COMP2121 Assignment - z5016815

Introduction

ARM is a form of micro-controller architecture, that has become widely used over time in several embedded system applications. The ARM7TDMI is an extension of ARM and is recognised as one of the most popular ARM cores. In short, the ARM7TDMI is a 32-bit embedded processor that has been specialized for powerful performance within the constraints of relatively low power and size. [1]

The ARM7TDMI incorporates a 32-bit ARM instruction set, which is supplemented by a 16-bit thumb instruction set, that provides the ARM7TDMI with enhanced flexibility, allowing it to manipulate code instructions on a sub-routine level and diversify the application of the core (such as allowing the core to effectively utilize fast interrupts). The ARM7TDMI is also well known for having very low power consumption and operating on a very small scale. This is advantageous in our current society where users have a strong preference for smaller and more streamlined products. In addition, the ARM7TDMI sports a 32-bit Arithmetic Logic Unit as well as a 32-bit data bus that is used to transfer both instructing coding and data. The ARM7TDMI is also equipped with enhanced debug features that allow for improved performance and programmability. [2]

Some of the other salient features of the ARM7TDMI include conditional execution, whereby after comparing two different values, the ARM7TDMI is capable of implementing specific instructions if certain conditions are met. This improves efficiency as it can be done without the use of branching. The ARM7TDMI also utilizes caching. This can be achieved by adding on an optional component, granting the ARM7TDMI enhanced flexibility. The ARM7TDMI also has access to efficient hardware support for power saving, this involves setting the processor to a semi-sleep state when the microprocessor itself is not in use. [3]

Memory Models

ARM microprocessors only tend to utilize one memory space in a form that is commonly referred to as Von Neumann architecture. Generally speaking, ARM microprocessors typically possesses a maximum memory space with the equivalent of roughly 4 gigabytes. This means that they employ the use of a 32-bit memory space. This space is divided into address and processor registers (both of which are 32-bit). For I/O devices, the registers are often mapped into this memory space. This memory space also keeps track of all the code, instructions and the data. This means that there is no separate memory space for both the program and its constants. The addressing tends to be 4-byte aligned for words. For halfwords, the addressing is 2-byte aligned. [4]

The ARM architecture is little-endian (bits and bytes are numbered from the little-end), although in version 3, this architecture has been updated to bi-endian, which allows ARM to operate in both little-endian and big-endian mode. For ARM7TDMI, data can be represented in 8-bit (referred to as bytes), 16-bit (referred to as halfwords) and 32-bit (referred to as words).[5]

Registers

The ARM7TDMI typically possesses 37 registers. These registers are sorted into general purpose registers and status registers. Altogether, there are a total of 31 general purpose registers and 6 status registers. Register 14 is commonly used as the link register, although the register often operates similarly to a general purpose register, it also acts as a holder for the return address and helps redirect the processor after it has completed a specific function call. The PC or the program counter tends to point to the next instruction that is being fetched and not the next instruction that is being implemented, so it varies. The CPSR or the current program status register is another unique register that is often used to store conditional code information.[6]

Flags

The N, Z, C and V flags are important components of the ARM7TDMI microprocessor. The N flag is used to represent negative. This flag is just a single bit. If this flag is set to zero, then that would mean that the value is positive, however, if it is set to one, then that would mean a value is negative. The Z flag refers to zero. This bit would return a 1 if the final result of an operation was 0, in contrast, if the result was non-zero then the bit would return a 0 instead. The C flag represents the carry flag. This flag determines whether there is a carry over unsigned values. For example, when adding 2 values together, if a number needed to be carried over, then they carry flag would signal a 1 to show that. Otherwise, the carry flag would be set to 0 instead. The V flag stands for the overflow flag. This is quite similar to the carry flag, except it is used for signed values. Likewise to the C flag, if there is a carry-over in addition for example, the V flag would indicate a 1, otherwise the V flag would indicate just a 0. [7]

Operating Modes

Generally speaking, the 5 bits that are the least significant (0, 1, 2, 3, 4) are used as the operating modes. Usually, the processor is operating in the default mode. However, when these bits are modified, the processor is able to switch to different operating modes, with each bit possessing different and unique features. When the bits are set to 10000 (first bit 1, second, third, fourth and fifth bits are 0), then the microprocessor is generally in the user mode. This is just the default mode, where all the user registers are available for use, this is used for the majority of applications that require the microprocessor. When the bits are set to 10001, then we are able to achieve the FiQ mode. This mode is commonly referred to as the fast interrupt mode and is usually used for transferring across data or processing channels. 10010 represents the IRQ mode, this mode is used for interrupts and handles interrupts across the general purpose registers. 10011 denotes the SVC mode, or commonly known as the supervisor mode, this mode is a protected mode and is usually enabled after an encounter with a software interrupt. 10111 denotes the abort mode, this mode is entered after the microprocessor has experienced some issues with memory or data faults. This mode is entered into after the instruction Prefetch Abort has been called. 11111 represents the system mode, this mode is primarily for privileged users to run privileged tasks that tend to be unavailable to regular users. System mode is unique in that it can only be accessed after having previously accessed another privileged mode via modification of the CPSR (current program status register). The final operating mode is the undefined mode (und), as determined by the bits 11011.

The undefined mode is accessed once the operating system processes an undefined instruction that it has never encountered before, and is not recognised by the processor or the co-processors. [8]

Interrupt

As mentioned above, interrupts tend to occur once the operating mode transitions from the user mode to the interrupt modes FiQ and IRQ (which are the fast interrupt processing mode and the normal interrupt processing mode respectively). The FiQ and the IRQ are representative of standard hardware interrupts, these interrupts are not really part of the CPU tasks. For the ARM7TDMI interrupt system, the processor uses the current program status register (CPSR) and 5 saved program status registers (SPSR) to effectively enable and disable interrupts. The bits that enable and disable interrupts are stored in the control bits. These bits are referred to as the T, I and F bits. T represents the Thumb bit, and is used to interchange between Thumb instructions and the regular ARM instructions. The I bit refers to interrupts, when this bit is set to 1, it indicates that a normal or standard interrupt has been enabled. The F bit stands for Fast Interrupt. When this bit is set to 1, it indicates that a fast interrupt has occurred and been enabled. All these bits are located in the bottom 8 bits of the PSR (program status register).

To complement hardware interrupts, there are also interrupts referred to as software interrupts. These interrupts are often generated from the software (instructions denoted in the code) and not at the hardware level. It is a common way to invoke system calls while handling exceptions. [9]

Instruction Sets

Operands

All operands are usually 32-bits. Operands are based off registers and the final results are also stored in registers. ARM7TDMI incorporates a three operand instruction set, in combination with a 32-bit encoding scheme. Typically speaking, before being operated on, all values must be loaded into a register. After the values have been loaded into each register, then operands can be used on them. There are two types of instruction sets. The first instruction set is the regular 32-bit instruction set and the other instruction set is the 16-bit thumb instruction set. The thumb instruction set mainly allows for the microprocessor to change code on a subroutine level, providing the processor with enhanced flexibility and effectiveness. [10]

Addressing Modes

Typically for ARM7TDMI, there are 5 standard addressing modes (modes 1 - 5). Mode 1 stipulates the Shifter operands for data processing instructions, mode 2 is used for loading and storing the values for unsigned bytes and words. Mode 3 is used for loading and storing signed bytes and halfwords. Mode 4 is for storing and loading multiples and mode 5 is for storing and loading additional coprocessors to assist with the existing processor. [11]

Conditional Execution

As mentioned previously conditional execution is whereby after comparing two different values, the ARM7TDMI microprocessor is capable of implementing specific instructions if certain conditions are met. This is accomplished through a two letter condition after the initial code. This means that actual branching can be avoided in the code, by utilizing more complex instructions that only execute if certain flags are active. The overall advantage of this type of execution is that apart from incorporating less branches in coding, we are also able to use less power, thus making the processor more efficient. Also the overall code would be smaller and more comprehensible by other users. [12]

Stack Operations

Stack Operations are another component of ARM7TDMI processors. Stacks are forms of data structures where the data is bound by 2 functions (pop and push) that either add an element to the data structure or remove the most recently added element from the data structure. The ARM7TDMI processor utilizes the transfers between combinations of multiple registers and instructions to simulate the effect of the pop and push instructions. Generally speaking, there are four types of stacks that are used for ARM processors, these are full ascending, empty ascending, full descending, and empty descending. The Ascending and Descending conditions specify the growth of the stack in the processor memory. A full stack indicates that the stack points to the last item added on top of the stack and the empty stack indicates that the stack pointer points to the next address that is available for an element to be pushed onto the stack.

The Stack pointer is the register that points to the top of the stack, this means that if an element was recently added to the stack, the stack pointer would point to that element. Otherwise, when initializing an empty stack, the stack pointer would point to the location where an element would be pushed onto the stack. [13]

Multiple bit shift instructions

ARM7TDMI provides the use of multiple bit shift instructions, as the name suggests, these instructions can be shifted across multiple bits rather than only 1 at a time. The ARM7TDMI employs a 32 by 32 matrix to shift these bits accordingly. The corresponding rows in this matrix are then turned on to signify a left or right shift, thus connecting an input bit with an output bit. [14]

In and Out registers

AVR uses In and Out registers (commonly referred to as I/O registers). The use of these registers is very memory efficient as they tend to be stored away from the actual AVR memory, thus making them more easily accessible. Unfortunately, ARM processors such as the ARM7TDMI does not possess I/O registers. Instead ARM processors utilize a I/O in a memory mapped format. In this format, all registers are present and appear on the memory map. Unlike AVR, instructions on these registers can only follow the existing instruction set that is presented for all standard registers. Therefore in this regard, AVR shows higher utility and flexibility than ARM processors. [15]

Power Saving

AVR machines have access to sleep instructions that allow the microcontrollers to save power by enabling a sleep mode during periods of low user activity. ARM also have access to a similar system. The ARM processor utilizes a command referred to as Wait for Interrupt. This command puts the processor into a period of suspended action whereupon it does nothing until a certain interrupt has been received (effectively acts as a pseudo-sleep mode). This interrupt is then received by Fast Interrupt Processing Mode (FiQ) or the normal interrupt processing mode (IRQ) and the processor then exits from its current pseudo-sleep mode. While the processor is experiencing the wait for interrupt command, the processor experiences reduced power consumption. [16]

Watchdog Timer

The watchdog timer is a timer that is present in AVR machines. AVR machines are unique in that they have access to the wdr instruction for resetting the watchdog timer. ARM processors like the ARM7TDMI processor also have access to this useful instruction in the form of a PRESETn and WDOGRESn instructions. This set of instructions is used as signals for a reset. [17]

Data Types

As mentioned above, the ARM7TDMI processor has access to 3 main data types. These data types include words (represented in 32-bit), halfwords (represented in 16-bit) and bytes (which are represented in 8-bit). These follow the respective alignments where words(32-bit) are aligned to four-byte boundaries, halfwords (16-bit) are aligned to two-byte boundaries and bytes (8-bit) are aligned with one-byte boundaries. For 64-bit data types that are usually inaccessible with the standard data type structure for ARM7TDMI processors, we can simply use 2 registers to represent this value effectively. These 2 registers have 32-bits each (together they make up 64). An example of a program that adds 2 different 64-bit signed and unsigned integers is shown below given by the ARM information centre. [18]

```
ADDS    R4, R0, R2
ADC     R5, R1, R3
```

Below is a subtraction example, that has also been supplied by the ARM information centre.[19]

```
SUBS    R6, R6, R9      ; subtract the least significant words
SBCS    R9, R2, R1      ; subtract the middle words with carry
SBC     R2, R8, R11     ; subtract the most significant words with carry
```

Conclusion

The AVR microcontroller is an 8-bit processor that is commonly found in simple applications. As it is only an 8-bit processor, it is not nearly as complex and intricate as the 32-bit processor found in ARM models. However, AVR microcontrollers are cheaper in this regard, this makes them quite effective in larger and

more simplistic applications that require less processing power, such as engines, motors and other forms of machinery. [20][21]

In contrast, all ARM processors are 32-bit processors. This makes them extremely flexible due to its large variety of features as well as its faster processing speed and improved memory. ARM7TDMI processors are also effective as they require low power to operate, making them very convenient for smaller applications that are required to operate on low power. This low power cost combined with its relatively low price makes them favourable and accessible to consumers and suppliers. Typical ARM7TDMI uses are found in small video consoles such as the mobile phones like the Nokia 6110, due to the qualities mentioned above. [22]

Bibliography

1. ATMEL. 2017. *The ZAP Group*. [ONLINE] Available at:http://www.zap.org.au/elec2041-cdrom/board/doc/data/atmel-arm7tdmi-datasheet-summary.pdf [Accessed 5 September 2017]
2. ARM7TDMI - eLinux.org. 2017. *ARM7TDMI - eLinux.org*. [ONLINE] Available at: http://elinux.org/ARM7TDMI. [Accessed 5 September 2017].
3. Atmel. 2017. *ARM7TDMI Datasheet*. [ONLINE] Available at: http://www.atmel.com/Images/DDI0029G_7TDMI_R3_trm.pdf. [Accessed 5 September 2017].
4. Atmel. 2017. *ARM7TDMI Datasheet*. [ONLINE] Available at: http://www.atmel.com/Images/DDI0029G_7TDMI_R3_trm.pdf. [Accessed 5 September 2017].
5. Endianness - Wikipedia. 2017. *Endianness - Wikipedia*. [ONLINE] Available at: https://en.wikipedia.org/wiki/Endianness. [Accessed 5 September 2017]
6. Atmel. 2017. *ARM7TDMI Datasheet*. [ONLINE] Available at: http://www.atmel.com/Images/DDI0029G_7TDMI_R3_trm.pdf. [Accessed 5 September 2017].
7. Atmel. 2017. *ARM7TDMI Datasheet*. [ONLINE] Available at: http://www.atmel.com/Images/DDI0029G_7TDMI_R3_trm.pdf. [Accessed 5 September 2017].
8. Processor modes - ARMwiki. 2017. *Processor modes - ARMwiki*. [ONLINE] Available at: https://www.heyrick.co.uk/armwiki/Processor_modes#SVC_mode. [Accessed 6 September 2017].
9. Institut für Technische Informatik. 2017. *Institut für Technische Informatik*. [ONLINE] Available at: http://www.iti.uni-stuttgart.de/~radetzki/Seminar06/08_report.pdf. [Accessed 6 September 2017].
10. Atmel. 2017. *ARM7TDMI Datasheet*. [ONLINE] Available at: http://www.atmel.com/Images/DDI0029G_7TDMI_R3_trm.pdf. [Accessed 6 September 2017].
11. Atmel. 2017. *ARM7TDMI Datasheet*. [ONLINE] Available at: http://www.atmel.com/Images/DDI0029G_7TDMI_R3_trm.pdf. [Accessed 6 September 2017].
12. Atmel. 2017. *ARM7TDMI Datasheet*. [ONLINE] Available at: http://www.atmel.com/Images/DDI0029G_7TDMI_R3_trm.pdf. [Accessed 6 September 2017].
13. Atmel. 2017. *ARM7TDMI Datasheet*. [ONLINE] Available at: http://www.atmel.com/Images/DDI0029G_7TDMI_R3_trm.pdf. [Accessed 6 September 2017].

14. Atmel. 2017. *ARM7TDMI Datasheet*. [ONLINE] Available at: http://www.atmel.com/Images/DDI0029G_7TDMI_R3_trm.pdf. [Accessed 6 September 2017].
15. Atmel. 2017. *ARM7TDMI Datasheet*. [ONLINE] Available at: http://www.atmel.com/Images/DDI0029G_7TDMI_R3_trm.pdf. [Accessed 6 September 2017].
16. infocenter.arm.com. 2017. *No page title*. [ONLINE] Available at: http://infocenter.arm.com/help/topic/com.arm.doc.dht0008a/CJAHGJEF.html. [Accessed 7 September 2017].
17. ARM Information Center. 2017. *ARM Information Center*. [ONLINE] Available at: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0270b/Chdjgiai.html. [Accessed 7 September 2017].
18. ARM Assembly Language Programming - Chapter 3 - The Instruction Set. 2017. *ARM Assembly Language Programming - Chapter 3 - The Instruction Set*. [ONLINE] Available at: http://www.peter-cockerell.net/aalp/html/ch-3.html. [Accessed 7 September 2017].
19. ARM Information Center. 2017. *ARM Information Center*. [ONLINE] Available at: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0552a/BABFFEJF.html. [Accessed 7 September 2017].
20. Atmel AVR 8-bit and 32-bit Microcontrollers. 2017. *Atmel AVR 8-bit and 32-bit Microcontrollers*. [ONLINE] Available at: http://www.atmel.com/products/microcontrollers/avr/default.aspx. [Accessed 8 September 2017].
21. Kanda Electronics Blog. 2017. What is AVR microcontroller?. [ONLINE] Available at: https://www.kanda.com/blog/microcontrollers/avr-microcontrollers/avr-microcontroller/. [Accessed 10 September 2017].
22. ARM7 - Wikipedia. 2017. *ARM7 - Wikipedia*. [ONLINE] Available at: https://en.wikipedia.org/wiki/ARM7. [Accessed 8 September 2017].