

## Re: Segmentation in real mode

**Source:** <http://coding.derkeiler.com/Archive/Assembler/alt.lang.asm/2005-03/0436.html>

---

**From:** Beth (*BethStone21\_at\_hotmail.NOSPICEDHAM.com*)

**Date:** 03/12/05

Date: Sat, 12 Mar 2005 11:35:50 GMT

Tom wrote:

> *Hello,*

Hi :)

> *For some time now I have been learning assembly on my own using  
> Webster. So far I have managed to build a small paper-and-pencil uP and  
> read/comprehend most of the material from the 16-bit DOS version of AoA  
> (I understand DOS is obsolete, but I have found it simpler to start  
> with, hope to do better in the future).*

Cool :)

> *Would you be kind enough as to help me clarify the following points  
> (regarding real mode programming):  
>  
> 1. Why do segment registers hold segments' paragraph numbers rather  
> than their absolute addresses?*

Well, 16-bits (the size of the original 8086's registers) can only address 64KB maximum ( $2^{16} = 65536$  bytes or 64KB)...some means is, therefore, required to "extend" the accessible range to beyond the 64KB that 16-bits alone can address...

As to why Intel used this particular method of doing it rather than some other schemes...well, goodness knows why...it doesn't seem particularly sensible to me at all...I've got a post elsewhere here exactly proposing that it probably would have made more sense to do it in a completely different way...but, well, regardless, this IS how Intel chose to do it, so we have to live with it, whatever the "why?" for choosing this particular method was...

Though, some kind of method DID need to be chosen to get passed that 64KB "barrier" that only using 16-bits to address memory creates...as, simply, with 16-bits only for an "address", you can only access 64KB...it's a simple physical thing: There just is not enough bits to represent any number larger than 65535 with only 16-bits...to address further, we need more bits (note, of course, that the original 8086 chip was a 16-bit

## alt.lang.asm: Re: Segmentation in real mode

chip...the problem, indeed, is "solved" when the '386 showed up by using 32-bits – as well as taking advantage of its more complicated "segmentation" scheme in "protected mode" – for addresses...but this didn't apply to the original 16-bit 8086 and what we see in "real mode" is the CPU operating by the original 8086 rules in this CPU mode for "compatibility")...

So, basically, 16-bits is not enough because that can only address 64KB...we need more bits to address further...now, logically enough, if registers are only 16-bits then, simply, to get more bits in an address, we need more than one register...for this purpose, Intel introduced the "segment registers" ...and full absolute addresses are made by a combination of "segment" and "offset" \_together\_...

Hence, that at least answers one of your questions: The segment registers can't hold full absolute addresses because, simply, they are only 16-bits in size and that's not big enough to address more than 64KB...the "segment registers" are NOT used alone...they are the "top part" of a "segment:offset" absolute address (also known as a "far" address)...

Now, personally, I reckon it would have made the best sense to simply use the "segment registers" as if they were just an "extra 16-bits" (or even just "an extra 4 or 8 bits" or something) that's "stuck in front" of the "offset" to get the absolute address...elsewhere here, I've written a post making the "case" for why this probably would have been better...in the long run...and, yes, with the benefit of "hindsight" (which, to be fair, the Intel engineers didn't have...for example, they had no idea that the PC would later appear and start using this chip...nor that PCs would "swamp" the competition and basically obliterate all other desktop machine architectures, other than the Apple Mac...indeed, the last chip they made – the 4004 – was, in fact, the first microprocessor and was used for things like "calculators"...this was among the first 16-bit microprocessors (\*ahem\* I'll phrase it that way to avoid the "contraversy" that the group had over whether it was "first" or merely "one of the first"... "among the first" could also mean "first" too ;) ever made...well, the real reason is probably simply that they weren't "in that mindset"...weren't thinking along these kinds of lines at the time...the "weight" of the decision was not apparent at the time they were making it :))...

BUT that, of course, is another matter...a game of "what if"...whatever the reason they chose the particular kind of "segmentation" that they did eventually chose, we have to live with that...

And, simply, Intel thought to use these "segment registers" just to add an extra 4 bits onto the addressable range – making it 20-bit, which is a 1MB range – and did so by simply "biasing" the "segment register" by 4 bits and then adding "segment" and "offset" together...that is:

$$\text{Absolute address} = (\text{segment} * 16) + \text{offset}$$

...which then "extends" that 64KB range, up to around 1MB, by effectively adding on an extra 4 bits to give a 20-bit absolute address (actually, because of the use of addition, it's a tiny bit further than a 20-bit address but this was used to no great purpose)...

My "best guess" as to why Intel chose this scheme – which is a bit "brain-dead" in many regards – is to remember one other small but significant fact: CPUs typically weren't in "CPU families" at this time...the 8086 is similar to their previous 4004 chip, for example, but it's not "fully backwards compatible" with it...it was typical that each chip was, in fact, created afresh with a new design (perhaps "based" on a previous design but not made "100% backwards compatible" with that design, to start off a "CPU family")...

Hence, the probable reason (my "best guess", anyway) as to why Intel chose this particular scheme? They were simply NOT thinking of any "future extension" at all...while designing this, they presumed that they'd do as they'd always mostly done previously in simply creating a brand new CPU design...you know, they created the 4004 then they created the 8086...then, some years later, they could create some brand new 16016 design or whatever...

And this was the "mindset" at the time...hence, the 8086 itself did not require anything more than 20-bit addresses and they thought they'd use this scheme to OPTIMISE...with absolutely no regard to it in any way being "extended" in the future...in that context, it can be seen to make sense...if there was no prospect of any '286, '386, '486, etc. "family" following on from the chip, then, instead, you'd concentrate on how to make what you have got more "optimal"...they decided they wanted a chip with a 1MB address space and then they HARD-WIRED it to that...

This, of course, proved a bad idea when, later, IBM decide to put the 8086 into their PC designs...and then "desktop" machines go from strength to strength taking over the world...at this point, Intel now realise that they are going to have to maintain "backwards compatibility" and "expand" this range of chips to meet the PC user's requirements (the PC user still wants to be able to run their older software just fine but, yes, if Intel improved things then they'd "upgrade")...the "CPU family" was born and, simply, one aspect of "backwards compatibility" is that you can only ADD, not modify, not revise, not subtract...all the previous features have to be maintained "as is"...even if, indeed, they were actually "a bit crap" or "we'd never have done it this way, if we'd only known!"...it has to be maintained fully so that older software still runs exactly as intended on the newer chips...

It's interesting to note that the 80286 immediately attempted to "correct" the problem (note: There was a 80186 chip but this was never intended or used in PC systems...so, in PC terms, the 80286 was the "effective successor" to the 8086 :)...I think, basically, Intel knew it wasn't good...that because the scheme chosen is "hard-wired" to 20-bits and 1MB, it couldn't be "extended"...this had simply been an "optimisation" kind of

thing specifically for their 8086 design, which they had intended to have a 1MB "roof" (so, this limitation was designed into the original 8086 chip)...it was a design with no "forethought" because, simply, at the time, Intel did not see that it needed to apply any forethought...you know, at the time of design, they did not see a long "family" of "backwards compatible" chips that would be central to a dominant "desktop" architecture that is still going strong as it approaches its 30th anniversary, let's remember...you know, this design is nearly as old as I am...chips have never lasted that long beforehand (well, they wouldn't have, would they? That would have been in the '40s and computers themselves didn't really exist...there was the "enigma" stuff, of course, but that was – especially by our standards today – more like a "complex machine" than a "computer"...and it was for a particular war-time purpose, not too much time spent envisaging a world full of these things...indeed, there's an infamous mis-prediction quote that was spoken \_decades later\_ that still shows the "mindset" that people were operating by: "I don't see a need for more than 5 or 6 computers world-wide"...oh boy, does the person who said that now look dumb! Ah, to be fair, of course, computers were room-sized, highly expensive and not particularly useful but for "complex calculations"...with \_those\_ kinds of computers, there, indeed, wouldn't have been much need for them :)...)

It was simply a "mindset" thing, I think...remembering that this was among the first – in fact, if not \_the\_ first – architecture that went on to become a "CPU family"...hence, you know, they weren't thinking about how the chip could be "expanded" because no-one at the time ever did "expand" chips...they just started afresh with a new and better design...the "situation" of being placed into the PC was what changed that...the PC was "too big" and relied upon by too many customers with too much "old software" that they needed to run (and these were mostly "business customers" too, of course...so, you know, lots of money, little sense, a reluctance to arbitrarily "upgrade" unless they could \_SEE\_ a definite advantage to doing it, etc. :) to go around messing with changing the chip inside it...

It was a "backwards compatible" nightmare because, simply, "backwards compatible" (chip-wise) as more or less \_invented\_ for these chips...and invented with the \_next\_ generation, not the original design...the original design have absolutely \_NO\_ concern whatsoever with "future expansion"...at least, not in the way it eventually went...

And, if you reconsider the 8086 design from the perspective of an engineer who's NOT thinking that their design would last any further than this one chip (which, by design, was specified only to need around 1MB "address space", so you really didn't give a crap beyond that)...well, the way the "segmentation" stuff works on the 8086 could be seen as a kind of "optimisation" for that chip...just like being able to use both "near" and "far" addresses, which saves a byte here and there...

> 2. In Chapter 6 of AoA (*The instruction set, 6.11.1 Simple arithmetic*  
> 1) Mr. Hyde gives an example of code in which he declares a procedure

## alt.lang.asm: Re: Segmentation in real mode

- > *MAIN*, which is duly terminated with "Main endp". What does the second
- > "end Main" expression stand for? Why is this procedure not called? Or
- > is it?

Right...basically because "endp" is for ending the procedure..."end" is for ending the source file...different purposes (though, admittedly, the way they read, it does seem very odd indeed...the way the "end" directive specifies the `_start_` of all things is, indeed, a rather confusing facet of this kind of syntax...the NASM assembler (and others similar to it) don't bother with the "end" directive at all because it is kind of confusing and seems "unnecessary" with modern machines to have to put "end" at the end of the source file, rather than simply using the "end of file" as the end :).)

In generic terms:

`<label> endp` – ends procedure called "label"

`end [<label>]` – ends source file and optionally specifies the "entry-point" – start address – for execution of the code

Simply, the two serves completely different purposes...and it's just "confusing phrasiology" that it does, indeed, `_LOOK_` like they seem to be saying the same thing...just a poor choice of name for the directives, really...yes, as weird as it is, "end Main" is NOT actually saying "end of main", it's saying "end of source file...oh, by the way, start the program at Main"...but, yeah, it sure doesn't "read" that way, does it? Just a "quirk" of this style of syntax...

Beth :)