

Forum:[Hack Forums](#)

Thread: [What would Binary be considered as?](#)

((void (*)())NULL)() - April 11, 2011, 2:31 pm

Hello. What would Binary be? Like would it be a programming language? Same with Hexadecimal, ASM, and other low level machine code languages.

Zer0Fact0r - April 11, 2011, 3:30 pm

Binary and hexadecimal are merely numeral systems. Assembly is the lowest-level programming language, and it can be represented as mnemonics or bytes.

Ethereal - April 11, 2011, 3:43 pm

Machine code, I should imagine. However, it's not really a language, just a numerical representation of data.

/thread

Disease - April 11, 2011, 4:14 pm

(Apr 11, 2011 02:31 PM)((void (*)())NULL)() Wrote: [\[CITE\]](#) Hello. What would Binary be? Like would it be a programming language? Same with Hexadecimal, ASM, and other low level machine code languages.

It's a number system just like decimal (what we count in).

Binary - Base 2
Octal - Base 8
Decimal - Base 10
Hexadecimal - Base 16

Just to name the more common number systems.

((void (*)())NULL)() - April 11, 2011, 5:44 pm

(Apr 11, 2011 03:43 PM)Ethereal Wrote: [\[CITE\]](#) Machine code, I should imagine. However, it's not really a language, just a numerical representation of data.

/thread

Thanks. At least I got part of it right "low level machine code".

alfonzo1955 - April 11, 2011, 5:57 pm

Machine code is the lowest level, not assembly.

Zer0Fact0r - April 11, 2011, 6:38 pm

(Apr 11, 2011 05:57 PM)alfonzo1955 Wrote: [\[CITE\]](#) Machine code is the lowest level, not assembly. Assembly is machine code.

((void (*)())NULL)() - April 11, 2011, 7:36 pm

(Apr 11, 2011 06:38 PM)Zer0Fact0r Wrote: [\[CITE\]](#) Assembly is machine code.
That's what I thought lol. ?

kashug - April 11, 2011, 7:43 pm

(Apr 11, 2011 06:38 PM)Zer0Fact0r Wrote: [\[CITE\]](#) Assembly is machine code. This is wrong. Assembly gets assembled into machinecode. Hex/binary is machinecode.

You could write 1 assembly code and run it through 2 different assemblers and get 2 different binary-codes, dependent on the architecture on the system it is supposed to run on.

`((void (*)())NULL)()` - April 11, 2011, 7:47 pm

(Apr 11, 2011 07:43 PM)kashug Wrote: [\[CITE\]](#) This is wrong. Assembly gets assembled into machinecode. Hex/binary is machinecode.

You could write 1 assembly code and run it through 2 different assemblers and get 2 different binary-codes, dependent on the architecture on the system it is supposed to run on.
Then what is Assembly?

You're correct according to [this](#)

kashug - April 11, 2011, 7:55 pm

(Apr 11, 2011 07:47 PM)`((void (*)())NULL)()` Wrote: [\[CITE\]](#) Then what is Assembly?

You're correct according to [this](#) I would say a language that directly describes the machine-code. Each assembly-instruction gets directly converted to 1 specific binarycode. The difference is basically that if you write the machine-instruction
1110 1111 1111 0000 1110 1111 1111 0000 = 0xEFF0EFF0
They directly relate to each-other and describe the same mathematical value.

while the assembly instruction
add t1, t2, t3 # t3 = t1+t2

Would become 2 different machinecodes depending on if you want it to run on an x86 processor, MIPS-processor or a RISC-processor.

TLF-KoBE - April 11, 2011, 7:56 pm

(Apr 11, 2011 05:57 PM)alfonzo1955 Wrote: [\[CITE\]](#) Machine code is the lowest level, not assembly.
(Apr 11, 2011 07:43 PM)kashug Wrote: [\[CITE\]](#) You could write 1 assembly code and run it through 2 different assemblers and get 2 different binary-codes, dependent on the architecture on the system it is supposed to run on.

Both are correct. Assembly is assembled into machine code, which is the lowest level. Machine code is machine specific so each different machine would have a different set of instructions depending on the different components.

Edit: sorry my post was a bit late...

`((void (*)())NULL)()` - April 11, 2011, 8:19 pm

How do you guys tell if it's high level or low level machine code? Sorry for being such a noob, I joined Hack Forums a couple of days ago.

kashug - April 11, 2011, 8:25 pm

(Apr 11, 2011 08:19 PM)`((void (*)())NULL)()` Wrote: [\[CITE\]](#) How do you guys tell if it's high level or low level machine code? Sorry for being such a noob, I joined Hack Forums a couple of days ago. Huh? Machine-code is always lowlevel. Assembly is also a low-level language. Things like C is highlevel languages.

bottomy - April 11, 2011, 8:53 pm

Actually it could be considered the same level as machinecode, since assembly in it's purest context is just the human readable representation of machinecode.

(Apr 11, 2011 07:55 PM)kashug Wrote: [\[CITE\]](#) I would say a language that directly describes the machine-code. Each assembly-instruction gets directly converted to 1 specific binarycode. The difference is basicallt that if you write the machine-instruction
1110 1111 1111 0000 1110 1111 1111 0000 = 0xEFF0EFF0
They directly relate to each-other and describe the same mathematical value.

while the assembly instruction
add t1, t2, t3 # t3 = t1+t2

Would become 2 different machinecodes depending on if you want it to run on an x86 processor, MIPS-processor or a RISC-processor.

What are you talking about? Assembly is specific to a certain architecture, so if you have assembly for PowerPC and tried assembling it for x86 you would just get errors. So the hexadecimal or binary can be converted to assembly and vice versa.

oprichniki - April 11, 2011, 10:37 pm

"Programming in Binary" would differ with different machines, but there's a basic pattern in how the coding is done. In reality, you just have a long, continuous binary string, but the computer interprets various sections of it depending on what the code is. For example, an op code might be of size 8 bits, so the cpu takes those 8 bits, decodes them, and executes the op code. If the op code requires 2 1 byte arguments, then the next 16 bits are used for the arguments. So for one command, you just used 24 bits. Then, the process repeats: the next 8 bits are decoded as an op code, etc. A program counter keeps track of where you are in the binary stream, and that value can be changed to jump to different parts of the code. That's how loops work.

((void (*)())NULL)() - April 12, 2011, 5:15 am

I think that Assembly would not be considered machine code. Machine code uses numbers right? Like Hexadecimal, and Binary. Assembly uses words instead of numbers, I think. It's just what I think, I might be wrong.

Ethereal - April 12, 2011, 9:32 am

(Apr 12, 2011 05:15 AM)((void (*)())NULL)() Wrote: [\[CITE\]](#) I think that Assembly would not be considered machine code. Machine code uses numbers right? Like Hexadecimal, and Binary. Assembly uses words instead of numbers, I think. It's just what I think, I might be wrong.
Assembly is a textual representation of machine-code.

Zer0Fact0r - April 12, 2011, 1:10 pm

(Apr 11, 2011 07:43 PM)kashug Wrote: [\[CITE\]](#) This is wrong. Assembly gets assembeled into machinecode. Hex/binary is machonecode.

(Apr 11, 2011 03:30 PM)Zer0Fact0r Wrote: [\[CITE\]](#) Assembly is the lowest-level programming language, and it can be represented as mnemonics or bytes.

Reversing: Secrets of Reverse Engineering Wrote:Another important concept to get out of the way is machine code (often called binary code, or object code). People sometimes make the mistake of thinking that machine code is "faster" or "lower-level" than assembly language. That is a misconception: machine code and assembly language are two different representations of the same thing. A CPU reads machine code, which is nothing but sequences of bits that contain a list of instructions for the CPU to perform. Assembly language is simply a textual representation of those bits—we name elements in these code sequences in order to make them human-readable. Instead of cryptic hexadecimal numbers we can look at textual instruction names such as MOV (Move),

XCHG (Exchange), and so on.

No, assembly is the human-readable form of machine code. Did you miss the second part of what I wrote?

(Apr 11, 2011 07:43 PM)kashug Wrote: [\[CITE\]](#) You could write 1 assembly code and run it through 2 different assemblers and get 2 different binary-codes, dependent on the architecture on the system it is supposed to run on.

Assembly is platform-specific. Each opcode is equivalent to exactly one set of bytes. For example, the "nop" instruction for the x86 ISA represents 90h. "nop" for a different architecture may be a completely different opcode altogether.