

An Overview of the Instruction Set Architecture of ARM7TDMI

—— Assignment for COMP2121 Microprocessors and Interfacing

Author: Jingcheng Li
Student zID: z5109027
Date: 05/09/2017

Introduction

For the last decades, ARM has provided many kinds of RISC (Reduced Instruction Set Computer) architectures and solutions for different industries, such as mobile phones, laptops, automotive, arms and even super computers. While ARM family presents high performance so that the designers can integrate more functionalities into the product, it is rather small-sized, with significant low power consumption, which is very outstanding at that time. Finally, it has become the most widely used instruction set architecture as over 100 billion ARM processors were produced in 2017.

The ARM7TDMI core processor is one of the members of ARM's general-purpose 32-bit, and of course, it is the most widely used solution in the industry. Based on the Von Neumann architecture, the ARM7TDMI core processor contains the 32-bit ARM instruction set also with the 16-bit Thumb instruction set, which was a pathbreaking solution at that time. As far as we know, 16-bit architectures are usually higher code density and flexible (The size of Thumb code is normally 65% of ARM code), but since 32-bit architectures contain larger address space which present higher performance. So, the combination of the 32-bit ARM instruction set and the 16-bit Thumb instruction set provides both higher performance and higher code density. Users can switch between ARM state and Thumb state by using BX instruction, this gives them more options and flexibilities to balance the performance and space to meet their requirements.

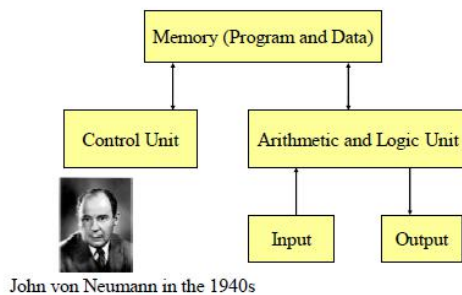
Like other microprocessors or microcontrollers, ARM7TDMI has conditional execution. It not only uses branches but also has its own mnemonic process. Also, ARM7TDMI has four unique condition code flags (Negative, Zero, Carry, and Overflow), they help with checking whether an instruction should be executed. We will discuss more about this later.

Compared with CISC (Complex Instruction Set Computer), RISC architectures have simpler and fewer instructions, also with lower cycles per instruction. This increases the speed and, as fewer transistors is required, saves much power and makes the products cheaper.

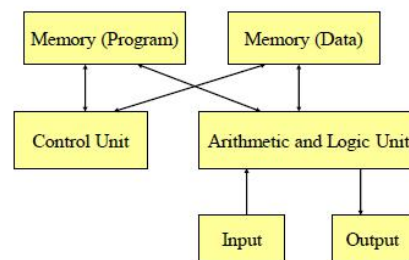
To make the instructions flow faster to the core processor, the ARM7TDMI core processor uses an instruction pipeline to achieve this. There are three stages where instructions are executed, fetch stage, decode stage and execute stage. Firstly, in the fetch stage, we get instructions from memory. Secondly, we try to decode the instruction. And in the execute stage, after doing the register fetch, we execute the operations and write the registers back. Unlike non-pipelined processors, which only execute one instruction at a time, the instruction pipeline help us to process one instruction while its successors are processed in previous stages at the same time. Working as a cache, the instruction pipeline makes sure that the instructions are executed continuously, which increase the processing speed. Another important thing is that the Program Counter always points to the instruction being fetched, which means the number it counts is always two more than the real result.

Memory models

As we know, there are two well-known architectures, Harvard Architecture and the Von Neumann Architecture. AVR uses Harvard Architecture, which program and data are stored separately, so we can access code and data at the same time. Since the ARM7TDMI core processor is based on the Von Neumann architecture, it uses a single memory bus which contains both instructions and data. Hence, we cannot do data transfer and operating instructions at the same time. The memory is a linear collection of bytes numbered in ascending order from zero, and the total space is 4 GB. And the size of the addressable cell is one byte. Unlike AVR microcontrollers, ARM7TDMI's 37 registers (31 general-purpose registers 6 status registers) are in the core. In the memory, to avoid corrupting neighboring bytes, users need to align data by the requirement. For example, words as 32-bit, need to be aligned to 4-byte boundaries, while halfwords as 16-bit, need to be aligned to 2-byte boundaries. As a flexible processor, the ARM7TDMI is bi-endian, which means data can be stored in either little-endian or big-endian in the memory. In ARM7TDMI, little-endian is the default mode.



1-1 The Von Neumann Architecture



1-2 Harvard Architecture
















Registers






The ARM7TDMI core processor contains 37 registers, with 31 general purpose 32-bit registers and 6 status registers (One CPSR register and five SPSRs). There are 16 registers, r0 to r15, that can be directly accessed at any time.

In the ARM state register set, registers r0 to r13, can hold either data or address values, while r14 and r15 have other special functions. Register r14 is known as the Link Register of the subroutine, which stores the address of the next instruction after a branch with link instruction. R14 receives value of r15 when we execute Branch with Link instruction. Also, when interrupts and exceptions arise, r14 stores the values of r15. R15, known as the Program Counter, is a pointer to the instruction that is being fetched, which is just two instructions after the instruction being executed. In ARM state, bits [31:1] contains the Program counter. Also, r13 is used as the Stack Pointer.

In the Thumb state register set, there are eight general registers, r0 to r7, along with registers for Stack Pointer, Link Pointer and Program Counter. SP, LP and PC in Thumb state map onto r13, r14, r15 in ARM state. However, CPSR and SPSRs of Thumb state and CPSR and SPSRs of ARM state are same.











Program status registers contains one current program status register and five saved program status register. The CPSR contains information about the current program, mode and processor, as well as the condition code flags. On the other hand, the SPSRs contain the value This contains the information of program status registers and condition code flags saved because of the entry to the current mode.






ARM-state general registers and program counter					
System and User	FIQ	Supervisor	Abort	IRQ	Undefined
r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7
r8	 r8_fiq	r8	r8	r8	r8
r9	 r9_fiq	r9	r9	r9	r9
r10	 r10_fiq	r10	r10	r10	r10
r11	 r11_fiq	r11	r11	r11	r11
r12	 r12_fiq	r12	r12	r12	r12
r13	 r13_fiq	 r13_svc	 r13_abt	 r13_irq	 r13_und
r14	 r14_fiq	 r14_svc	 r14_abt	 r14_irq	 r14_und
r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)	r15 (PC)

ARM-state program status registers					
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	 SPSR_fiq	 SPSR_svc	 SPSR_abt	 SPSR_irq	 SPSR_und

 = banked register

1-3 Register Organization in ARM Status

Thumb-state general registers and program counter					
System and User	FIQ	Supervisor	Abort	IRQ	Undefined
r0	r0	r0	r0	r0	r0
r1	r1	r1	r1	r1	r1
r2	r2	r2	r2	r2	r2
r3	r3	r3	r3	r3	r3
r4	r4	r4	r4	r4	r4
r5	r5	r5	r5	r5	r5
r6	r6	r6	r6	r6	r6
r7	r7	r7	r7	r7	r7
SP	 SP_fiq	 SP_svc	 SP_abt	 SP_irq	 SP_und
LR	 LR_fiq	 LR_svc	 LR_abt	 LR_irq	 LR_und
PC	PC	PC	PC	PC	PC

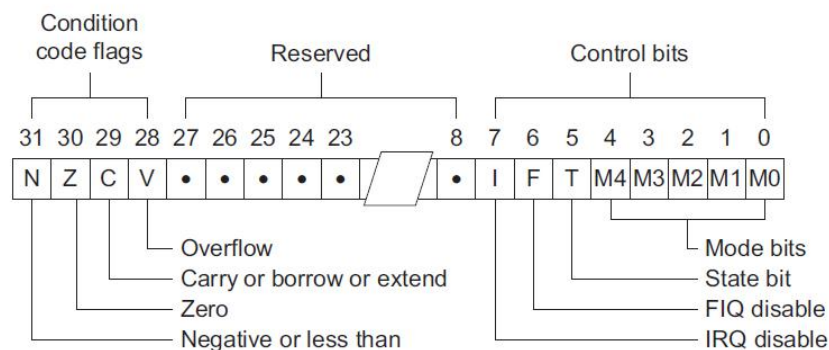
Thumb-state program status registers					
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	 SPSR_fiq	 SPSR_svc	 SPSR_abt	 SPSR_irq	 SPSR_und

 = banked register

1-4 Register Organization in Thumb Status

N, Z, C and V flags

In ARM7TDMI core processor, there are four condition code flags Negative (N), Zero (Z), Carry (C), and Overflow (V). Set by arithmetic and logical operations, or MSR and LMD instructions, containing the information of the latest ALU operation, they are used to test if an instruction can be executed. N is used to check if the first bit of a 32-bit result is one, if it is true, then the result is negative. Z refers that if it is set to one, then the result is zero. C is set to one if there was a carry-out during the ALU operation. To check if a signed overflow occurs, we need to test if V equals one. For signed comparison, there is a basic limitation among N and V, such as signed greater than or equal, we need to test if N equals V. Also for signed greater than, z needs to be equal to zero and N needs to be equal to V. While when we are doing unsigned comparison, for example unsigned higher, we need to check if C equals to one and Z equals to zero.



1-5 The Program Status Registers Format

Operating modes

You need to give detailed descriptions for each operating mode. Why are these operating modes needed? How does each operating mode work?

The least significant 5 bits are set for operating modes. There are seven modes of operation when we are using the ARM7TDMI core processor, they have an influence on the status of the registers. Different combinations of mode bits determine the operating modes. The seven modes are:

- 10000: User mode, the most frequently used program execution state.
- 10001: Fast Interrupt mode, supports data transfer and channel process, which could complete the work very fast.
- 10010: Interrupt mode, used to handle general purpose interrupts, like do not need very fast speed, such as clock ticks and screen VSync.
- 10011: Supervisor mode, an operating system protecting mode, would be entered after system reset.
- 10111: Abort mode, entered after failing to load instructions (Prefetch Abort) or data (Data Abort). Prefetched Abort means that the processor wants to load a failed

instruction, and Data Abort means that the processor wants to fetch inaccessible data.

- 11011: Undefined mode, operating system's privileged user mode, when we receive an undefined instruction, if the ARM7TDMI coprocessor does not respond, or the instruction is undefined, the mode is entered.
- 11111: System mode, an operating system's privileged user mode, you need to modify CPSR to enter the system mode.

Except User mode, the other modes are regarded as privileged modes, which are used to handle interrupts or get protected resources. Also, there is another concept called banked registers. In non-user modes, banked registers are mapped to some of the registers, holding contents of different operating modes. Normally, operating modes have only two banked registers, r13 and r14, but FIQ mode has seven banked registers, r8_fiq to r14_fiq.

Interrupts

In ARM7TDMI core processor, there are two kinds of interrupts, hardware interrupts caused by external events of peripherals and software interrupts. As we talked above, for hardware interrupts there are general purpose interrupts (IRQ) as well as interrupts that require fast response (FIQ). Designers can decide what interrupt instructions are produced by using control bits, if more specifically, interrupt disable bits.

There are three kinds of control bits, while mode bits are for operating modes and T bit is for operating states, the interrupt disable bits are for processing interrupts. Two bits I and F are set for enabling or disabling interrupts, if I bit is set, the processor will disable IRQ interrupts, and if F bit is set, then the processor will disable FIQ interrupts.

To handle hardware interrupts, the processor will need to change its mode based on the interrupt instructions received, by storing CPSR and the Program Counter into SPSR and the Link Register, the previous mode is saved. Then the processor changes the mode and branches to the handler. At last, the those saved information will be restored and the following instructions will be executed.

The Software Interrupt instruction, also known as SWI, is used to enter the Supervisor mode, usually to request a supervisor function. The software interrupts force the Program Counter to a specific value and cause the instruction pipeline to be refilled. At first, mode change happens, the return address and the CPSR is stored into the Link Register and the SPSR. Then the return address is modified to facilitate return. Finally, we complete the refilling of the instruction pipeline.

Instruction Set

Instruction formats

Instructions in the ARM7TDMI core processor are 32-bit, which contain arithmetic, logical operations, comparisons and data movements. The syntax consists of operations, condition field, control field, registers and Operand2, while sets condition codes, byte operation and halfword operation are optional. Operand2 means the second operand, which makes the instructions very flexible. The forces address translation, but we cannot use it with pre-index address. Just like its name, #32bit_imm is a 32-bit constant. If we want to use a list of registers, we could use <reglist> with comma-separated. In ARM7TDMI core processor, the result is stored into a specific register, instead of an initial set of registers or one of the operand registers. Also, since there are two operating states, the ARM state and the Thumb state, there is also instruction set for the Thumb state, whose instructions are 16-bit.

Addressing modes

Addressing modes, ways of determining the address, are procedures shared by many instructions, for generating values used by instructions. There is one addressing mode, Mode 1, which is a shifter of operands as values for data processing instructions. For the other four addressing modes, what they process is more traditional, the values they deal with are memory addresses. Mode 2 loads and stores word or unsigned byte. Mode 3 loads and stores halfword or load signed byte. Mode 4 loads and stores multiple. While Mode 5 loads and stores coprocessor. These addressing modes generate values in many different ways, like with or without offset and self-increment or self-decrement.

Conditional execution

As we mentioned above, instructions in ARM7TDMI can be execute conditionally, despite of using common branch or jump instructions, ARM7TDMI allows the use with most mnemonics. When we need to check the condition, we compare it with the current processor flags, and if it does not meet the requirement, the instruction will be regarded as a no-op, hence we do not need to do a conditional check and branch like in other architectures, which increases the speed of execution as well as the code density. All ARM instructions are conditionally executed, while only branch instruction is executed conditionally in Thumb state. There are four condition code flags (Negative, Zero, Carry, and Overflow) in ARM architecture, when the conditional execution is processed, these flags are checked and the instruction will be executed if it pass the test.

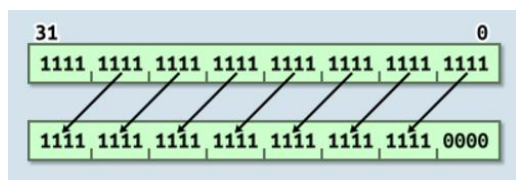
Stack operations

In AVR, normally we set up the stack frame from the top of data memory in a descending order. And use orders like push and pop to transfer the data between the stack frame and the registers. On the other hand, ARM7TDMI is much more powerful and provides many choices. While also works in the load and store operations, the use of stack pointer is more flexible. For example, there are four types of stack frame, combinations between full or empty and ascending or descending. If the it is a full stack, the stack pointer will point to the recently used variable, while if it is an empty stack, then the stack pointer will point to the first empty location.

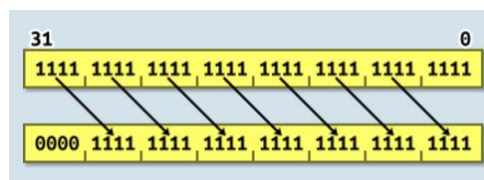
When we are doing push and pop operations, which also known as store and load, we execute instructions on the Stack Pointer. Initially, register r13 is regarded as the Stack Pointer. We can execute the stack operation together with CPSR or user registers.

Multiple-bit shift instructions

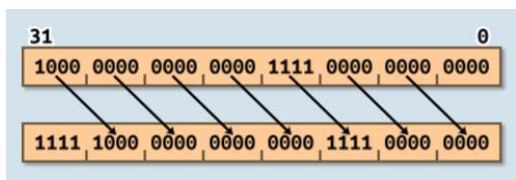
In ARM7TDMI, there is a special part named the Barrel Shifter, it is a functional unit which can be used in several different circumstances. It provides five types of shifts and rotates which can be applied to Operand2. The five operations are: logical shift left, logical shift right, arithmetic shift right, rotate right and rotate right extended. While the operations of logical shift left, logical shift right and rotate right are quite traditional, arithmetic shift right is a little bit tricky. The arithmetic shift right, also can be regarded as signed shift, is equivalent to division by a positive, integral power of the radix. It's same as the >> command in C. The rotate right extended operation is a 33-bit rotate with carry bit.



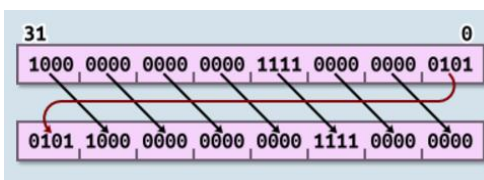
1-6 Logical Shift Left



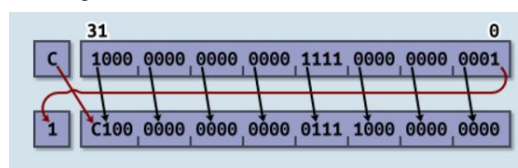
1-7 Logical Shift Right



1-8 Arithmetic Shift Right



1-9 Rotate Right



1-10 Rotate Right Extended

In and out

A microcontroller or processor needs to have the access to the I/O devices so that we could do the execution and read the result. In AVR we could use in and out to access I/O registers. However, in ARM7TDMI, there are no specific instructions for I/O, since it has memory mapped input and output and all peripherals are in memory map. Designers can directly do read and write operations like store and load, on the memory, instead of using other special instructions. It reduces the designers' work to get familiar with the instruction set.

Power saving and different frequencies

In many cases, we do not need the function to keep looping so that it only returns when it is required, which means it works in a low-power state. Normally we could temporally turn off the processor or let it into a sleep mode. In AVR microcontroller, we can use sleep to let the microcontroller enter the sleep mode. While in ARM7TDMI core processor, we could use Wait For Interrupt (WFI) instruction, hence the processor enters a low-power mode until an interrupt, an asynchronous abort or a debug event happens.

Since ARM7TDMI is a kind of architecture and anyone can make their own ARM7TDMI processors, designers could choose the clock frequencies by themselves to meet their needs, like any value up to 2.5GHz. Different frequencies provide more options to the designers so that they can implement their products more flexibly, for example, if you want faster speeds you can just have higher frequency.

Watchdog timer

As we know, watchdog timer is an electronic timer that is used to detect software crash and recover from computer malfunctions. In AVR microcontrollers, there is a watchdog timer and we can use the instruction wdr to reset the watchdog timer. ARM7TDMI core processor does not have similar instructions, but the function of DBGACK can be used to inhibit watchdog timers.

Data types

Unlike AVR which use 8-bit registers, the ARM7TDMI processor supports the following data types: 32-bit words, 16-bit halfwords and 8-bit bytes. If we want to use data types that is larger than the maximum limitation, we need to use multiple registers, it works in AVR, and it will be the same in ARM7TDMI. Then for 64-bit signed and unsigned integers, we need two registers to construct it. Like what we did in AVR, we need the carry bit to do the addition and subtraction. Assembly code for addition and subtraction are as followings:

- Addition:
 - ADDS R1, R3, R5 ($R1 \leftarrow R3 + R5$)
 - ADC R2, R4, R6 ($R2 \leftarrow R4 + R6$)
- Subtraction:
 - SUBS R1, R3, R5 ($R1 \leftarrow R3 - R5$)
 - SBC R2, R4, R6 ($R2 \leftarrow R4 - R6$)

Conclusion

8-bit AVR microcontroller is an inexpensive, low-power consumption device with very good platform support. Although its processing power and performance is a lot less than the ARM processors, it still works well for some simpler single systems. AVR microcontrollers have been used in various applications such as security systems, robots, experimenters and even Microsoft Xbox hand controllers.

As we mentioned above, ARM7TDMI processors provide matured and powerful solutions, the 32-bit processor with large memory presents high performance, with still very low-power consumption and cheap price. Since it gives designers flexible options, along with its significant salient features, it can be used in more applications especially those who require strong performance as well as power-saving attribute. As we mentioned, ARM7TDMI is well used in PDAs, mobile phones, automotive, modems and personal audio products. Some notable examples are: Nokia 6110, Dreamcast, Nintendo DS and Game Boys Advance.

With the rapid development of hardware devices and growing requirements of the users, more and more new and powerful ARM architecture processors are released, but ARM7TDMI processor is still a very classic one and performs as a vital character in the industry.

Reference list

1. (2012, January 16). Processor modes. Retrieved September 6, 2017, from https://www.heyrick.co.uk/armwiki/Processor_modes
2. Wikipedia. (2017, July 27). Arithmetic shift. Retrieved September 6, 2017, from https://en.wikipedia.org/wiki/Arithmetic_shift
3. Abdelrazek, A. (2006, July & Aug.). Exception and Interrupt Handling in ARM (Rep. No. 1). Retrieved September 5, 2017, from website.
4. Bramley, J. (2013, September 11). Condition Codes 1: Condition flags and codes. Retrieved September 09, 2017, from <https://community.arm.com/processors/b/blog/posts/condition-codes-1-condition-flags-and-codes>
5. Thomas, D. (2012, March 3). Introduction to ARM: Barrel Shifter. Retrieved September 6, 2017, from <http://www.davespace.co.uk/arm/introduction-to-arm/barrel-shifter.html>

6. ARM. (n.d.). ARM7TDMI Core Processor Product Overview. Retrieved September 4, 2017, from <http://infocenter.arm.com/help/index.jsp?topic=%2Fcom.arm.doc.ddi0210c%2FI1040101.html>
7. ARM. (n.d.). Load and store instructions. Retrieved September 4, 2017, from <http://infocenter.arm.com/help/index.jsp?topic=%2Fcom.arm.doc.dvi0027b%2Far01s04s10.html>
8. Wu, H. (2017, July). COMP2121 Microprocessors and Interfacing Notes. Retrieved September 2, 2017, from WebCMS3
9. Wikipedia. (2017, July 27). ARM7. Retrieved September 3, 2017, from <https://en.wikipedia.org/wiki/ARM7>