupt Controller (AIC)** is an 8-level priority, individually maskable, vectored interrupt controller, providing handling of up to thirty-two interrupt sources. It is designed to substantially reduce the software and real-time overhead in handling internal and external interrupts.

The AIC drives the nFIQ (fast interrupt request) and the nIRQ (standard interrupt request) inputs of an ARM processor. Inputs of the AIC are either internal peripheral interrupts or external interrupts coming from the processor pins.

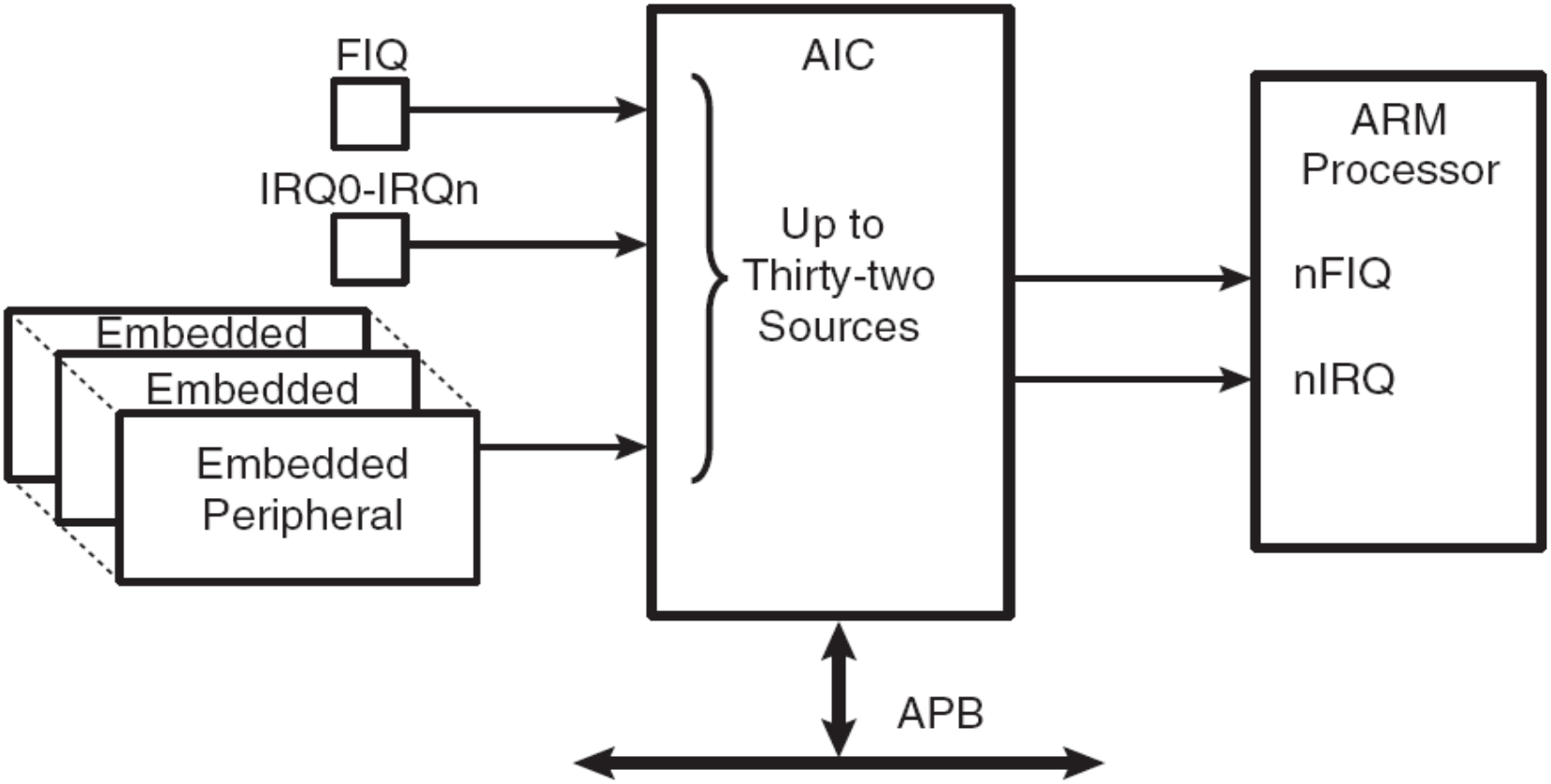
The 8-level Priority Controller allows the user to define the priority for each interrupt source, thus permitting higher priority interrupts to be serviced even if a lower priority interrupt is being treated.

Internal interrupt sources can be programmed to be level sensitive or edge triggered.

The fast forcing feature redirects any internal or external interrupt source to provide a fast interrupt rather than a normal interrupt.

Atmel Advanced Interrupt Controller (2)

AIC connection diagram (picture property of Atmel Corp.)



Atmel Advanced Interrupt Controller (3)

Most importantly, the AIC provides a straightforward way to implement **directly vectored interrupts**, therefore simplifying the interrupt handlers implementation, and also speeding up the interrupt treatment.

The interrupt handler addresses corresponding to each interrupt source can be stored in the registers AIC_SVR1 to AIC_SVR31 (Source Vector Register 1 to 31). When the processor reads AIC_IVR (Interrupt Vector Register), the value written into AIC_SVR corresponding to the current interrupt is returned.

This feature offers a way to branch in one single instruction to the handler corresponding to the current interrupt, as AIC_IVR is mapped at the absolute address 0xFFFF F100 and thus accessible from the ARM interrupt vector at address 0x0000 0018 through the following instruction:

ldr pc, [pc, #-&F20]

When the processor executes this instruction, it loads the read value in AIC_IVR in its program counter, thus branching the execution on the correct interrupt handler.

Volatile Variables

When changing a variable value **asynchronously**, special care must be taken to avoid possible code optimization related problems.

```
volatile int a = 0;

void __irq irq_handler ( ) {
        ...
        a = 1;
        ...
}

void main ( ) {
        ...
        while ( !a );
        ...
}
```

Program Safety

When working with exceptions, one must take a lot of possible dangers into consideration. Let us note at least some of the most frequently occurring problems:

- Asynchronously accessed variables (usage of *volatile* type)
- Possible interruption of a critical process – i.e. external memory access, time-critical operation… (global exceptions disabling)
- Unintentional overwrite of a shared variable – possibly in the course of computation (usage of local buffers)
- Introduction of non-deterministic latencies into main program flow and lower-priority event handling (careful priority assessment, fast handlers, time complexity evaluation, worst-case consideration)
- Exceptions nesting – possible stack overflow (worst-case evaluation)

ARM Assembler Basics (1)

ARM instruction set can be divided into six categories:

- Branch instructions
- Data-processing instructions
- Status Register transfer instructions
- Load and store instructions
- Coprocessor instructions
- Exception-generating instructions

Almost all ARM instructions can be executed **conditionally**, depending on the **condition flags status**. Conditional execution is determined by the 4-bit **condition field,** which is a part of the instruction **opcode**. The conditions allow:

- Tests for equality and non-equality
- Tests for inequalities <, >, <=, >= (both signed and unsigned)
- Unconditional execution

ARM Assembler Basics (2)

Branch instructions:

- **Unconditional jumps** up to +-32MB (26-bit signed offset) can be performed by directly writing R15 (Program Counter) or by using B (Branch) instruction (equivalent)
- **Conditional branching** forward or backward up to +-32MB (26-bit signed offset)
  - BL (Branch and Link) instruction
    - Preserves address of the next instruction in R14 (LR)
    - Used to subroutine call
    - Return is performed by simply moving R14 (LR) into R15 (PC)
  - BCC (Branch if Carry Clear)
  - BEQ (Branch if Equivalent, that is if the Zero flag is set)
- There are also subroutine calls allowing to change the instruction set (ARM-Thumb or vice versa)
  - BX (Branch and Exchange) instruction (Branches and switches instruction set to Thumb)
  - BLX (Branch, Link and Exchange) – like BX, but preserves R14 (LR)

ARM Assembler Basics (3)

Branch instructions addressing:

The address contained in the instruction operand can be max. 26 bits long (8 bits are reserved for instruction opcode; the operand can be 24 bits long. As each instruction address must be word-aligned, the two least significant bits are always 0 and therefore are omitted), allowing relative branches in the range of +-32MB.

The address of the branch can be computed from the operand using following algorithm:

- Sign-extending the 24 bit operand to 32 bits
- Shifting the result left by two bits (the two zero LSBs)
- Adding the contents to PC

ARM Assembler Basics (4)

Long jumps:

Another approach is needed to perform jumps over the +- 32MB relative range. Those so-called **long jumps** must be performed in two consecutive steps:

- Prepare the target address into a suitable memory location
- Move the memory content directly into the PC

ARM Assembler Basics (5)

Data-processing instructions:

- **Arithmetic/Logic instructions**
  - There are twelve instructions sharing a common format (up to two operands (sources) and one result (target))
    - One of the source values must always be a register
    - The other can be an immediate value, or a register (optionally shifted)
  - They can optionally update the condition flags
  - The R15 (PC) can be used directly as the target register (!), allowing to implement various jumps very elegantly
- **Comparison instructions**
  - There are four comparison instructions
  - Basically, they are arithmetic/logic instructions, but not writing the result anywhere and always updating the condition flags
- **Multiply instructions**
  - Support 32-bit or 64-bit (long) results (long numbers are stored into two registers)
- **Count Leading Zeros instruction** (CLZ) – determines the number of Most Significant Bits (MSBs) that are zero

ARM Assembler Basics (6)

Status Register transfer instructions:

- Serve to transfer contents of a general-purpose register to or from the Program Status Register (CPRS or SPRS)
- There are two Status Register transfer instructions
  - MRS – moves PSR contents to a general-purpose register
  - MSR – moves a general-purpose register contents to PSR

ARM Assembler Basics (7)

Load and store instructions:

- Serve to transfer contents of a register to or from the memory
- Load and store a single register instructions
  - Load and store a **32-bit word**, **16-bit half-word** or **8-bit byte**, optionally with **zero-extension** and/or **sign extension**
  - Three addressing modes, all using a **Base Register** and an **Offset**
    - Offset can be specified by a register value (optionally shifted) or an immediate value
    - **Offset addressing** – offset is directly added or subtracted to/from the Base Register value
    - **Pre-indexed addressing** – same as previous, but the computed address is stored back into the Base Register as a side-effect
    - **Post-indexed addressing** – the Base Register directly forms the address, the offset is added/subtracted afterwards and written back to the Base Register as a side-effect

ARM Assembler Basics (8)

Load and store instructions (continued):

- Load and store multiple registers instructions
  - LDM (Load Multiple) and STM (Store Multiple) instructions perform a **block transfer** of any number of general-purpose registers to and from the memory
  - Four addressing modes are available:
    - Pre-increment
    - Post-increment
    - Pre-decrement
    - Post-decrement
  - Multiple registers accessing functions allow very efficient **subroutine entry/exit operations** and **block copies**
- Swap register/memory contents instructions
  - The SWP (Swap) instruction serves to swap register/memory contents. It is very useful for Operating Systems to perform **atomic semaphore updates**.

ARM Assembler Basics (9)

Coprocessor instructions:

There are three types of coprocessor instructions:

- **Data-processing instructions** – start a coprocessor-specific internal operation
- **Data transfer instructions** – transfer coprocessor data to/from the memory
- **Register transfer instructions** – serve for ARM core – coprocessor data transfers

ARM Assembler Basics (10)

Exception-generating instructions:

There are three types of exception-generating instructions:

- **Software interrupt instructions** – these are often used to perform OS service calls
- **Software breakpoint instructions** – similar to SW interrupts, serve to accommodate debugger functions
- **Unrecognized instructions** (for example coprocessor instructions when no coprocessor is present) – used either to generate an error or to emulate unavailable instructions

Lecture Summary – Essential Things to Remember

- The **Advanced Interrupt Controller (AIC)** is an 8-level priority, individually maskable, vectored interrupt controller, providing handling of up to thirty-two interrupt sources.
- **Asynchronously** accessed variables must be declared as **volatile**, to avoid possible code optimization related problems. Certain precautions must be taken in order to maintain program safety when working with exceptions.
- **ARM** instruction set can be divided into **6 categories** (name them)
- Almost all ARM instructions could be executed **conditionally**.
- There are three types of exception generating instructions, i.e. **Software interrupt instructions**, **Software breakpoint instructions** and **Unrecognized instructions**.