

# Forum:[Hack Forums](#)

Thread: [x86 vs. x64 \(32-bit versus 64-bit\)](#)

**Hackers** - March 28, 2024, 9:04 pm

What do you think of x86 versus x64 programs?

For 64-bit it's becoming more popular & modern devices have no issues running them. However, I've noticed many tools are still built with 32-bit.

Personally, I have been doing some more x86 for testing out some DLLs of my own & testing things. However, I usually rock 64-bit. I've fully transitioned (except for few projects I already started) to Visual Studio & it's much, MUCH easier to swap between 32 and 64 builds. I'm liking VS for sure (was using VS Code w/ manual gcc compiles before).

What do you code for mostly?

Here is a snippet from GPT on pros & cons ~ take with a grain of salt for obvious reasons :)

The main differences between compiling a program for x86 (32-bit) vs x64 (64-bit) architectures, focusing on the end result, can be summarized as follows:

Addressable Memory:

x86: Limited to addressing up to 4 GB of RAM.

x64: Can address more than 4 GB of RAM, up to 16 exabytes theoretically.

Data Types:

x86: Generally, the size of long and pointer types is 32 bits.

x64: Long and pointer types are typically 64 bits, allowing for operations on larger data sets.

Performance:

x86: May have slightly faster execution for some 32-bit applications due to smaller data types and less memory usage.

x64: Can offer better performance for applications that require large data types or need to access more RAM, due to a larger number of registers and direct memory access capabilities.

Compatibility:

x86: Can run on both 32-bit and 64-bit operating systems, though with the memory limitations of 32-bit architectures.

x64: Requires a 64-bit operating system. Can run 32-bit applications through emulation or compatibility layers, but native 64-bit applications cannot run on a 32-bit OS.

#### Number of General Purpose Registers:

x86: Has 8 general-purpose registers.

x64: Extends the number of general-purpose registers to 16, improving the efficiency of compiled code.

#### Operating System Support:

x86: Widely supported by both modern and older operating systems.

x64: Supported by modern operating systems, with increasingly dominant support over x86 due to its advantages in handling large memory and performance.

#### Software Availability:

x86: While still widely supported, newer software versions and features may not be available for 32-bit versions.

x64: Increasingly becoming the standard for software releases, offering more features and better performance.

I had another tool audit the list for accuracy:

Spoiler ([Click to View](#))

Quote: Your summary of the main differences between compiling for x86 (32-bit) and x64 (64-bit) architectures is accurate and well-organized, focusing on key aspects that differentiate the end result of programs compiled for these two architectures. Here's a brief review for accuracy and additional clarity:

**Addressable Memory:** Correct. x86 architectures are limited to 4 GB of addressable memory, whereas x64 architectures can theoretically address up to 16 exabytes of memory. This vast difference makes x64 more suitable for applications requiring large amounts of memory.

**Data Types:** Correct. The transition from 32-bit to 64-bit increases the size of long and pointer types from 32 bits to 64 bits, impacting data handling and operations within applications.

**Performance:** Correct, with the nuance that the performance benefits of x64 over x86 depend on the application's specific needs. x64 architectures can handle larger data more efficiently and benefit from additional registers, but this does not automatically translate to faster execution for all types of applications.

**Compatibility:** Correct. x86 applications can run on both 32-bit and 64-bit systems (though with limitations on 64-bit systems), while x64 applications require a 64-bit operating system.

Number of General Purpose Registers: Correct. The increase in the number of general-purpose registers in x64 architectures allows for more efficient processing and can significantly impact the performance of certain applications.

Operating System Support: Correct. Modern operating systems increasingly support x64 architecture preferentially, with x86 support maintained for compatibility with legacy applications and systems.

Software Availability: Correct. There's a trend towards x64 software due to its advantages, and while x86 applications are still widely supported, some new features and optimizations might only be available for x64 versions.

Overall, your comparison accurately captures the key differences between compiling programs for x86 vs. x64 architectures, with an emphasis on how these differences impact the end result in terms of performance, memory access, and compatibility.

**breaker1911** - March 29, 2024, 1:39 am

32-bit if you targeting wider support by old device, which still using 32-bit, they're weak  
64-bit can run 32-bit app, but 32-bit will crash if you run 64-bit app.

**Hackers** - March 29, 2024, 12:04 pm

(Mar 29, 2024 01:39 AM)breaker1911 Wrote: [\[CITE\]](#) 32-bit if you targeting wider support by old device, which still using 32-bit, they're weak

64-bit can run 32-bit app, but 32-bit will crash if you run 64-bit app.

True, true - makes sense

Do you think that when it comes to whitehat, new software there is any reason to use 32? Do you think it's still worth it to compile two versions, 1 for 32 & 1 for 64, or is it better to just go with 64 at this pt?

**Sajuuk** - March 30, 2024, 8:10 am

In terms of development for Windows, we all should've been targeting both x64 and x86 regardless of what architecture we're developing on. Many of the tools I use today offer both 32 and 64-bit versions presumably purely for compatibility reasons.

As far as I can tell, the large majority of Windows users are running 64-bit, so building for x64 makes total sense because of the edge it holds over x86. Since 32-bit apps are backwards compatible and will run on 64-bit systems, it's always wise to compile versions of your project that will run on either.

Targeting only x86 opens up the entire Windows userbase, but you won't be able to use the performance x64 offers. Targeting only x64 cuts your userbase, but by exactly how much in these times I'm not entirely sure. I've watched many of the programs I've used over the years slowly switch over to x64 only, and due to nothing other than typical technological progression.

**PulseLink** - April 4, 2024, 5:09 pm

RunPE also must be aware of the target architecture to handle operations. The pointer size and register assignments need to be correct, calling conventions, etc.

On 64 bit systems win32 applications are run under WoW64, essentially emulating a 32bit environment passing arguments via fastcall through the registers. Another thing to account for in a runpe application when handling filesystem/registry.

at the end of the day there are things you can do as a malware developer to ensure code runs

correctly on the machine its executed on. I see very very big malware services have terrible execution rates because the developer designed it for a 64bit system etc.

even calling WriteProcessMemory or SetThreadContext might have look different depending on the expected behavior on the two architectures.

Personally, for malware, a native executable typically in win32 and code modifications are made for 64bit support on the final compiled binary has proven the most stable. However some will argue that .net is just as stable as c, so it really depends on the use-case.

(Mar 30, 2024 08:10 AM)Sajuuk Wrote: [\[CITE\]](#) In terms of development for Windows, we all should've been targeting both x64 and x86 regardless of what architecture we're developing on. Many of the tools I use today offer both 32 and 64-bit versions presumably purely for compatibility reasons.

As far as I can tell, the large majority of Windows users are running 64-bit, so building for x64 makes total sense because of the edge it holds over x86. Since 32-bit apps are backwards compatible and will run on 64-bit systems, it's always wise to compile versions of your project that will run on either.

Targeting only x86 opens up the entire Windows userbase, but you won't be able to use the performance x64 offers. Targeting only x64 cuts your userbase, but by exactly how much in these times I'm not entirely sure. I've watched many of the programs I've used over the years slowly switch over to x64 only, and due to nothing other than typical technological progression.

Targeting as well as actively coding your program to compile & execute in both environments, sadly I have recently seen some stealers that are only compiled for x64. But at the end of the day on 64-bit Windows systems, 32-bit applications are run under WoW64, which emulates a 32-bit environment. As long as the code handles registry/filesystem calls correctly then there should be no issues, but certainly an application can compile and execute, but face errors in functionality if both architectures aren't taken into consideration while coding a project or malware.

(Mar 29, 2024 12:04 PM)nodes Wrote: [\[CITE\]](#) True, true - makes sense

Do you think that when it comes to whitehat, new software there is any reason to use 32? Do you think it's still worth it to compile two versions, 1 for 32 & 1 for 64, or is it better to just go with 64 at this pt?

Why split the files up? runPE allows you to write malware compatible in both environments if you take the considerations of the code into mind.

Infinity. - April 5, 2024, 7:18 am

(Mar 29, 2024 01:39 AM)breaker1911 Wrote: [\[CITE\]](#) 32-bit if you targeting wider support by old device, which still using 32-bit, they're weak

64-bit can run 32-bit app, but 32-bit will crash if you run 64-bit app.

Sorry, bit drunk, but who focuses much on 32-bit apps anymore? I thought people don't do that post COVID? I mean unless you're facing scenarios such as..

**Legacy Software:** Many older applications were developed for 32-bit architectures and may still be in use today. Some organizations continue to rely on legacy software for compatibility or cost reasons.

**Hardware Limitations:** Some older hardware may not support 64-bit computing, necessitating the use of 32-bit applications.

**Resource Efficiency:** In certain cases, 32-bit applications may be preferred for their lower resource demands compared to their 64-bit apps, especially on systems with limited memory or processing

power.

**Compatibility:** While 64-bit systems can run 32-bit applications using compatibility layers like WoW64 on Windows, there may still be instances where organizations or users choose to use native 32-bit applications for compatibility reasons.

**Embedded Systems:** Many IoT devices still uses 32-bit architecture.?

**Hackers** - April 5, 2024, 9:02 am

(Apr 5, 2024 07:18 AM)Infinity. Wrote: [\[CITE\]](#) Sorry, bit drunk, but who focuses much on 32-bit apps anymore? I thought people don't do that post COVID? I mean unless you're facing scenarios such as..

**Legacy Software:** Many older applications were developed for 32-bit architectures and may still be in use today. Some organizations continue to rely on legacy software for compatibility or cost reasons.

**Hardware Limitations:** Some older hardware may not support 64-bit computing, necessitating the use of 32-bit applications.

**Resource Efficiency:** In certain cases, 32-bit applications may be preferred for their lower resource demands compared to their 64-bit apps, especially on systems with limited memory or processing power.

**Compatibility:** While 64-bit systems can run 32-bit applications using compatibility layers like WoW64 on Windows, there may still be instances where organizations or users choose to use native 32-bit applications for compatibility reasons.

**Embedded Systems:** Many IoT devices still uses 32-bit architecture.?

I have recently done a few programs in x86 instead of x64 and it's not much different, easy enough to justify doing it to have more support for a program.

I have a lot of users who have used my free software complain about it not being 32bit compatible, but that's mostly for the programs they are using/crypting, not their actual environment/OS.

**nucleo\_type09** - April 12, 2024, 6:22 pm

Sorry but i don't get it, why should you still use 32 bit systems today? i though i was in 2024 not 2012 ha ha, why don't we just adapt the code within and develop a good 32 bit system adapter so they run both on 32 64/86 ? why is it that hard? :)

also by carefully deciding what DT using while writing, what size a variable should express, and how much memory X will hold at max, i think we all should be fine, problems rise when we don't pay attention to the lowest part of the code where low level operations are involved...

**Hackers** - April 14, 2024, 4:54 am

(Apr 12, 2024 06:22 PM)nucleo\_type09 Wrote: [\[CITE\]](#) Sorry but i don't get it, why should you still use 32 bit systems today? i though i was in 2024 not 2012 ha ha, why don't we just adapt the code within and develop a good 32 bit system adapter so they run both on 32 64/86 ? why is it that hard? :)

also by carefully deciding what DT using while writing, what size a variable should express, and how much memory X will hold at max, i think we all should be fine, problems rise when we don't pay

attention to the lowest part of the code where low level operations are involved...

I think some older systems use 32, but from what I understand there are a lot of tools (especially on HF) that were written as x86/32bit and users today still use them. For example, a lot of users complained that some of my crypters only worked for 64bit, because they were using tools in 32. I agree though that in 2024, people should be considering x64 a baseline and ideally 32bit should be transitioned away from more and more...

I think having a program available for both is still a good use of time, especially if done upfront in a way where 32bit support doesn't bog you down. If you make a purposeful effort to write code that works in both, besides the memory caps and a few other differences, there isn't too much that needs to change in the code, especially if you are using common APIs. That said, this can probably be abstracted and it would be cool for compilers to do more to support both, or to have shared definitions & funcs that are wrappers for both pre-compile, converted to their efficient / specific functions on compile.

**nucleo\_type09** - April 14, 2024, 5:18 am

(Apr 14, 2024 04:54 AM)nucleo\_type09 Wrote: [\[CITE\]](#) I think some older systems use 32, but from what I understand there are a lot of tools (especially on HF) that were written as x86/32bit and users today still use them. For example, a lot of users complained that some of my crypters only worked for 64bit, because they were using tools in 32. I agree though that in 2024, people should be considering x64 a baseline and ideally 32bit should be transitioned away from more and more...

I think having a program available for both is still a good use of time, especially if done upfront in a way where 32bit support doesn't bog you down. If you make a purposeful effort to write code that works in both, besides the memory caps and a few other differences, there isn't too much that needs to change in the code, especially if you are using common APIs. That said, this can probably be abstracted and it would be cool for compilers to do more to support both, or to have shared definitions & funcs that are wrappers for both pre-compile, converted to their efficient / specific functions on compile.

exactly, i think in python you don't have to worry about system architecture and compatibilities, but as c++ programmer you should always stick with x64, even tho a lot of programs are written on x32, those programs can easily be transitioned to x64, by making use of directives and pre-defined headers used to declare data size and capacity, then based on the architecture, the size and capacity will adapt, so you don't have to worry about it, right? you don't even have to have 2 different versions of Y if you implement good c++ tactics...

**Hackers** - April 14, 2024, 5:56 am

(Apr 14, 2024 05:18 AM)nucleo\_type09 Wrote: [\[CITE\]](#) exactly, i think in python you don't have to worry about system architecture and compatibilities, but as c++ programmer you should always stick with x64, even tho a lot of programs are written on x32, those programs can easily be transitioned to x64, by making use of directives and pre-defined headers used to declare data size and capacity, then based on the architecture, the size and capacity will adapt, so you don't have to worry about it, right? you don't even have to have 2 different versions of Y if you implement good c++ tactics...

Yeah, that's the benefit of higher level languages, you can swap systems/servers/env and your code is expected to work across them as long as dependencies are installed. For C++ you have the option of writing code for both with compiletime if statements, or using code that works for both anyways, or writing for 1 specific type. If you go deeper down into PEB/TEB, IAT, and creating PEs in runtime - that's the first time I really had to dive deep on 32 vs 64, but before that the C++ I wrote didn't require much attention for 32/64 differences. With the direct in-memory PE methods though, I was forced to learn more. You can write definition wrappers to make the program or DLL code easy to work with, and abstract away and differences in your includes so that main.cpp (or similar) is clean, and try to limit this as much as possible. The other thing I've found is the 4GB default limit for 32-bit programs, but that hasn't actually affected me in any way just yet.

All in all, agreed with writing for x64 primarily. However, wondering if there are certain use cases where 32 makes more sense, such as with some sort of custom hardware, or legacy platform. Are there enough of these use cases to make 32-bit something worth being a master in (maybe not for me or you, but just in general) ? Or do you think that 32bit is essentially not worth thinking about at all unless asked? The jump from 64 to 32 is really not bad and with AI, oldschool Googling, etc, it's not like someone who knows 64 C++ definitions from memory wouldn't be able to convert to 32 easily

nucleo\_type09 - April 14, 2024, 8:09 am

(Apr 14, 2024 05:56 AM)nodes Wrote: [\[CITE\]](#) Yeah, that's the benefit of higher level languages, you can swap systems/servers/env and your code is expected to work across them as long as dependencies are installed. For C++ you have the option of writing code for both with compiletime if statements, or using code that works for both anyways, or writing for 1 specific type. If you go deeper down into PEB/TEB, IAT, and creating PEs in runtime - that's the first time I really had to dive deep on 32 vs 64, but before that the C++ I wrote didn't require much attention for 32/64 differences. With the direct in-memory PE methods though, I was forced to learn more. You can write definition wrappers to make the program or DLL code easy to work with, and abstract away and differences in your includes so that main.cpp (or similar) is clean, and try to limit this as much as possible. The other thing I've found is the 4GB default limit for 32-bit programs, but that hasn't actually affected me in any way just yet.

All in all, agreed with writing for x64 primarily. However, wondering if there are certain use cases where 32 makes more sense, such as with some sort of custom hardware, or legacy platform. Are there enough of these use cases to make 32-bit something worth being a master in (maybe not for me or you, but just in general) ? Or do you think that 32bit is essentially not worth thinking about at all unless asked? The jump from 64 to 32 is really not bad and with AI, oldschool Googling, etc, it's not like someone who knows 64 C++ definitions from memory wouldn't be able to convert to 32 easily

from my prospective, writing a good c++ program both for 32/64 bit architectures is just as easy as writing just for x64/x86\_64 ones, the only difference is in how you handle DT and DTS, the only thing to keep in mind in my opinion is the max value and the threshold value size for DTs, to which they become problematic for x64 but not x32 architectures, if you know that an int holds 32 bits(4 bytes =>  $2^{31} - 1$  => 2,147,483,647 of max value expressed) amount of space for x32 systems and 64 bits(8 bytes =>  $2^{63} - 1$  => 9,223,372,036,854,775,807 max value exp.) amount of space for x64 systems, then you can easily make the code adapt to those values by using values that can fit into those. The solution is just to think about your code more carefully, let's see some PoC...

```
Code #ifdef __x86_64__
#define INT_THRESHOLD 9223372036854775807
#else
#define INT_THRESHOLD 2147483647
#endif
```

```
int main() {
long int x = 0;
unsigned long long int y = 0;
```

```
while(x < INT_THRESHOLD){
++x;
// do some calculations or some other tasks, like maybe stretching a vector or increment customer
crypto balance..
}
```

```
return 0;
}
```

IDK, let me know what you think about it, to me it makes a lot of sense do it this way, because you keep track of the max value expressed by x and when it reaches the threshold it just stops there, preventing further buffer overflows... :)

**Hackers** - April 14, 2024, 9:18 am

```
(Apr 14, 2024 08:09 AM)nucleo_type09 Wrote: \[CITE\] Code #ifdef __x86_64__
#define INT_THRESHOLD 9223372036854775807
#else
#define INT_THRESHOLD 2147483647
#endif
```

```
int main() {
long int x = 0;
unsigned long long int y = 0;
```

```
while(x < INT_THRESHOLD){
++x;
// do some calculations or some other tasks, like maybe stretching a vector or increment customer
crypto balance..
}

return 0;
}
```

IDK, let me know what you think about it, to me it makes a lot of sense do it this way, because you keep track of the max value expressed by x and when it reaches the threshold it just stops there, preventing further buffer overflows... :)

Yeah this is what I mean basically - writing the code for both systems by having the different definitions/values

And if there are different functions needed because of type mismatches or something else, you can always have a wrapper func that converts a type for the system or does things differently, if you don't want more if/else statements in the main code code.

**SIM-5457022** - April 14, 2024, 12:38 pm

(Mar 28, 2024 09:04 PM)nodes Wrote: [\[CITE\]](#) What do you think of x86 versus x64 programs? For 64-bit it's becoming more popular & modern devices have no issues running them. However, I've noticed many tools are still built with 32-bit.

Personally, I have been doing some more x86 for testing out some DLLs of my own & testing things. However, I usually rock 64-bit. I've fully transitioned (except for few projects I already started) to Visual Studio & it's much, MUCH easier to swap between 32 and 64 builds. I'm liking VS for sure (was using VS Code w/ manual gcc compiles before).

What do you code for mostly?

Here is a snippet from GPT on pros & cons ~ take with a grain of salt for obvious reasons :)

The main differences between compiling a program for x86 (32-bit) vs x64 (64-bit) architectures, focusing on the end result, can be summarized as follows:

#### Addressable Memory:

x86: Limited to addressing up to 4 GB of RAM.

x64: Can address more than 4 GB of RAM, up to 16 exabytes theoretically.

#### Data Types:

x86: Generally, the size of long and pointer types is 32 bits.

x64: Long and pointer types are typically 64 bits, allowing for operations on larger data sets.

#### Performance:

x86: May have slightly faster execution for some 32-bit applications due to smaller data types and less memory usage.

x64: Can offer better performance for applications that require large data types or need to access more RAM, due to a larger number of registers and direct memory access capabilities.

#### Compatibility:

x86: Can run on both 32-bit and 64-bit operating systems, though with the memory limitations of 32-bit architectures.

x64: Requires a 64-bit operating system. Can run 32-bit applications through emulation or compatibility layers, but native 64-bit applications cannot run on a 32-bit OS.

#### Number of General Purpose Registers:

x86: Has 8 general-purpose registers.

x64: Extends the number of general-purpose registers to 16, improving the efficiency of compiled code.

#### Operating System Support:

x86: Widely supported by both modern and older operating systems.

x64: Supported by modern operating systems, with increasingly dominant support over x86 due to its advantages in handling large memory and performance.

## Software Availability:

x86: While still widely supported, newer software versions and features may not be available for 32-bit versions.

x64: Increasingly becoming the standard for software releases, offering more features and better performance.

I had another tool audit the list for accuracy:

Spoiler ([Click to View](#))

x86 is becoming obsolete. X64 is the new system. New intel processors are going to only support x64 and drop support for x86.

The next x64 is the future. the x86 is just there to support older programs.

**Hackers** - April 14, 2024, 12:50 pm

(Apr 14, 2024 12:38 PM)SIM-5457022 Wrote: [\[CITE\]](#) x86 is becoming obsolete. X64 is the new system. New intel processors are going to only support x64 and drop support for x86.

The next x64 is the future. the x86 is just there to support older programs.

Makes sense, hope this becomes more and more true.

Intel is a huge leader, so when they do that there will be 0 point in investing time with 32bit unless someone is paying you to do so. Sounds like there will still be manual ways to use 32bit, but it won't be default, won't be supported fully, and I see here they're saying that 32bit won't have protected mode. I looked up AMD and I guess they already started this process too.

Thank you for saying this - there really is nothing more important than what the processors support, compared to other pros/cons of 32bit.