

Summary Micro-Controller FS18

Alex Neher

June 30, 2018

Contents

1	Microcontroller	3
1.1	Components	3
1.2	Numerical systems	3
1.2.1	Tow's Complement	4
1.3	Logic Gates	4
1.4	Instruction Set Cycle	5
2	C	6

List of Figures

1.1	Von-Neumann Architecture	3
1.2	Fundamental logic-Gates used in MCs	4
1.3	Visualisation of the (de)multiplexer and the flipflop	5
1.4	Visualisation of the Instruction Set Cycle	5

1 Microcontroller

1.1 Components

Micro controllers are so called **Single Chip** Computer, meaning everything is on a single PCB, as opposed to e.g. a 'normal' PC.

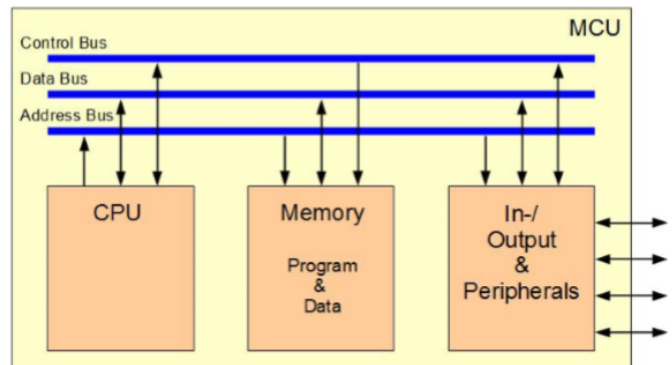
MC consist of at least four components:

CPU: Central Processing Unit

Memory: Where programs and data are stored

IO/Input-Output: Communication with Peripherals

Bus-System: Connects the components



There are two different architectures:

Fig.. 1.1: Von-Neumann Architecture

Von-Neumann: One shared bus for program and data. Program and data are in the same memory. Often found in low-cost MCs

Harvard: Two separate bus systems for program and data. Often found in high-performance MCs

Usually, a read/write-operation goes through four steps:

1. CPU puts the address on the address bus
2. Either the memory or the IO claim the address as their
3. CPU tells the component via the control bus whether the operation is read or write
4. *read:* The memory or IO places the data of the requested address on the data bus
write: The CPU writes the data on the mentioned address via data bus

1.2 Numerical systems

In MC, variables and constants are seldom stored as decimal value. Rather they're either stored as a binary or a hexadecimal value.

In general, mathematical terms an n -digit integer to the base B can be expressed as:

$$\sum_{i=0}^{n-1} x_i * B^i = X_0 * B^0 + x_1 * B^2 + [...] + x_{n-1} * B^{n-1}$$

Or easier: *Multiply the n -th digit of the integer with B^{n-1} starting from the right with $n = 0$*

Example

1100'0101₂ (binary) to decimal

$$\begin{aligned} &= 1 * 2^0 + 0 * 2^1 + 1 * 2^2 + 0 * 2^3 + 0 * 2^4 + 0 * 2^5 + 1 * 2^6 + 1 * 2^7 \\ &= 1 + 0 + 4 + 0 + 0 + 0 + 64 + 128 \\ &= \underline{\underline{197_{10}}} \end{aligned}$$

If you have to convert a number between two 'exotic' systems, say base 8 to base 3, it's usually easier to convert it to decimal first and then convert it to the desired system again ($x_8 \rightarrow x_{10} \rightarrow x_3$). An exception to that is binary to hexadecimal and vice versa. One digit in hexadecimal represents four digits in binary, so you can directly convert blocks of four:

$1100'0101_2$ to hexadecimal:

$$1100 = 0 * 2^0 + 0 * 2^1 + 1 * 2^2 + 1 * 2^3 = 0 + 0 + 4 + 8 = 12_{10} = C_{16}$$

$$0101 = 1 * 2^0 + 0 * 2^1 + 1 * 2^2 + 0 * 2^3 = 1 + 0 + 4 + 0 = 5_{10} = 5_{16}$$

$$\rightarrow 1100'0101_2 = C5_{16}$$

1.2.1 Tow's Complement

Especially in MC-technology, signed numbers (that can also be negative) are mostly stored as *two's complement*. You basically take the binary number, invert every digit and add one. So -28 would be stored as

$$28_{10} = 16 + 8 + 4 = 2^2 + 2^3 + 2^4 = 0001'1100_2$$

invert

$$0001'1100 \rightarrow 1110'0011$$

add one

$$1110'0011 + 1 = \underline{\underline{1110'0100}}$$

1.3 Logic Gates

Logic Gates																			
Function	Symbol	Equation	Table	Switch Setup															
AND		$Q = a \cdot b$	<table><tr><th>b</th><th>a</th><th>Q</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	b	a	Q	0	0	0	0	1	0	1	0	0	1	1	1	
b	a	Q																	
0	0	0																	
0	1	0																	
1	0	0																	
1	1	1																	
OR		$Q = a + b$	<table><tr><th>b</th><th>a</th><th>Q</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	b	a	Q	0	0	0	0	1	1	1	0	1	1	1	1	
b	a	Q																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	1																	
NAND		$Q = \overline{a \cdot b}$	<table><tr><th>b</th><th>a</th><th>Q</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	b	a	Q	0	0	1	0	1	1	1	0	1	1	1	0	
b	a	Q																	
0	0	1																	
0	1	1																	
1	0	1																	
1	1	0																	
NOR		$Q = \overline{a + b}$	<table><tr><th>b</th><th>a</th><th>Q</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	b	a	Q	0	0	1	0	1	0	1	0	0	1	1	0	
b	a	Q																	
0	0	1																	
0	1	0																	
1	0	0																	
1	1	0																	
NOT		$Q = \overline{a}$	<table><tr><th>a</th><th>Q</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	a	Q	0	1	1	0										
a	Q																		
0	1																		
1	0																		
XOR		$Q = \overline{a} \cdot b + a \cdot \overline{b}$ $Q = a \oplus b$	<table><tr><th>b</th><th>a</th><th>Q</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	b	a	Q	0	0	0	0	1	1	1	0	1	1	1	0	
b	a	Q																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	0																	
XNOR		$Q = a \cdot b + \overline{a} \cdot \overline{b}$ $Q = \overline{a \oplus b}$	<table><tr><th>b</th><th>a</th><th>Q</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	b	a	Q	0	0	1	0	1	0	1	0	0	1	1	1	
b	a	Q																	
0	0	1																	
0	1	0																	
1	0	0																	
1	1	1																	

Fig.. 1.2: Fundamental logic-Gates used in MCs

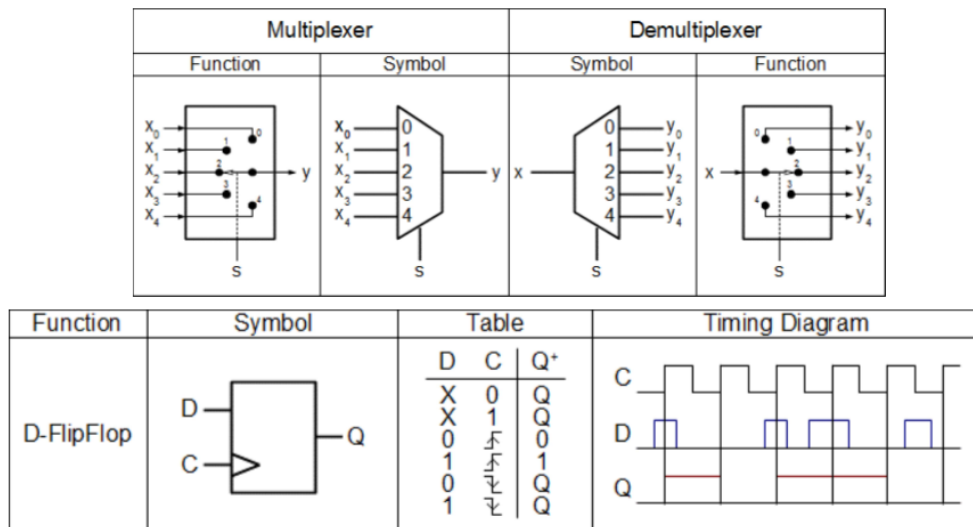


Fig.. 1.3: Visualisation of the (de)multiplexer and the flipflop

Multiplexer: The multiplexer is a combinational logic circuit, which allows us to select one of many input lines and route it to the single, common output line. The demultiplier does the exact opposite: it takes one input and you can select to which output line it is routed

FlipFlop: Idunno

1.4 Instruction Set Cycle

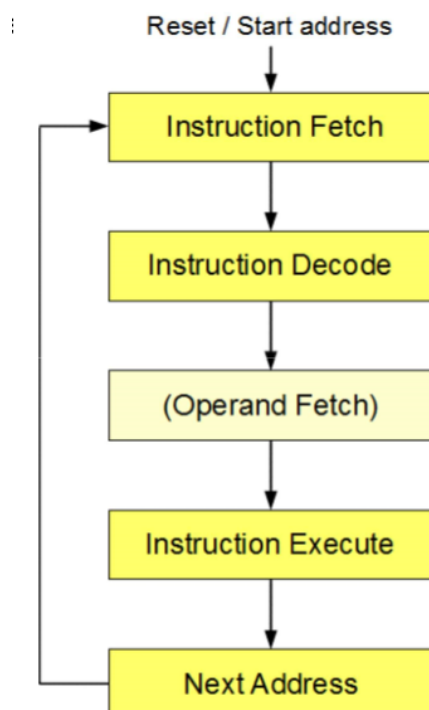


Fig.. 1.4: Visualisation of the Instruction Set Cycle

The way a CPU executes instructions can be shortened to **FDE**. It stands for **F**etch, **D**ecode **E**xecute. S As can be seen in fig. 1.4,

the CPU first fetches the instruction from the memory, then it decodes it and decides if it has to fetch a second operand (e.g. for an addition). Afterwards it executes said instruction and moves on to the next address.

2 C