

What is the "FS"/"GS" register intended for?

Asked 12 years, 3 months ago Modified 2 years, 11 months ago Viewed 131k times



140



So I know what the following registers and their uses are supposed to be:

- CS = Code Segment (used for IP)
- DS = Data Segment (used for MOV)
- ES = Destination Segment (used for MOVS, etc.)
- SS = Stack Segment (used for SP)

But what are the following registers intended to be used for?

- FS = "File Segment"?
- GS = ???

Note: I'm **not** asking about any particular operating system -- I'm asking about what they were intended to be used for by the CPU, if anything.


[assembly](#) [x86](#) [cpu-architecture](#) [cpu-registers](#) [memory-segmentation](#)

Share Follow

edited May 13, 2020 at 0:05

 [Peter Cordes](#)
354k 49 683 932

asked May 30, 2012 at 4:57

 [user541686](#)
209k 133 549 912

- 32 As far as I know, the F and G in these two do not stand for anything. It's just that there was room on the CPU (and in the instruction set) for six user-specifiable segment registers, and someone noticed that besides the "S"tack segment, the letters "C" and "D" (code and data) were in sequence, so "E" was the "extra" segment, and then "F" and "G" just sort of followed. – [torek](#) May 30, 2012 at 5:08
- 3 Could have been, it's always hard to know what was going on in someone else's head unless you were there at the time (and I was on the other coast, nowhere near Intel's design team). – [torek](#) May 30, 2012 at 5:14
- 31 Just think of how much fun we could have had with BS register :-} – [Ira Baxter](#) May 30, 2012 at 5:38
- 5 I always used GS as the "Graphics Segment". :-} – [Brian Knoblauch](#) May 30, 2012 at 11:49
- 3 How about "G"eneral "S"egment? – [S.S. Anne](#) Apr 27, 2019 at 15:24

[Report this ad](#)

6 Answers

Sorted by: Highest score (default)



There is what they were intended for, and what they are used for by Windows and Linux.



156



The original intention behind the segment registers was to allow a program to access many different (large) segments of memory that were intended to be independent and part of a persistent virtual store. The idea was taken from the [1966 Multics operating system](#), that treated files as simply addressable memory segments. No BS "Open file, write record, close file", just "Store this value into that virtual data segment" with dirty page flushing.

Our current 2010 operating systems are a giant step backwards, which is why they are called "Eunuchs". You can only address *your* process space's single segment, giving a so-called "flat (IMHO dull) address space". The segment registers on the x86-32 machine can still be used for real segment registers, but nobody has bothered (Andy Grove, former Intel president, had a rather famous public fit last century when he figured out after all those Intel engineers spent energy and his money to implement this feature, that nobody was going to use it. Go, Andy!)

AMD in going to 64 bits decided they didn't care if they eliminated Multics as a choice (that's the charitable interpretation; the uncharitable one is they were clueless about Multics) and so disabled the general capability of segment registers in 64 bit mode. There was still a need for threads to access thread local store, and each thread needed a a pointer ... somewhere in the immediately accessible thread state (e.g. in the registers) ... to thread local store. Since Windows and Linux both used FS and GS (thanks Nick for the clarification) for this purpose in the 32 bit version, AMD decided to let the 64 bit segment registers (GS and FS) be used essentially only for this purpose (I think you can make them point anywhere in your process space; I don't know if the application code can load them or not). Intel in their panic to not lose market share to AMD on 64 bits, and Andy being retired, decided to just copy AMD's scheme.

It would have been architecturally prettier IMHO to make each thread's memory map have an absolute virtual address (e.g. 0-FFF say) that was its thread local storage (no [segment] register pointer needed!); I did this in an 8 bit OS back in the 1970s and it was extremely handy, like having another big stack of registers to work in.

So, the segment registers are now kind of like your appendix. They serve a vestigial purpose. To our collective loss.

Those that don't know history aren't doomed to repeat it; they're doomed to doing something dumber.

Share Follow

edited Feb 19, 2021 at 16:18



peterh
1

answered May 30, 2012 at 5:15



Ira Baxter
94.8k ● 23 ● 178 ● 351

-
- 12 @supercat: A simpler, more brilliant scheme that would have let them address 65536 times as much storage, would been to have treated the segment registers as full upper 16 bit extension of the lower 16 bits, which is in essence what the 286, 386 and Multics did. – [Ira Baxter](#) Sep 17, 2013 at 21:59
-
- 5 @IraBaxter: The problem with that approach is that 80286-style segments have a sufficiently high overhead than one ends up having to store many objects in each segment, and thus store both segment and offset on every pointer. By contrast, if one is willing to round memory allocations up to multiples of 16 bytes, 8086-style segmentation allows one to use the *segment alone* as a means of identifying an object. Rounding allocations up to 16 bytes might have been slightly irksome in 1980, but would represent a win today if it reduced the size of each object reference from 8 bytes to four. – [supercat](#) Sep 17, 2013 at 22:27
-
- 8 Those registers *are* used in modern operating systems. They're mostly dedicated to point to information about task control blocks, at least in the two major OSes now available for x86 chips. And, since they are no longer "general purpose" even for their original intent, you can't use them for much. Better to pretend on x86-64 systems that they simply don't exist until you need the information they let you access in the thread control blocks. – [Ira Baxter](#) Sep 26, 2014 at 15:11
-
- 7 The appendix analogy is really bad based on outdated science; it's related to the immune system, so definitely *not* "vestigial". It detracts from the actual post. Other than that, it's a good response. – [code_dredd](#) Dec 6, 2015 at 8:16
-
- 12 Thanks for the amusing, no-holds-barred treatment of segmented vs flat memory :) Having also written code on 6809 (with and without paged memory), 6502, z80, 68k and 80[123]?86, my perspective is that segmented memory is a horror show and I'm glad it was consigned to the dustbin of history. The use of FS and GS for efficient access of thread_local data is a happy unintended consequence of an historical error. – [Richard Hodges](#) Jan 4, 2018 at 12:41
-


64


The registers `FS` and `GS` are segment registers. They have no processor-defined purpose, but instead are given purpose by the OS's running them. In Windows 64-bit the `GS` register is used to point to operating system defined structures. `FS` and `GS` are commonly used by OS kernels to access thread-specific memory. In windows, the `GS` register is used to manage thread-specific memory. The linux kernel uses `GS` to access cpu-specific memory.

Share Follow

edited Aug 25, 2016 at 18:54

answered May 30, 2012 at 5:06

 Johan

 cytinus

75.6k

25

197

329

5,554

8

37

47

1 Were they intended to be used for OS-defined purposes, or to facilitate code which needs to do something like `*dest++ = lookup[*src++]`; which would otherwise be rather awkward if dest, lookup, and src were at three unrelated locations. – [supercat](#) Sep 17, 2013 at 20:23

10 On Windows FS is indeed for thread specific storage. See documented map of the block pointed by FS here en.wikipedia.org/wiki/Win32_Thread_Information_Block – [Nedko](#) Aug 21, 2015 at 11:28

2 It's not just on Windows. GS is also used for the TLS on OS X. GS is also used by 64bit kernels to keep track of system structures during context switches. The OS will use SWAPGS to that effect. – [E.T](#) Aug 9, 2017 at 22:57

"In windows, the GS register is used to manage thread-specific memory"... isn't it FS ? – [tuket](#) May 21, 2021 at 8:01

@tuket their 32-bit os uses fs and their 64-bit os uses gs. linux did the opposite move. – [Johan Boulé](#) Sep 2, 2021 at 15:04

26

FS is used to point to the thread information block (TIB) on windows processes .

one typical example is [\(SEH\)](#) which store a pointer to a callback function in `FS:[0x00]` .

GS is commonly used as a pointer to a thread local storage (TLS) . and one example that you might have seen before is the stack canary protection (stackguard) , in gcc you might see something like this :

```
mov    eax,gs:0x14
mov    DWORD PTR [ebp-0xc],eax
```

Share Follow

edited Sep 25, 2018 at 5:35

answered Sep 25, 2018 at 4:51

 zerocool

3,454

2

28

42

3 This doesn't actually answer the question. The question states *Note: I'm not asking about any particular operating system -- I'm asking about what they were intended to be used for by the CPU, if anything.* – [Michael Petch](#) Sep 25, 2018 at 5:27

22 @MichaelPetch ya i know i just want to add this as good info for those who read this q/s in SO – [zerocool](#) Sep 25, 2018 at 5:29

13

TL;DR;

What is the "FS"/"GS" register intended for?

Simply to access data beyond the default data segment (DS). Exactly like ES.

The Long Read:

So I know what the following registers and their uses are supposed to be:

[...]

Well, almost, but DS is not 'some' Data Segment, but the default one. Where all operation take place by default (*1). This is where all default variables are located - essentially `data` and `bss`. It's in some way part of the reason why x86 code is rather compact. All essential data, which is what is most often accessed, (plus code and stack) is within 16 bit shorthand distance.

ES is used to access everything else (*2), everything beyond the 64 KiB of DS. Like the text of a word processor, the cells of a spreadsheet, or the picture data of a graphics program and so on. Unlike often assumed, this data doesn't get as much accessed, so needing a prefix hurts less than using longer address fields.

Similarly, it's only a minor annoyance that DS and ES might have to be loaded (and reloaded) when doing string operations - this at least is offset by one of the best character handling instruction sets of its time.

What really hurts is when user data exceeds 64 KiB and operations have to be commenced. While some operations are simply done on a single data item at a time (think `A=A*2`), most require two (`A=A*B`) or three data items (`A=B*C`). If these items reside in different segments, ES will be reloaded several times per operation, adding quite some overhead.

In the beginning, with small programs from the 8 bit world (*3) and equally small data sets, it wasn't a big deal, but it soon became a major performance bottleneck - and more so a true pain in the ass for programmers (and compilers). With the 386 Intel finally delivered relief by adding two more segments, so any series [unary](#), [binary](#) or [ternary](#) operation, with elements spread out in memory, could take place without reloading ES all the time.

For programming (at least in assembly) and compiler design, this was quite a gain. Of course, there could have been even more, but with three the bottleneck was basically gone, so no need to overdo it.

Naming wise the letters F/G are simply alphabetic continuations after E. At least from the point of CPU design nothing is associated.

*1 - The usage of ES for string destination is an exception, as simply two segment registers are needed. Without they wouldn't be much useful - or always needing a segment prefix. Which could kill one of the surprising features, the use of (non repetitive) string instructions resulting in extreme performance due to their single byte encoding.

*2 - So in hindsight 'Everything Else Segment' would have been a way better naming than 'Extra Segment'.


*3 - It's always important to keep in mind that the 8086 was only meant as a stop gap measure until the [8800](#) was finished and mainly intended for the embedded world to keep 8080/85 customers on board.

Share Follow

edited Sep 17, 2021 at 0:48

answered May 12, 2020 at 23:05

 **Peter Cordes**
354k ● 49 ● 683 ● 932

 **Raffzahn**
341 ● 3 ● 8

2 Wow, thank you for explaining all this! This explains a lot and makes so much sense! +1 – [user541686](#) May 12, 2020 at 23:14

▲
5 According to the Intel Manual, in 64-bit mode these registers are intended to be used as additional base registers in some linear address calculations. I pulled this from section 3.7.4.1 (pg. 86 in the 4 volume set). Usually when the CPU is in this mode, linear address is the same as effective address, because segmentation is often not used in this mode.

▼
So in this flat address space, FS & GS play role in addressing not just local data but certain operating system data structures (pg 2793, section 3.2.4) thus these registers were intended to be used by the operating system, however those particular designers determine.

🔖
🔄 There is some interesting trickery when using overrides in both 32 & 64-bit modes but this involves privileged software.

From the perspective of "original intentions," that's tough to say other than they are just extra registers. When the CPU is in *real address mode*, this is like the processor is running as a high speed 8086 and these registers have to be explicitly accessed by a program. For the sake of true 8086 emulation you'd run the CPU in *virtual-8086 mode* and these registers would not be used.

Share Follow

edited Apr 26, 2019 at 11:03

answered Apr 26, 2019 at 2:04



1



The FS and GS segment registers were very useful in 16-bit real mode or 16-bit protected mode under 80386 processors, when there were just 64KB segments, for example in MS-DOS.

When the 80386 processor was introduced in 1985, PC computers with 640KB RAM under MS-DOS were common. RAM was expensive and PCs were mostly running under MS-DOS in real mode with a maximum of that amount of RAM.

So, by using FS and GS, you could effectively address two more 64KB memory segments from your program without the need to change DS or ES registers whenever you need to address other segments than were loaded in DS or ES. Essentially, [Raffzahn has already replied](#) that these registers are useful when working with elements spread out in memory, to avoid reloading other segment registers like ES all the time. But I would like to emphasize that this is only relevant for 64KB segments in real mode or 16-bit protected mode.

The 16-bit protected mode was a very interesting mode that provided a feature not seen since then. The segments could have lengths in range from 1 to 65536 bytes. The range checking (the checking of the segment size) on each memory access was implemented by a CPU, that raised an interrupt on accessing memory beyond the size of the segment specified in the selector table for that segment. That prevented buffer overrun on hardware level. You could allocate own segment for each memory block (with a certain limitation on a total number). There were compilers like Borland Pascal 7.0 that made programs that run under MS-DOS in 16-bit Protected Mode known as DOS Protected Mode Interface (DPMI) using its own DOS extender.

The 80286 processor had 16-bit protected mode, but not FS/GS registers. So a program had first to check whether it is running under 80386 before using these registers, even in the real 16-bit mode. Please see an [example of use of FS and GS registers a program for MS-DOS real mode](#).

Share Follow

edited Feb 5, 2021 at 13:29

answered Feb 4, 2021 at 23:17



Maxim Masiutin

4,507 • 7 • 63 • 85