The Atari ST M68000 tutorial part 1 – on the theory behind programming

Hi everybody! This is perihelion of poSTmortem (aka Andreas Wahlin) writing. Me and Aldebaran (aka Lars Lindblad) have just started this little project of ours; the demo group poSTmortem. As a first step towards actually doing something with the ST, we thought that writing a tutorial might be good. In this way, we can teach ourselves and you (whoever is listening).

This tutorial is aimed at people who, like ourselves, want to learn to code assembler for the Atari ST. What? You say, this is the year 2002, why on earth would you want to learn how to code, for one thing, assembler, and assembler for the Atari? You must be crazy. Well, you might think the Atari is dead, but we say it survived itself, it has risen again, it is; poSTmortem (totally lame, right?). You don't need any programming skills, although it might help since learning to code from assembly language is probably quite suicidal in a pedagogical view. I'll try to cover the basics of general programming and setting you up in this tutorial, and in part two we will do some rather simple program to get things started. I will however, assume that you have some basic skills in Atari management, like file copying and so on. Also, I will assume you have a real Atari, emulating might work fine, but nothing beats the real thing.

What is programming? Programming is the art and science of making the computer do what you want it to do. BTW, programming is also known as coding, I will use these two terms somewhat mixed probably. So, how do we do that? By telling the computer what to do, and then do it. Since you don't have shit for brains, you will know that when you double click on a .prg file (known as .exe files on a PC), the computer will do stuff, like running a game. So, what if we could create our own .prg files… Yes, we can do that, this is where you'll learn how!

Every computer has a memory, this is where the number on your Atari is derived from, 520 have half a Meg (short for Mega Byte) of ram, and 1040's have one Meg. Your usual PC these days have about 256 megs of ram, Bill Gates once said "nobody needs more than 640 K ram (about the memory amount of an Atari 520). If these numbers confuse you, do not worry, they aren't important right now. The memory is very temporary; it gets wiped out every time you turn off your Atari, unlike diskettes (thank God). When you run a program, the computer loads the program into memory, and then executes (follows the instructions given by the program).

Every area of the memory has an address, so you know where you are. You can think of these addresses as normal street addresses, but perhaps it would be better if you thought of them more as page indexes in a book. Every page is filled with information on how to act. So, let's assume that we have a monitor capable of displaying one of two colours, either black or white. The memory address $20 holds this colour. 1 means black, 0 white. If memory looks like this, we get a black screen.

Address	Value
…	

$19 12
$20 1
$21 67
…
(each address can hold a number between 0 and 255)

To understand a bit better about memory, because this is very important, we expand our little example. There is an area in the memory reserved for the user. This area is just for information storage, and does not affect the hardware. Let's say that the monitor is capable of displaying text as well, the text is also either black or white as previously stated, and the information on the text colour is to be found at $21. Further, the address $22 holds a pointer to the text to be displayed. Pointer? Argh! Well, it's really quite easy, a pointer is a reference to another part of the memory. Show first, talk later.

Memory Value
…
$19 12
$20 1
$21 0
$22 101
…
$101 Hello World!
$200 DOH!
…

In this example, the "user memory" begins after position $100. All address positions $0 - $100 does something with the hardware in some way, but after that, it's just storage space. With the above values, and given our premise, the text "Hello World!", will be displayed in white text on black background. Let's say we were to change the values to this instead.

Memory Value
…
$19 12
$20 1
$21 1
$22 200
…
$101 Hello World!
$200 DOH!
…
(changed values at $21 and $22)

We would get the text "DOH!" written in black text on black background, not to clever. Therefore, a pointer is a reference to another place.

Because $22 only can hold a number of 0 - 255, the text “Hello World!” would never fit, so instead we point to an area in the user memory, which can hold much more than just a number between 0 - 255. Are you beginning to grasp how computers work?

If you have an enquiring mind, and I hope you do, you’ll probably wonder why the addresses $0 - $100 only hold numbers, while it seems that addresses $101 - … can hold letters. The answer is, they actually can’t hold any letters. In addition, it doesn’t quite look like I’ve shown you either. This might get somewhat complicated, hold your hat and don’t cry if you don’t get it. Just read it, let it go, meditate a bit and you’ll reach enlightenment.

The addresses $101 and $200 actually only hold numbers, but the computer have a decode key so that each number can be decoded to a letter. Let’s say that

! = 0
A = 1
B = 2
C = 3
D = 4
…
and so on, then it really looks like this

Memory	Value	Meaning
$200	3	D
$201	14	O
$202	7	H
$203	0	!

…

So you see, there are just numbers, also, as I said, each address can only hold a number between 0 - 255, meaning that each letter is held in one address. But but but but, why doesn’t more stuff get displayed as text? Why does it stop at $203? The memory must continue after that surely. Let’s look at this memory setup.

Memory	Value	Meaning
$200	4	D
$201	15	O
$202	8	H
$203	0	!
$204	15	O
$205	4	D

…

The text on the screen would display as “DOH!OD” and probably much more (the rest of the memory in fact). Well, here we use a control number,

let's say that the computer knows that when in reaches the number 255 the text ends there. If memory looks like this

Memory	Value	Meaning
…		
$19	12	something
$20	1	background colour
$21	0	text colour
$22	200	pointer to text on screen
…		
$200	4	D
$201	15	O
$202	8	H
$203	0	!
$204	255	end of text
$205	4	D

the text "DOH!" would be displayed in white on black background. When the computer reaches $204, it sees the number 255, which means stop displaying text, so the letter (or rather, value) at $205 and following addresses will not be displayed. Like I said, this may be a bit advanced, don't panic. We will get much more concrete in tutorial 2. I just want you to have a theoretical basis so you know what's what and so you can refer back to this. Just let this sink into your unconscious, when the time is right and you have correct understanding, it will surface and you will get it.

Now, for last theory lesson; how do you actually make something happen? As we know, there are .prg files that make stuff happen. With our above knowledge, we know that they affect memory. We can write down simple commands in a text file, and then have that text file translated into the .prg format, so that the computer will understand what we say. A program that can pull this off is known as a compiler, a compiler usually comes with a text editor, suited for programming needs. The text file you use to create a .prg file, is knows as the source code. Let's take another example, this time let's assume we wrote this source code.

```
Put #1 at $20
Put #0 at $21
Put #200 at $22
Put #4 at $200
Put #15 at $201
Put #8 at $202
Put #0 at $203
Put #255 at $204
Initialize monitor
```

Now, as you can guess, # stands for value, a numerical value in our

case, and $ stands for address. Now, if we compile this source code, that is, translate it to a format the computer understands, we will get a .prg file. When we double click on that file, the computer will do what it says above; the different values will be loaded into the different addresses, creating the memory profile given above. The last line “Initialize monitor” is for engaging the monitor. When the monitor is engaged, the Atari knows that it should look at $20, $21 and $22 to gather the data needed. So instead of “Initialize monitor”, perhaps we could’ve written

Activate $20
Activate $21
Activate $22

Because what we really want to do is to make the information on these addresses happen; we want the computer to process the information given. This is long and clumsy however, and the line “Initialize monitor”, or whatever you might call it, is far simpler.

The computer, internally, understands nothing but 1’s and 0’s, all text and numbers I have given above is for human understanding (more on binary understanding later). Also, none of the commands or memory addresses have any significance for the Atari, they are examples only.

OK, theory lesson over. Hope I haven’t scared you away. In the next tutorial we will get into how to actually make a .prg file. It won’t do much, but at least you will get to see your code in action.

perihelion of poSTmortem, 2002-03-31

“Conan, what is best in life?”
“To crush your enemies, see them driven before you, and to hear the lamentations of their women.”
- Conan the Barbarian

Last edited 2002-04-07