If this is your first visit, be sure to check out the **FAQ** by clicking the link above. You may have to **register** before you can post: click the register link above to proceed. To start viewing messages, select the forum that you want to visit from the selection below.

If you'd like to receive bonuses in the Official Wiremod Gmod Server, Link your Steam account to you forum account **here!**

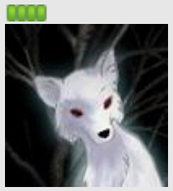
Thread: The "Foxy" CPU - Part 2

12-27-2008 #1

-=Fox=-

Wire Sofaking

WIREMOD MEMBER



Join Date: Feb 2007
Location: Somewhere in my Mind...
Posts: 1,864
Blog Entries: 7

The "Foxy" CPU - Part 2

Since I can't edit my Post I'm trying out something different, the last three sections are going to be put here so that I can try and edit the posts. I think it's the limitation for the site.

So here ya go!

The Basic Sections are still around, here:
http://www.wiremod.com/forum/cpu-tut...-foxy-cpu.html

Section VIII
(Advanced Concepts)

In this section, I'll be reviewing some more advanced CPU Programming concepts. Items such as reading and writing strings, segment registers, other macros and eventually Interrupts.

Note: Mcopy is covered in Section X under RAM, so if your looking for it, go there.

Reading and Writing Strings
(The DB Macro)

When using the console the easiest way to put text on the screen is using the DB macro. The DB macro makes a string of numbers and can also take straight text, which it translates into ASCII for use with the console screen.

In order to construct a DB string, you must use a label so that the CPU knows where the string is located in memory. Also you should not put DBs in the middle of your code, they need to come BEFORE the code (similar to alloc)

An Example of a bit of data used:

Label: db 1,2,3,4,5,6,7,8,9,10;

An Example of a zero-terminated String:

Label: db 'This is a String.',0;

Remember to put DB's under labels. Also ASCII strings use a single quote to denote ASCII Data. The reason why we usually terminate ASCII strings with zeros is so that in the code we know where the string ends.

Now your asking, "How the heck do I pull the data from that!?!?!". Well, that part is fairly simple, putting the string to the console is a little trickier.

In this Example we'll pull data from a small string, and apply a little math:

Code:

```
Data;

alloc result // put the result here

String1:
 db 9,1000,1; //yeah you know where I'm going with this one lol.

Code;

Main:
 mov eax,String1  //Yes it is a memory location but we want the ADDRESS
 mov ebx,#eax    //Pull the VALUE at the memory location (string)
 inc eax             // Move to the next value in the String
 mov ecx,#eax   //put that one in ECX
 inc eax
 mov edx,#eax  //here all the numbers have been pulled
jmp Maths

Maths:
 mul ebx,edx  //we get 9000
 add ebx,ecx  // now we have... 9001!
 mov #result,ebx //all done
```

Of course all that does it put the result in a spot. To be more useful we need to do something like write a string to a console.

So here is a bit of code that I'm using that should work for you too!

Code:

```
Data;

Line1:
 db 'Hola craps it works!',0: //the String is here

Code:

Writestring: //EBX = address to write, EAX = the db line to write, then call it.
 cmp #eax,0  //check to see if we have reached end of string "0"
je Writend
 mov #ebx,#eax  //put the chars to console!
 inc ebx        //Bump Console up one cell
 mov #ebx,999   //output char params
 inc ebx          //next console cell
 inc eax          //next value in the string
jmp Writestring      //keep writing till we hit a 0
Writend:
ret

Main: //the above is a CALL, read the comments on how to use it.
 mov ebx,65536 //Location of where to write the string to.
 mov eax,Line1  //The address of where Line is located is put into eax
call Writestring  //Make the call!
```

And that's all there is to it, the line with; mov #ebx,999 is the char parameters and you can change that to get different text color or background colors as well.

To Do:
Segment Registers! **<- NEXT!**
Interrupts (will probably be the last done in the sections)
ORG and OFFSET Macros

Section IX
(CPU Support Devices)

There are a couple of devices that are used with the CPU that aren't mentioned in a lot of tutorials. Such as the Data Port and the Address Bus. Here I'll explain what they are and what they're for!

Data Port
(Overview)

The Data Port is a device that allows the CPU to **interface with regular wire devices**. So you can hook your CPU up to an Airplane, Car, or Life Support devices and any existing wire components that gives an output and can take an input.

The Data Port has 8 Input and 8 Output ports, both inputs and outputs are labled from 0-7.

Wire the **IO Port** of the CPU to the Dataport. Select any port and click.

To use the Data Port you need to know a couple of commands and a parameter.

Code Examples:

In/Out

Code:
```
in eax,0 \\takes a value on Input Port 0 of the data port and stores it in eax
add eax,ebx \\a simple function, adds ebx to eax and stores the value in eax
out eax,0 \\puts the contents of eax onto the Output Port 0
```

PortX

To do the same thing as above using ports instead (smaller compiled code, so it's faster!). Remember: Intruction **Destination**,*Source*

Code:
```
mov eax,port0 \\pulls value from port 0 and stores in eax
add eax,ebx \\same as above
mov port0,eax  \\puts value of eax to port 0
```

Address Bus

The address bus is a device that allows you to **connect Highspeed Devices to the CPU**.

The Address Bus has 4 memory slots that are configured by Offset and Size.

To wire to the CPU wire the **Membus** on the CPU to the Address Bus.

The Offset is where the memory for the highspeed device starts. It is important to keep this number in mind, the most

common first slot device for an address but is 0. However don't let this throw you off. The memory actually starts AFTER the CPU memory (which is 65536 cells) and starts at 65536.

The Size is the number of cells to allocate for the highspeed device, for example the Highspeed Ranger takes 7 cells, the 64 Store RAM takes 64 cells.

So for example the configuration, if your setting up a pair of HS rangers and a 128 byte memory module, the configuration should be;

Memory 1
Offset: 0
Size: 7

Memory 2
Offset: 8
Size: 7

Memory 3
Offset: 15
Size: 128

And so on, for up to 4 slots.

Multiple Address Buses
(Add more devices to CPU)

We all know about the Address Bus can let you add up to 4 High Speed Devices to your CPU, but what if you want to add more than 4? Luckily it's pretty simple!

To add another 4 devices to the CPU you have to sacrifice on of the Memory slots for the Address Bus that you already have. **Now you simply attach another Address Bus to the memory slot!**

Now some Lame ASCII Art!

Code:

```
[Address Bus1] - Main Address Bus
[Dev1][Dev2][Dev3][Address Bus 2] - Address Bus attached to Memory4
------------------[Dev4][Dev5][Dev6][Dev7] - Attached to Address Bus 2
```

Now you may be asking, *"How do I configure the Address Spaces?"*

It's as simple as "allocating" Address Space to the extra Address Bus.
Let's say we have an Address Bus connected to the **Memory4** slot on the main Address Bus.

For Address Bus 1

Memory1
Offset:0
Size:256

Memory2
Offset:256
Size:256

Memory3
Offset:512
Size:256

Memory4 - **Let's Put the Second Address Bus Here!**

Offset: 768 - Where the Second Address Bus Starts
Size: 1024 - The amount of cells allocated to the second Address Bus

For Address Bus 2

Memory1
Offset:0 - This is ITS offset this starts at address (65536+768)
Size:256 - This is the allocated slot for THIS Address Buss' Memory1

Memory2
Offset:256
Size:256

Memory3
Offset:512
Size:256

Memory4
Offset:768
Size:256

As you can see all the memory slots on the second address bus can be used but only if they're "allocated" by the first address bus, you can have 4 secondary Address Buses per Address Bus and you can keep stacking them up as much as you want for however many devices you need!

TODO:
Multiple Data Ports

Section X
(Highspeed Devices)

There are may devices that the CPU can use, here I'll try to describe them as best as I can and give you as much information as I can about the devices and how to use them! Some of the information here might be a bit general until I start to know them better.

A lot of this stuff is already in the Documentation for Highspeed Devices, all the techinal infomation I have here is from there, so the credit belongs to Black Pheonix for being awesome enough to show us.

Description of Highspeed Devices

Highspeed devices are called such because they have whats called **"Internal Memory"** the inputs and outputs of Highspeed devices are purely memory, by inputting data into memory the device responds accordingly, the same with pulling information from a Highspeed device. Accessing a Highspeed Device is similar to **DMA** (Direct Memory Access) in computers, because that is exactly what your doing. This is what makes Highspeed Devices "Highspeed", that is why learning how to use memory effectively comes into play.

Note: Be sure you have read and understood working with memory before trying to use Highspeed Devices. Reading **Section V** and beyond is highly recommended.

Wire Keyboard
(Description and Functions)

The Keyboard is probably one of the more mysterious devices in wire and seems to only have a use with the CPU. However! This is not the case! If you attach a regular screen to the keyboard you can see numbers that are displayed as you press keys on the keyboard. You can use this regular wire data (as opposed to Highspeed data) to trigger events or even use with the E-gate to produce any number of devices!

Standard Wire Interface

Inputs: None

Outputs: (1) Key Pressed

To use:

Attach a Wire device to the keyboard to get ASCII Numbers from the keyboard.

Highspeed Interface:

Inputs: (1-31) Writing to buffer clears the buffer after the cell you wrote to.

Outputs: (0) Key Numbering (1-31) Keyboard Buffer (32-255) ASCII Mapped Array
Zeroth (first cell) contains the number of keys pressed.
Cells 1-31 contain the keyboard buffer of keys pressed.
Cells 32-255 contain an ASCII Mapped Array

To use:

In order to pull the key pressed from the device you need to read the **second** cell of the Keyboard (which is actually cell 1), the first cell (cell 0) of the device tells you how many keys are in the buffer. When you read from the device you need to write back to it, the key that was pressed back to the second cell of the device.

Code example:

This example assumes that a console is hooked up to an Address bus (memory 1) as well as the keyboard (memory 2) with the correct allocations. It is designed to put characters on the screen, and that's it.

Code:

```
Data;
define key_off,67586     //Second byte of Keyboard Mem
define con_off,65536     //First byte of Console Mem
alloc charpos,0          //Counter used to determine postion of keys in Writer.

Code;
Write:             //General Write Function
 mov ebx,#key_off  //Move Key to ebx save to write back to buffer
 mov eax,con_off   //Set eax to start of console memory
 add eax,#charpos  //Add to eax the value of the counter (charpos)
 mov #eax,#key_off //Move character into console
 inc eax           //Move to next cell over
 mov #eax,999      //Move charparam into cell
 mov #key_off,ebx  //Write char back to keyboard buffer
 add #charpos,2    //Move to next cell, ready to write
jmp Write         //Go back to key detection
```

Console
(Wire Console Screen)

The Console is arguably on of the coolest inventions out there for the CPU. It allows you to display text on a screen controlled by the CPU! With the Console screen you can now make programs that have an interface you can read.

Standard Wire Interface:

Inputs:
Char - ASCII value of character to be put on screen
CharX - Character Position (X axis)
CharY - Character Position (Y axis)
CharParam - The color/background of characters 999 = White on Black
CLK - On/Off
Reset - Resets the screen

Outputs:

Screen

To Use:

Set CLK to 1 to turn on the screen, for white text on black background use 999 as the Charparam, CharX is the location of the character on the horizontal (30 characters per row) and CharY is the Vertical (16 Rows). Char is the ASCII number that is associated with a number, give it to the Char input.

Highspeed Interface:

Information here is based on wiring the Screen to the CPU directly or used as the first Memory Device on an Address Bus with 0 offset.

Inputs:
Memory locations 0 - 1080 display text on the screen, Even addresses are cells that hold the Character, Odd addresses hold the Character Parameters.
Memory locations are for the following:
Clk: 2047
Clear Screen : 2041

Outputs:
None
However, in odd cases, you can actually use the Console as Ram

To write values to the Console here is a code Example:
For Writing String values to Console see DB Macro Examples

Code example:

Code:

```
Data;
define console,65536  //sets a place holder for use with console
define con_clk,67582  //sets a place holder for use with clk

Code;

mov #con_clk,1  //turns the console ON

Main:
 mov eax,console //moves into eax the MEMORY Address for the console
 mov #eax,55      //moves a number in to the console
 inc eax             //move to next cell in console
 mov #eax,999     //move 999 into the next cell over for Char Parameters
jmp Main:
```

This code will simply output a number to the console after turning it on.

Ram
(Ram Memory Modules & Mcopy)

Ram Memory modules are probably the easiest to work with using the CPU. The code to use ram is as simple as using the memory in the CPU itself. Ram is simply an extension to the CPUs internal memory!

Standard Wire Interface
(To Do)

Inputs:
Clk, AddrRead, AddrWrite, Data, Reset

Clk - clock (on/off)
AddrRead - the location of the Data Cell to READ from.
AddrWrite - Location of the Data Cell to WRITE to.
Data - The value to Save

Reset - Resets the Ram Chip

Outputs:
Gives Value of memory location

For multi-dimensional Ram (such as 64x64 or 64x64x64) the outputs are the same, the address inputs are altered slightly adding a pair for each dimension. The Data Input remains the same.
For 2D memory:AddrWriteX, AddrReadX
AddrWriteY, AddrReadY
For 3D memory:
AddrWX, AddrRX
AddrWY, AddrRY
AddrWZ, AddrRZ

Writing to X or Y or Z is fairly simple for 64x64 simply write data to Y until you reach 63 (0-63) then increment Y (next row). The same approach goes to writing to a 64x64x64 Ram chip (holy crap...). When you get to the end of X increment Y (row), when you get to the end of Y increment Z ("up").

Interesting notes:

Multi-Dimensional Memory

64x64 = 4,096 = 4KBs
64x64x64 = 262,144 = 256 KBs

Single Dimensional Memory

8 Store = 8 Cells = 8 Bytes
64 Store = 64 Cells = 64 Bytes
32K = 32,768 Cells
128K = 131,072 Cells
1MB = 1,048,576 Cells
EEPROMs (max per drive) 256K = 262,144 (Persistant!)
Dynamic Memory (Unofficial) 2MBs Max...
2,097,152 Cells (Optional Persistance!)

I'm not sure how the Muli-dimensional ram interfaces with highspeed devices... so if someone could fill me in that would be awesome.

Highspeed Interface

Inputs:
Memory location greater than 65535

Outputs:
Memory location Greater that 65535

To Use:

This example is based on a setup with a Ram chip of any size connected to the CPU Address Bus directly, this example shows transferring data from CPU memory to ram using traditional methods and the Mcopy instruction.

To transfer data using traditional methods
(good for small data transfers or non-linear/scattered data transfers)

Code Example:

Code:

```
Data;

define cpu_offset,65536     //the first cell in an HS device, one past CPU memory
define data_size,1024          //1024 bytes of data to move (a lot!) 1KB
alloc counter                   //Use a counter to keep track of how many bytes have been moved!
```

```
Code;

Setup: //don't jump back to here
 mov eax,5000         //set a location for where the data is
 mov ebx,cpu_offset //set the location for the CPUs offset to ebx
 out 0,0                 //Reset "finished" indicator, so you know its done
jmp Main

Main: //main loop that moves the data around
 cmp #counter,1024 //see if we moved 1024 bytes yet
je End // If all bytes moved goto Idling.
 mov #eax,#ebx  //transfers the byte at 5000 to 65536 first run
 inc eax    //move to next cell in CPU to transfer
 inc ebx    //move to next cell in RAM to transfer
 inc #counter //increment the counter by 1 when 1 byte is transferred
jmp Main  //Rinse, Repeat!

End:  //Uber Leet Idle Loop, make an indicator of some kind to let you know it is done
 out 0,100 //tell us when the deed is done! So we know it worked!
jmp End
```

Just like that! This program will transfer 1KB of data from CPU memory to External RAM, or EEPROM. This is sufficient to do Image Data from a 32x32 Digital Screen!

Be sure to note the indicator and the counter, the indicator obviously will tell us that the job is finished, this is a good tool to use and good for debugging, use often! The counter is always a good thing to have in software so you know how often a piece of code has been executed, in this case, each loop transfers one byte, so we know how much we transferred. Now we can tell the program to stop, when it'd done and let us know!

Mcopy
(It's Over Here!)

In this example we will do the EXACT same thing only using the "near-instantanious" copying mechanism that ownz all, the wonderful Mcopy instruction! Using the same setup scenario as above.

I'm covering Mcopy here because the example here and above would be EXACTLY the same, so in an effort to reduce redundancy, I've placed it here with an example of how to use it!

NOTE: Mcopy only transfers up to 8192 Bytes of data at a time, this is the maximum size. To do more than one set, simply use the Mcopy instruction more than once as ESI and EDI are both incremented by how much data you transferred!

Code Example:

Code:

```
Data; Same definitions only... we don't need a counter! ESI and EDI do that for us!

define cpu_offset,65536
define data_size,1024

Code;

Setup: //don't jump back to here
 mov eax,5000         //set a location for where the data is
  mov ebx,cpu_offset //set the location for the CPUs offset to ebx
 out 0,0                 //Reset "finished" indicator, so you know its done
jmp Main

Main: //Since where only doing one transfer with Mcopy it is simpler.
 mov esi,eax  //ESI is the SOURCE address to start at!
 mov edi,ebx  //EDI is the DESTINATION address to transfer TO!
 mcopy 1024 //Mcopy only takes one paremeter, SIZE
jmp End

End:  //Thats IT! Holy Cow!
 out 0,100
jmp End
```

Since we're transferring less than 8192 bytes we can safely run Mcopy only once, it is very reliable so have fun with it! Mcopy also makes things simpler and less code, and Mcopy is extremely efficent for large data transferrs, in fact to format a 32KB RAM module would take exactly FOUR Mcopy instruction and only take a few seconds. Doing that the traditional way, even with a fast CPU would take quite a bit longer.

Digital Screen

The Digital Screen is a device that renders 1024 pixels on the screen, its memory layout is a little different than the console screen as it serves a different purpose. The digital screen has been used most commonly as a means of displaying images (usually ranger based "depth/height maps").

Standard Wire Interface:
(To Do)

Inputs:PixelX - Location of pixel in X axis (horizontal)
PixelY - Location of pixel in Y axis (vertical)
PixelG - Color of Pixel (RGBGGG format)
Clk - On/Off
FillColor - Fill Color
ClearRow - Clear Row (X axis)
ClearCol - Clear Column (Y axis)

Output:

Screen

Highspeed Interface:

Inputs:
Memory location greater than 65535 up to 66559 directly connected, uses 1024 cells. Set Memory Slots in Address Bus size to 1024.

On - Cell 2047
Clear Screen - Cell 2041

Outputs:
On screen

NOTE: The Digital Screens Pixels use ONE memory cell as opposed to the Consoles two. The pixels are represented by a "dual-pixel" in this format;

RGBGGG - for example: 900000 should be Red, 000555 (555) should be Grey, 000 is Black and 999 would be White (for shades of Grey).

To Do:
(researching code example)

Data Plug
(Sockets & Plugs)

Input:
High Speed Device

Output:
Highspeed Memory Location of Device(s)

One of the more fun devices that can be used with the CPU are the Data Sockets (under Data Plug) that can take Data Plugs. To attach a Data Plug to a Data Socket, simply get the plug near the socket and it will get "attracted" to the socket and "lock in", then it will be ready for use!

But the most common question is... **"What the heck IS it?"**. The Data Plugs are simply an extension of the Address Bus Memory Slots. It allows you to detach devices and items from the Address Bus

For example; If you want to use Ram that you can attach and detach from a CPU setup, spawn a Data Plug (grab Data Plug and Right Click) and attach a Ram Chip to the Data Plug. Wire the Data Plug to the Ram chip. That's all that's required to connect Ram or CPUs to the Data Plugs.

The Data Sockets are wired to Address Buses, if you configured your Address Bus to something like this to accept 32KB Ram modules;

Memory1
Offset: 0
Size : 32768

You don't have to change the configuration at all! Simply Wire the Memory1 port of the Address Bus to the Data Socket and BAM! That's it!

Now you can attach that Data Plug with the Ram Chip on it, you can access the Ram as if it was directly wired directly to Memory1. ^_^

Of course the difference is now you can unplug it and take it with you wherever you go!

Highspeed Rangers
(Rangers on Steroids!)

Highspeed Rangers are a variant of the traditional ranger, although they don't have the capability of the regular rangers (SteamID, EntityID etc.) it does have the ability to see colored props and measure distance.

"Well if it's so limited... Why use it?"

It does what it does, better than what a regular ranger can do. Even faster than what the E-Gates can do! I know most of you have heard of the Wired Camcorder by McLovin, which uses a Highspeed Ranger and can actually record video at about 5 fps with a 16x16 resolution!

While this task would be near-impossible for an E-gate to achive, the CPU can complete this task easily. Even a low speed CPU can capture 32x32 images in just a few seconds.

The main perk? It's **FAST**. Although the HS ranger can do Camera stuff, it can be used in any device that requires multiple traces at blinding speeds.

So how do we use it? It doesn't have any wire inputs so a E2 Wirelink or CPU are required.

The Input and Outputs follow in the list below. (Taken from Borsy's post, and expanded) Here are the Cell numbers of the HS Ranger and what they're for.

0 = Trace! (1) **(Input)**
A Trace is another way of saying "Take a Measurement" ie. it will output what it hits. Send a Value of 1 to this Cell each time you want to find out what the ranger hits.

1 = Result **(Output)**
The result of a Trace is put here in Cell 1 of the HS Ranger. Read this cell to pull the resuts of the trace. This is the only Output for the HS Ranger.

2 = Range (0 ... 2000) **(Input)**
Range is what is means, just like the regular ranger, send a value to this Cell as a value of Distance.

3 = Default to Zero (0 or 1) **(Input)**
This is similar to the Checkbox for the regular ranger. Write a 1 (Yes) or 0 (No) to tell the Ranger to Default to Zero or not.

4 = SkewX (-1 ... 1) **(Input)**
SkewX is the X position of the Ranger, it takes values from -1 to 1. Send a Value to this Cell to set the X of the Ranger to Scan to.

5 = SkewY (-1 ... 1) **(Input)**

SkewY is the Y position of the Ranger, it takes values from -1 to 1. Send a Value to this Cell to set the Y of the Ranger to Scan to.

6 = Trace Water (0 or 1) **(Input)**
This is the also the same as the "Hit Water" check box in the regular ranger. Give this Cell a 1 (Yes) or a 0 (No) to set this value

7 = unused
This Cell Exists, but is unused. I have no idea what happens if you use this cell for anything. Either A) It will cause Strange Results **OR** B) Could be used as a Byte of Memory. Either Prospect would be interesting! If you find out, let me know! ^_^

Code Example to Come!

I Need Suggestions!

Are there High Speed Devices that I'm missing?
Is there a device that has highspeed capability that isn't advertised?

Let me know I'll research it and Post information about it!

(UWSVN devices are accepted!)

Last edited by -=Fox=-; 02-11-2009 at 12:27 PM.

Share

denkdaetz likes this.

http://tiny.cc/OMFGWTFBBQ

Best People On Wiremod!

Black Phoenix, Azrael, **Jat Goodwin**, Magos Mechanicus, ITSBTH, Fizyk, g33v3s,**tuusita**, InfectiousFight, ief015

Pointless things that are pointless, are pointlessly pointless, therefore pointlessness is pointless.
So pointlessly pointing out the pointlessness of this pointless signature is utterly pointless.
My IQ is 123

Reply With Quote

12-27-2008 #2

IEF015

Wire Sofaking

WIREMOD MEMBER

Join Date: Feb 2008
Location: London, ON (Canada, eh?)
Posts: 1,687

Re: The "Foxy" CPU - Part 2

Holy crap. Nice one BP Jr.

Thumbs up from me.

Share

"It's my favourite country song. And I hate it."

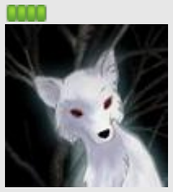
Reply With Quote

12-27-2008 #3

-=Fox=-

Wire Sofaking

WIREMOD MEMBER



Join Date: Feb 2007
Location: Somewhere in my Mind...
Posts: 1,864
Blog Entries: 7

Re: The "Foxy" CPU - Part 2

Updated Console Screen and DB macro, so I wanna see some programs now! :lol:

Share

http://tiny.cc/OMFGWTFBBQ

Best People On Wiremod!

Black Phoenix, Azrael, **Jat Goodwin**, Magos Mechanicus, ITSBTH, Fizyk, g33v3s,**tuusita**, InfectiousFight, ief015

Pointless things that are pointless, are pointlessly pointless, therefore pointlessness is pointless.
So pointlessly pointing out the pointlessness of this pointless signature is utterly pointless.
My IQ is 123

Reply With Quote

12-27-2008 #4

Hitman271

Wire Sofaking

WIREMOD MEMBER

Join Date: Feb 2008
Location: Why? You looking for somebody?
Posts: 738

Re: The "Foxy" CPU - Part 2

Code:

```
jmp DoThis:
Special:
 db 'H,999,e,999,l,999,l,999,o,999, ,999,W,999,o,999,r,999,l,999,d,999'

SpecialEnd:

DoThis:
mov esi,Special
mov edi,65550 //Place in Console Screen to write to

mov eax,esi
mov ebx,SpecialEnd

sub ebx,eax

mcopy ebx

//END
```

This is theoretically the fastest way to copy any string to screen if it has the parameters in it like that. This has no loop and uses the almost-infinite transfer opcode, mcopy.

Share

> Originally Posted by **Anticept**
> *This is not some place where you can toss your dick around and expect people to suck it.*

Community Gpu Thread. Post Yours!

Bouncy Ball

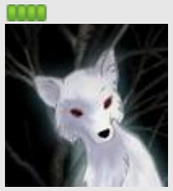
Reply With Quote

12-27-2008 #5

-=Fox=-

Wire Sofaking

WIREMOD MEMBER



Join Date: Feb 2007
Location: Somewhere in my Mind...
Posts: 1,864
Blog Entries: 7

Re: The "Foxy" CPU - Part 2

> Originally Posted by **Hitman271**
> *This is theoretically the fastest way to copy any string to screen if it has the parameters in it like that. This has no loop and uses the almost-infinite transfer opcode, mcopy.*

The only problem with your method.. is that it would take more work, it is FAR easier to setup a call to write the String, load parameters, then make the call.

That way writing a string only takes 2 lines, location of string, and the call, or with 3 lines you can tell the call where to put the string.

But right, that looks a LOT faster.

However... using DB to copy the contents of something like a GUI, to a spot in memory during an "initialization", then using the Mcopy command to render the text would also be a very fast way to implement writing to console. That way rendering menus or commonly used text would be rendered near instantly.

Also the Mcopy command can only transfer 8192 bytes of data at a time.

If you can test the theory, let me know what you get!

Share

http://tiny.cc/OMFGWTFBBQ

Best People On Wiremod!

Black Phoenix, Azrael, **Jat Goodwin**, Magos Mechanicus, ITSBTH, Fizyk, g33v3s,**tuusita**, InfectiousFight, ief015

Pointless things that are pointless, are pointlessly pointless, therefore pointlessness is pointless.
So pointlessly pointing out the pointlessness of this pointless signature is utterly pointless.
My IQ is 123

Reply With Quote

12-27-2008 #6

Hitman271

Wire Sofaking

WIREMOD MEMBER

Join Date: Feb 2008
Location: Why? You looking for somebody?
Posts: 738

Re: The "Foxy" CPU - Part 2

I didn't actually mean for my method to be used at all lol. It is in-fact an extremely hard way to do it. As I did make typos while putting in those 999.

BUT, if you were to put 999's a full page of memory, then place an inputted string into every other location, that way could be done.

Share

> Originally Posted by **Anticept**
> *This is not some place where you can toss your dick around and expect people to suck it.*

Community Gpu Thread. Post Yours!

Bouncy Ball

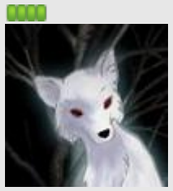
Reply With Quote

12-28-2008 #7

-=Fox=-

Wire Sofaking

WIREMOD MEMBER



Join Date: Feb 2007
Location: Somewhere in my Mind...
Posts: 1,864
Blog Entries: 7

Re: The "Foxy" CPU - Part 2

That's exactly how the string writing program does the work, only it fills the 999s in with the characters, doing the 999's as a "pre-write" would work... if you don't clear the screen xD

A writing program that pre-writes the 999's... is actually a really good idea, that would eliminate a few lines of code from the writing function... however... over all it would take the same amount of time, as Mcopy is inefficient for small bits of code, and thus takes longer, so you'd still have to write every other cell manually.

Share

http://tiny.cc/OMFGWTFBBQ

Best People On Wiremod!

Black Phoenix, Azrael, **Jat Goodwin**, Magos Mechanicus, ITSBTH, Fizyk, g33v3s,**tuusita**, InfectiousFight, ief015

Pointless things that are pointless, are pointlessly pointless, therefore pointlessness is pointless.
So pointlessly pointing out the pointlessness of this pointless signature is utterly pointless.
My IQ is 123

Reply With Quote

12-28-2008 #8

Hitman271

Wire Sofaking

WIREMOD MEMBER

Join Date: Feb 2008
Location: Why? You looking for somebody?
Posts: 738

Re: The "Foxy" CPU - Part 2

No, I meant writing a full block of 999's inside the cpu's ram. That way a multitude of screens could be written to in about three commands lol

Share

> Originally Posted by **Anticept**
> *This is not some place where you can toss your dick around and expect people to suck it.*

Community Gpu Thread. Post Yours!

Bouncy Ball

Reply With Quote

12-28-2008 #9

Re: The "Foxy" CPU - Part 2

LOL!!! Ok I gotcha! Pop in a code example, and I'll stuff it in the Advanced Concepts with your name on it

Updated RAM description with instructions on using MCOPY! Go check it out!

Two Birds with one stone this time ^_^

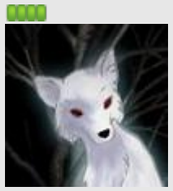
http://tiny.cc/OMFGWTFBBQ

Best People On Wiremod!

-=Fox=-

Wire Sofaking

WIREMOD MEMBER



Join Date: Feb 2007
Location: Somewhere in my Mind...
Posts: 1,864
Blog Entries: 7

Share

Black Phoenix, Azrael, **Jat Goodwin**, Magos Mechanicus, ITSBTH, Fizyk, g33v3s,**tuusita**, InfectiousFight, ief015

Pointless things that are pointless, are pointlessly pointless, therefore pointlessness is pointless.
So pointlessly pointing out the pointlessness of this pointless signature is utterly pointless.
My IQ is 123

Reply With Quote

12-28-2008 #10

Azrael

Developer

WIREMOD DEVELOPER

Join Date: Aug 2007
Posts: 1,959

Re: The "Foxy" CPU - Part 2

Oi. Hurry up and write about segment registers. They're un-FUCKING-believably useful.

Share

Dictated but not read,
Abigail Buccaneer

Reply With Quote

Page 1 of 5 1 2 3 ... Last

« Previous Thread | Next Thread »

Similar Threads

The "Foxy" CPU. By -=Fox=- in forum CPU Tutorials	Replies: 125 Last Post: 07-01-2011, 09:19 PM
Want to understand "interval" and "timer" in Expression 2 By anthraxyhe in forum Installation and Malfunctions Support	Replies: 25 Last Post: 12-08-2009, 09:36 AM
Xtensity's "Spaceman Turret V6" Tutorial, "Head Shots FTW" By Xtensity in forum Gate Nostalgia (Old School Wiring) Discussion & Help	Replies: 24 Last Post: 09-20-2008, 07:46 AM
the "WHAT HAPPENED TO" series, part II: GM_COMMODIUS By LimEJET in forum Off-Topic	Replies: 2 Last Post: 04-01-2008, 11:53 AM
Need the pack that has the "Generator" and "Razor" models...	Replies: 2

By Mr. Brightside in forum Installation and Malfunctions Support Last Post: 06-06-2007, 10:15 AM

Bookmarks

Digg
del.icio.us
StumbleUpon
Google
Facebook

Posting Permissions

You may not post new threads
You may not post replies
You may not post attachments
You may not edit your posts

BB code is On
Smilies are On
[IMG] code is On
[VIDEO] code is On
HTML code is Off
Trackbacks are On

Pingbacks are On
Refbacks are On

Forum Rules

All times are GMT -7. The time now is 01:30 PM.