ARM Assembly Language

10-15-2015

1. First Concepts

1.1 Machine code

-Assembly language and assemblers

ON means 10100111, ON is easier to remember.

A program written using these textual names for instructions is called an assembly language program.

The set of mnemonics that is used to represent a computer's machine code is called the assembly language of that computer.

Assembly language is the lowest level used by humans to program a computer.

The process of using an assembler to convert from mnemonics to machine code is called assembling.

-compilers and interpreters

A compiler, is similar to an assembler in that it converts from a human-readable program into something a computer can understand.

An alternative approach is provided by another technique used to make the transition from high-level language to machine code. This technique is know as interpreting. The most popular interpreted language is BASIC.

-Summary

Computers understand (respond to) the presence or absence of voltages. we can represent these voltages on paper by sequences of 1s and 0s (bits). The set of bit sequences which cause the computer to respond in some well-defined way is called its machine code. Humans can't tell 10110111 from 10010111 very well, so we give short names, or mnemonics, to instructions. The set of mnemonics is the assembly language of the computer, and an assembler is a program to convert from this representation to the computer-readable machine code. A compiler does a similar job for high-level languages.

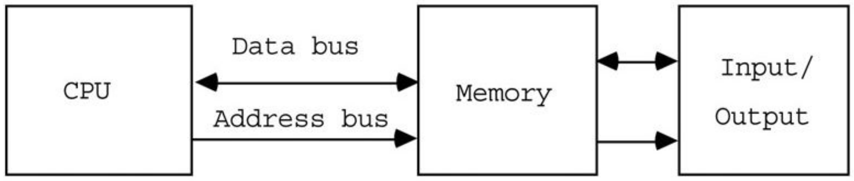
1.2 Computer architecture

-The CPU

central processing unit (CPU)

one particular type of CPU-the Acorn RISC Machine or ARM

- Computer busses



Busses are said to have certain widths.

For a given processor the width of the data bus is usually fixed, typical values are 8,16 and 32 bits.

On the ARM the data bus is 32 bits wide (i.e. there are 32 separate signals for transferring data), and the ARM is called a 32-bit machine.

The wider the data bus, the larger the amount of information that can be processed in one go by the CPU.

The ARM's address bus has 26 signals. The wider the address bus, the more memory the computer is capable of using.

For each extra signal, the amount of memory possible is doubled.

The ARM has 84 signals. 58 of these are used by the data and address busses, the remainder form yet another bus, is called the control signal bus, and groups together the signals required to perform tasks such as synchronising the flow of information between the ARM and other devices.

- **Memory and I/O**

Whatever the details of the computer's I/O, the CPU interacts with it through the data bus.

In fact, to many CPUs (the ARM being one) I/O devices 'look' like normal memory, this is called memory-mapped I/O.

A computer's memory (and I/O) may be regard as a collection of cells, each of which may contain n bits of information, where n is the width of the data bus.

The function of the address bus is to provide a code which uniquely identifies the desired cell.

The ARM has a 26-bit address bus, which allows (2^26)64 million cells (or 'locations') to be addressed.

- **Instructions**

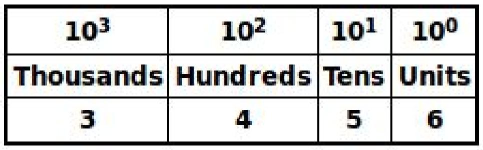
The fetch-decode-execute cycle may be performed so quickly that makes computers fast. The ARM, for example, can manage a peak of 8,000,000 cycles a second.

- **Summary**

The ARM, in common with most other CPUs, is connected to memory and I/O devices through the data bus and address bus. Memory is used to store instructions and data, I/O is used to interface the CPU to the outside world. Instructions are fetched in a normally sequential fashion, and executed by the CPU. The ARM has a 32-bit data bus, which means it usually deals with data of this size. There are 26 address signals, enabling the ARM to address 64 million memory or I/O locations.

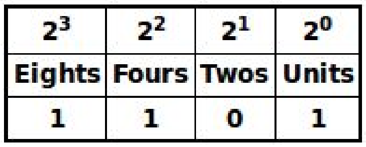
1.3 Bits, bytes and binary

All data and instructions in computers are stored as sequences of ones and zeros.



10^3	10^2	10^1	10^0
Thousands	Hundreds	Tens	Units
3	4	5	6

$3000 + 400 + 50 + 6 = 3456$



2^3	2^2	2^1	2^0
Eights	Fours	Twos	Units
1	1	0	1

$8 + 4 + 0 + 1 = 13$

In the ARM environment, 223 of possible codes are used to represent characters you can see on the screen. 95 of these are the usual symbols you see on the keyboard, e.g. the letters, digits and punctuation characters. Another 128 are special characters, e,g, accented letters and maths symbols. The remaining 33 are not used to represent printed characters, but have special meanings.

- **Binary arithmetic**

The rules for adding two decimal digits are:

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 0$ carry 1

	0	1	0	1
+	1	0	0	1
c			1	
	1	1	1	0

$= 8 + 4 + 2 = 14$

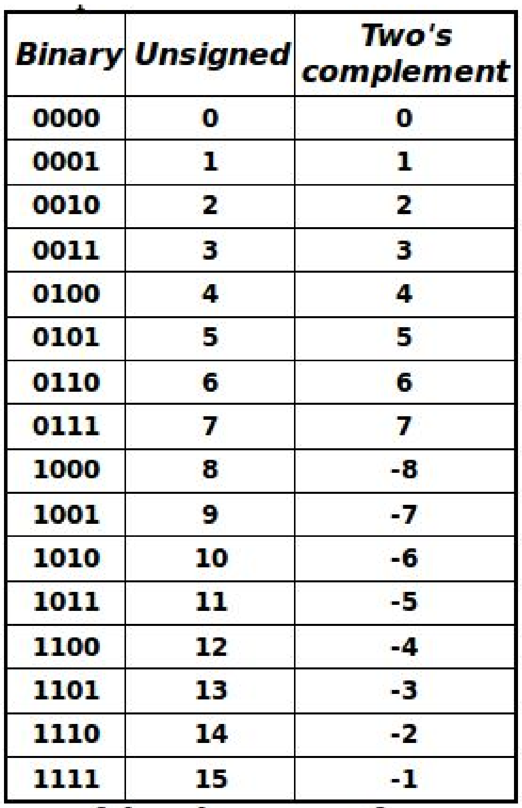
Binary subtraction is defined in a similar way:

$0 - 0 = 0$
$0 - 1 = 1$ borrow 1
$1 - 0 = 1$
$1 - 1 = 0$

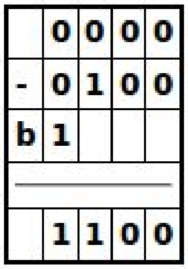
	1	0	0	1
-	0	1	0	1
b	1			
	0	1	0	0

$= 4$

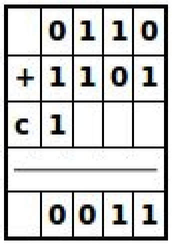
The most common way of representing a negative number is to use ‘two’s complement’ notation. we obtain for a number -n simply by performing the subtraction 0-n.



Binary	Unsigned	Two's complement
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	-8
1001	9	-7
1010	10	-6
1011	11	-5
1100	12	-4
1101	13	-3
1110	14	-2
1111	15	-1



	0	0	0	0
-	0	1	0	0
b	1			
	1	1	0	0



	0	1	1	0
+	1	1	0	1
c	1			
	0	0	1	1

to add 6 and -3, we would use:

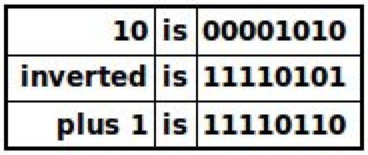
LSB: the rightmost bit represents two to the zero, and so is called bit zero. also called the least significant bit (LSB).

MSB: the highest bit of a N-bit number will be bit N-1, and naturally enough, called the most significant bit (MSB).

One way is to use one bit (usually the MSB) to represent the sign -0 for positive and 1 for negative.

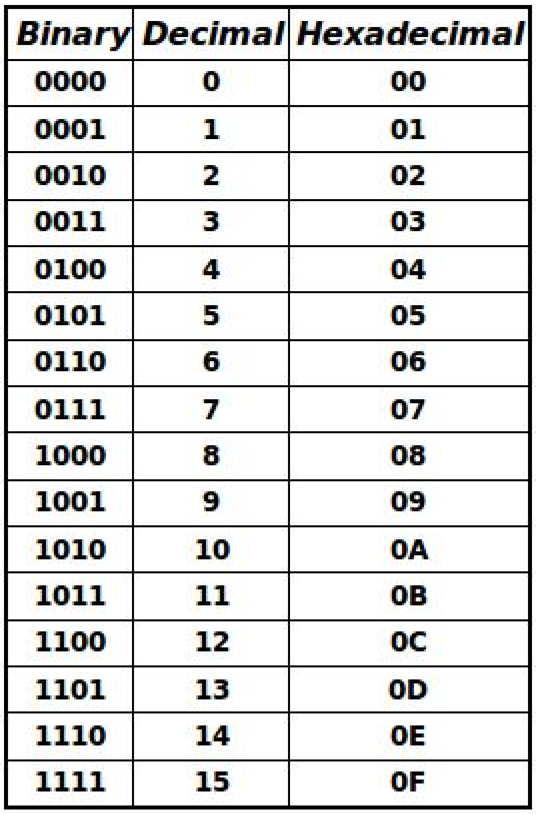
The number -1 is always ‘all ones’, i.e. 1111 in a four-bit system, 11111111 in eight bits etc.

To find the negative version of a number n, invert all of its bits (i.e. make all the 1s into 0s and vice versa) and add 1. For example, to find -10 in an eight-bit two’s complement form:



10	is	00001010
inverted	is	11110101
plus 1	is	11110110

- **Hexadecimal**



Binary	Decimal	Hexadecimal
0000	0	00
0001	1	01
0010	2	02
0011	3	03
0100	4	04
0101	5	05
0110	6	06
0111	7	07
1000	8	08
1001	9	09
1010	10	0A
1011	11	0B
1100	12	0C
1101	13	0D
1110	14	0E
1111	15	0F

The hex number &D9F is 13*16^2+9*16+15 or 3487

It is very easy to convert between hex and binary representation. Each hexadecimal digit is formed from four binary digits grouped from the left. For example:

11010110 = 1101 0110 = D 6 = &D6

11110110 = 1111 0110 = F 6 = &F6

Whether a given hex number represents a negative quantity is a matter of interpretation. For example, the byte &FE may represent 254 or -2, depending on how we wish to interpret it.

- **Large numbers**

The letter K after a number means ‘Kilo’ or ‘times 1024’. mega (abbr.M) is used to represent 1024*1024 bytes or just over one million.

- Memory and addresses

The memory of ARM is organised as bytes. Each byte has its own address, starting from 0. This is 26 bits, so the highest address is 11111111111111111111111111 or 3FFFFFF or 67,108,863. This enables the ARM to access 64M bytes of memory.

The ARM deals with data in 32-bit or four-byte units. Each such unit is called a word (and 32-bits is the word-length of the ARM).

The ARM is a byte-addressable machine, because every single byte in memory has its own address, in the sequence 0,1,2, and so on.

When complete words are accessed (e.g. when loading an instruction), the ARM requires a word-aligned address, that is, one which is a multiple of four bytes. So the first complete word is at address 0, the second at address 4, and so on.

The way in which each word is used depends entirely on the whim of the programmer. For example, a given word could be used to hold an instruction, four characters, or a single 32-bit number, or 32 one-bit numbers. It may even be used to store the address of another word. The ARM does not any interpretation on the contents of memory, only the programmer does.

When multiple bytes are used to store large numbers, there are two ways in which the bytes may be organised. The more common way- used by the ARM- is to store the bytes in order of increasing significance. For example, a 32-bit number stored at addresses 8..11 will have, bits 0..7 at address 8, bits 8..15 at address 9, bits 16..23 at address 10, and bits 24..31 at address 11.

If two consecutive words are used to store a 64-bit number, the first word would contain bits 0..31 and the second word bits 32..63.

There are two main types of memory. The programs you will write and the data associated with them are stored in read/write memory is RAM. This comes from the somewhat misleading term Random Access Memory. All memory used by ARMs is Random Access, whether it is read/write or not, but RAM is universally accepted to mean read/write.

RAM is generally volatile, its contents are forgotten when the power is removed. Most machines machines provide a small amount of non-volatile memory (powered by a rechargeable battery when the mains is switched off) to store information which is only changed very rarely, e.g. preferences about the keyboard auto-repeat rate.

The other type of memory is ROM -Read-only memory. This is used to store instructions and data which must not be erased, even when the power is removed. For example the program which is obeyed when the ARM is first turned on is held in ROM.

1.4 Inside the CPU

The data bus is used to transfer data and instructions between the CPU and memory or I/O.The address contains the address of the current location being accessed.

There are many other signals emanating from CPU. Examples of such signals on the ARM are r/w which tells the outside world whether the CPU is reading or writing data; b/w which indicates whether a data transfer is to operate on just one byte or a whole word; and two signals which indicate which of four possible 'modes' the ARM is in.

If we could examine the circuitry of the processor we would see thousands of transistors, connected to form common logical circuits. These go by names such as NAND gate, flip-flop, barrel shifter and arithmetic-logic unit (ALU).

What interests us is the way all of these combine to form an abstract model whose behavior we can control by writing programs. This is called the 'programmers' model'.

- **The instruction cycle**

Inside the CPU is a 24-bit store that acts as a counter. On reset, it is set to &000000. The counter holds the address of next instruction to be fetched. It is called the program counter(PC). When the processor is ready to read the next instruction from memory, it places the contents of the PC on to the address bus. In particular, the PC is placed on bits 2..25 of the address bus.

Bits 0 and 1 are always 0 when the CPU fetches an instruction, as instructions are always on word address, i.e. multiples of four bytes.

The CPU also outputs signals telling the memory that this is a read operation, and that it requires a whole word (as opposed to a single byte). The memory system responds to these signals by placing the contents of the addressed cell on to the data bus, where it can be read by the processor. Remember that the data bus is 32 bits wide, so an instruction can be read in one read operation.

From the data bus, the instruction is transferred into the first stage of a three-stage area inside the CPU. This is called the pipeline, and at any time it can hold three instructions: the one just fetched, and the one being executed. After an instruction has finished executing, the pipeline is shifted up one place, so the just-decoded instruction starts to be executed, the previously fetched instruction starts to be decoded, and the next instruction is fetched from memory.

Decoding the instruction involves deciding exactly what needs to be done, and preparing parts of the CPU for this. For example, if the instruction is an addition, the two numbers to be added will be obtained.

When an instruction reaches the execute stage of the pipeline, the appropriate actions take place, a subtraction for example, and the next instruction, which has already been decoded, is executed. Also, the PC is incremented to allow the next instruction to be fetched.

In some circumstances, it is not possible to execute the next pipelined instruction because of the effect of the last one. Some instructions explicitly alter the value of the PC, causing the program to jump (like a GOTO in BASIC). When this occurs, the pre-fetched instruction is not the correct one to execute, and the pipeline has to be flushed (emptied), and the fetch-decode-cycle started from the new location. Flushing the pipeline tends to slow down execution (because the fetch, decode and execute cycles no longer all happen at the same time) so the ARM provides ways of avoiding many of the jumps.

- **The ALU and barrel shifter**

Many ARM instructions make use of these two very important parts of the CPU. There is a whole class of instructions, called the data manipulation group, which use these units. The arithmetic-logic unit performs operations such as addition, subtraction and comparison. These are the arithmetic operations. Logical operations include AND, EOR and OR.

The ALU can be regarded as a black-box which takes two 32-bit numbers as input, and produces a 32-bit result. The instruction decode circuitry tells the ALU which of its repertoire of operations to perform by examining the instruction. It also works out where to find the two input numbers- the operands -and where to put the result from the instruction.

The barrel shifter has two inputs - a 32-bit word to be shifted and a count -and one output -another 32-bit word. As its name implies, the barrel shifter obtains its output by shifting the bits of the operand in some way. There are several flavours of shift: which direction the bits are shifted in, whether the bits coming out of one end re-appear in the other end etc.

The important property of the barrel shifter is that no matter what type of shift it does, and how many bits, it always takes only one ‘tick’ of the CPU’s master clock to do it. This is much better than many 16 and 32-bit processors, which take a time proportional to number of shifts required.

- **Registers**

An important part of the CPU is the register bank. In fact, from the programmer's point of view, the registers are more important than other components such as the ALU, as they are what he actually 'sees' when writing programs.

A register is a word of storage, like a memory location. On the ARM, all registers are one word long, i.e. 32 bits. There are several important differences between memory and registers. Firstly, registers are not 'memory mapped', that is they don't have 26-bit addresses like the rest of storage and I/O on the ARM.

Because registers are on the CPU chip rather than part of an external memory system, the CPU can access their contents very quickly. In fact, almost all operations on the ARM involve the use of registers. For example, the ADD instruction adds two 32-bit numbers to produce a 32-bit result. Both of the numbers to be added, and the destination of the result, are specified as ARM registers. Many CPUs also have instructions to, for example, add a number stored in memory to a register. This is not the case on the ARM, and the only register-memory operations are load and store ones.

The second difference is that there are far fewer registers than memory locations. The ARM can address up to 64M bytes (16M words) of external memory. Internally, there are only 16 registers visible at once. These are referred to in programs as R0 to 15. A couple of the registers are sometimes given special names, for example R15 is also called PC, because it holds the program counter value.

There being no distinction for example between R0 and R12. This availability of a large (compared to many CPUs) number of rapidly accessible registers contributes to the ARM's reputation as a fast processor.

1.5 A small program

On the left is a listing of simple BASIC FOR loop which prints 20 stars on the screen. On the right is the ARM assembly language program which performs the same task.

BASIC	***ARM Assembly Language***
`10 i=1`	`MOV R0,#1 ;Initialise count`
`20 PRINT "*";`	`.loop SWI writeI+"*" ;Print a *`
`30 i=i+1`	`ADD R0,R0,#1 ;Increment count`
`40 IF i<=20 THEN 20`	`CMP R0,#20 ;Compare with limit`
	`BLE loop`

Most of the ARM instructions should be self-explanatory. The word loop marks the place in the program which is used by the BLE (branch if less than or equal to) instruction. It is called a label, and fulfills a similar function to the line number in a BASIC instruction such as GOTO 20.

One thing you will notice about the ARM program is that it is a line longer than BASIC one. This is because in general, a single ARM instruction does less processing than a BASIC one. A program written in assembler will occupy more lines than an equivalent one written in BASIC or some other high-level language.

When assembled, the ARM program above will occupy five words (one per instruction) or 20 bytes. The BASIC program takes 50 bytes, so the size of the assembly language program (the 'source') can be misleading.

2. Inside the ARM

Some important attributes of the CPU:

Word size
The ARM's word length is 4 bytes. It's a 32-bit micro.

Memory
ARM uses a 26-bit address value. This allows for 2^26 or 64M bytes of memory to be accessed. ARM is really word-based. All word-sized transfers must have the operands in memory residing on word-boundaries. This means the instruction addresses have to be multiples of four.

I/O
Input and output devices are memory mapped. There is no concept of a separate I/O address space. Peripheral chips are read and written as if they were areas of memory. This means that in practical ARM systems, the memory map is divided into three areas: RAM, ROM, and input/output devices (probably in decreasing order of size).

Registers
ARM has sixteen 32-bit registers which may be used without restriction in any instruction. There is very little dedication - only one of the registers being permanently tied up by the processor.

Instructions
A small, easily remembered set of instruction types is available.

2.1 Memory and addressing

It is very unlikely that this much memory will actually be fitted in current machines, even with the ever-increasing capacities of RAM and ROM chips. One or four megabytes of RAM is a reasonable amount to expect using today's technology.