

**Information Security - II**  
**Prof. V. Kamakoti**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Madras**

**Lecture – 10**  
**Architectural Aid to Secure Systems Engineering**  
**Session – 9: X86 ISA – Part 2**

One of the most important register, but more mundane thing to study especially at the end of the day is this e flags, extended flag register.

Flags are very important. As I told you in one of the examples there, where I made a jump, jump less than or equal to do you remember in one of the session and that is because of the flags. I do an operation and the result of the operation not the data part, the characteristics of the result of the operation is captured in the flag. I do compare of two numbers, if the number on the left hand side is less than the number of the right hand side, the less than flag is set. If it is equal, the Zero Flag is set. If greater, greater flag is set. So, the whole program, many of the programs we write is very complex.

How do you characterize complexity in programming? Lot of logic's statement that we do, lot of K's statement that you do, lot of nested if that you do; this is how your logic is getting complex. All straight line programs this instruction next, next, next all sought of numerical computation those are not complex to the mind. It is complex to the system, because it needs to do lots more arithmetic operations. The logic operations actually determine the complicity of a program and all these logic operations basically get translated to conditional branch instructions, if j is greater than k max is equal to j l max is equal to k actually translated into a conditional branch instruction as we saw in one of the previous sessions. And these conditions are basically evaluated by this by seeing the e flags register.

(Refer Slide Time: 02:03)

## Other Registers

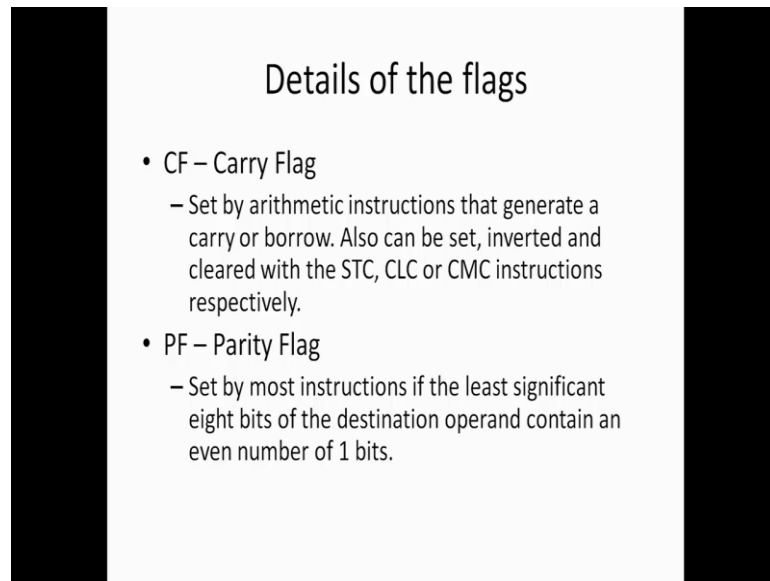
- EFLAGS – 32 Bit Register

VM	RF	NT	IO	IO	OF	DF	IF	TF	SF	ZF	AF	PF	CF
			PL	PL									

Bits 1,3,5,15,22-31 are RESERVED.  
18: AC, 19:VIF, 20: VIP, 21:ID

So, at the end of every arithmetic operation the flags get updated and the subsequent conditional jump instructions look at this flag. So, that is one of the precise reasons, why we want to spend some time to basically talk about this e flags register, Now, in this e flags register, there are 32 bits read from right to left and the bits one it is number 0 to 31 the bits 1, 3, 5, 15, 22 to 31 are reserved. We do not know what it is intended for, as of now as programmers we do not find anything there. The 18th bit is called alignment check, the 19 bit is called VIF, 20th is VIP 21 is I d and then all the other things are here. So, let us basically look at some of this bit.

(Refer Slide Time: 02:59)

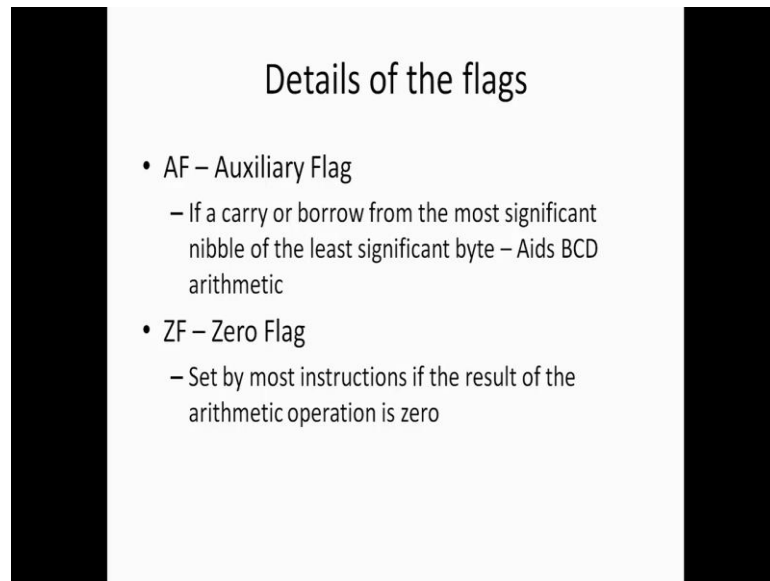


### Details of the flags

- CF – Carry Flag
  - Set by arithmetic instructions that generate a carry or borrow. Also can be set, inverted and cleared with the STC, CLC or CMC instructions respectively.
- PF – Parity Flag
  - Set by most instructions if the least significant eight bits of the destination operand contain an even number of 1 bits.

So, there is a bit for carry, this is called a Carry Flag. This is set by arithmetic instructions that generate a carry or borrow and this is basically necessary sometimes to detect Over Flow under flow etcetera. It can be said inverted and cleared by what you called as set carry instruction, clear carry and a complement carry. And there is also something called a Parity Flag. Parity Flag, which is set by most instruction; if the least significant 8 bits of the destination operand contain an even number of ones, This is this is a parity flag and this is very important for some error checking and other type of activities.

(Refer Slide Time: 03:42)



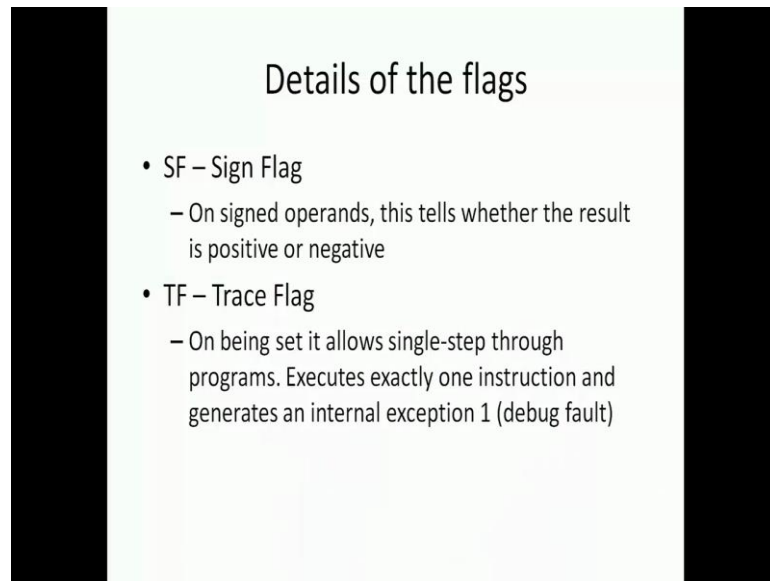
### Details of the flags

- AF – Auxiliary Flag
  - If a carry or borrow from the most significant nibble of the least significant byte – Aids BCD arithmetic
- ZF – Zero Flag
  - Set by most instructions if the result of the arithmetic operation is zero

There is Auxiliary Flag.

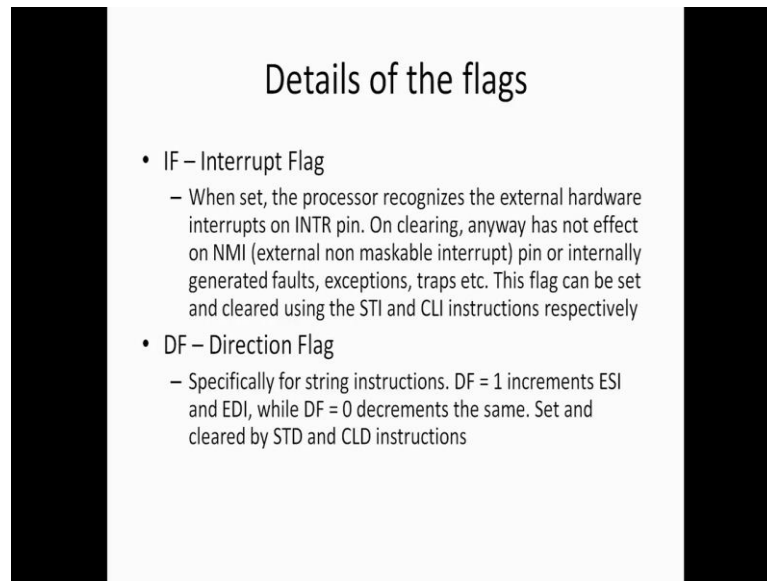
If a carry or borrow from the most significant nibble of the least significant byte. So, the least significant byte is 2 nibbles the most significant nibble is the later part of the least significant byte and if there is carry it will be set. And where do you use this? We use it in BCD arithmetic. And why is BCD arithmetic very important? Because, all your things are binary coded decimals. Many, many software's work on binary coded decimals. You know what is binary coded decimal? So, you do from so, all your thing is not hexadecimal right. Your excel does work in hexadecimal, right. You go mad if you start using reading hexadecimal numbers or binary numbers. So, you are still accustomed to binary decimal arithmetic and so, how do you perform decimal arithmetic using binary coded decimals and this is BCD is very, very important. The Zero Flag is set by most instructions if their result of the operation is 0.

(Refer Slide Time: 04:49)



There is Sign Flag which gives you whether the signed operands, this tells whether the result is positive or negative and that is how we do compare. So, compare how do you compare a and b? You subtract a and b; if the result is negative a is less than b result is 0 a is equal to b, result is greater than 0 if the result is positive it is great. So, the Sign Flag is also very important and then these are all flags which are basically set by operations and you will you will go and find some characteristic attribute of the result. Now there are flags which are used to control your execution. For example, Trace Flag on being set it allows single step though programs, you want to do single step though. So, what it does? You allow, you to go one step then it will take you to interrupt service routine. There you go and do something you check what are things again it will assume back and do. So, this allows you it generates an interrupt at the end of every instruction. So, that you could go and find out what that instruction is doing to essentially these are all use for debugging.

(Refer Slide Time: 05:58)



### Details of the flags

- IF – Interrupt Flag
  - When set, the processor recognizes the external hardware interrupts on INTR pin. On clearing, anyway has not effect on NMI (external non maskable interrupt) pin or internally generated faults, exceptions, traps etc. This flag can be set and cleared using the STI and CLI instructions respectively
- DF – Direction Flag
  - Specifically for string instructions. DF = 1 increments ESI and EDI, while DF = 0 decrements the same. Set and cleared by STD and CLD instructions

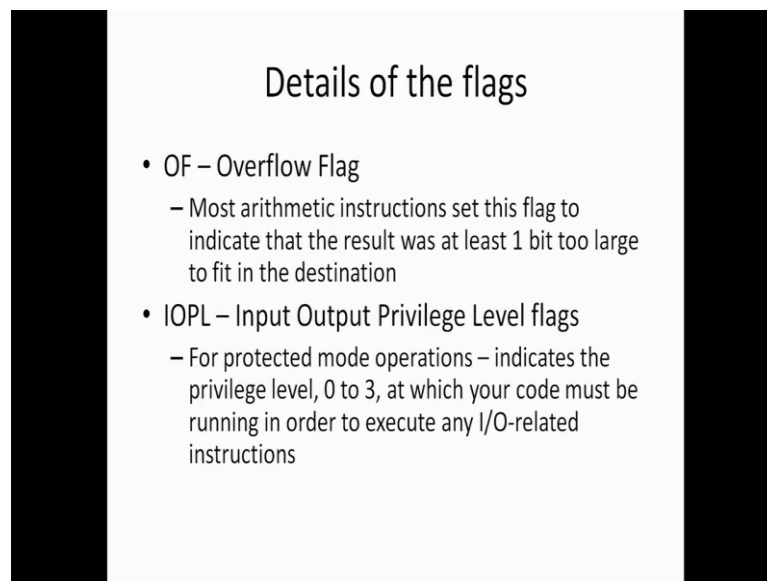
So, there is another Interrupt Flag, which is basically when said the processor recognizes the external hardware interrupts on INTR pins. If you if you if you reset it, if you clear it it has no affect at all. So, even if you interrupt nothing will happen correct. So, the simple way by which somebody can hack into your system is to go and reset this clear this Interrupt Flag. Then you go and do any interrupt accept all non maskable every external non maskable interrupts, correct? It will never recognize, the system will do some execution then you go and set it back then it can recognize. So, there are very, very important flag from a security prospective also this is very important. So, this flag can be set and cleared using STI and CLI instruction.

So, slowly let us try and understand every flag from a security prospective also right what will happen if I disable all interrupts? You lock that system from the user correct it is a denial of service right then. So, that is one of the reason while we will see later that your STI and CLI, the set interrupt and clear interrupt are call privileged instructions they can be executed only when the processor is executing at the highest privilege. We will talk about it as we go right and when it is in the highest privilege. So, if you get a privilege gain. Then the first thing you will go is to go and screw this interrupt then you are isolated then you do whatever you want. So, please understand the thing. So, so this

is what this course is adding. So, all these things are available in Intel developer manual. What is it that you are doing? know you will ask me this question.

This extra masala, what I am adding here from a security prospective. So, there is something called a direction flag specifically for string instructions. So, one flag is dedicated for string instruction. So, I said right, there is a segment register there is an offset. There is another segment register another offset and a counter. So, from that segment register base that offset to this segment register this offset, copy this one by x should I copy it in the forward direction should I copy it in the backward directions. That is a question This Direction Flag will tell you. This direction flag will either will tell you it will increment or it will decrement. It will tell you whether in this direction should I go or in the opposite direction I go. So, that is also very, very interesting.

(Refer Slide Time: 08:52)



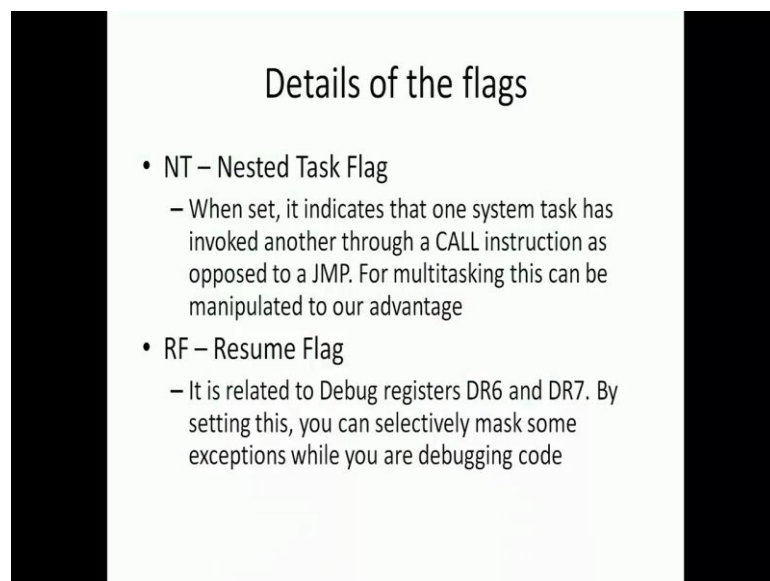
**Details of the flags**

- OF – Overflow Flag
  - Most arithmetic instructions set this flag to indicate that the result was at least 1 bit too large to fit in the destination
- IOPL – Input Output Privilege Level flags
  - For protected mode operations – indicates the privilege level, 0 to 3, at which your code must be running in order to execute any I/O-related instructions

And there is a Over Flow flag is very, very important. This is something again arithmetic instructions will set this flag to indicate that the result was at least one bit too large to fit in the destination. IOPL this is also a very, very important flag. So, this is set by the operating system. When I want to do an I-O, right? So, the processor will execute in four privilege levels 0, 1, 2, 3 why? 4 we will talk about it in subsequent session. 4 privilege levels means, how many bits I need to represent this 4 privilege levels? 2 bits I need. So,

the bits I set here if I set it has 3; that means, any process can use it 0, 1, 2, 3 but, if I set it has 2 then the process at privilege level 3 cannot do any I o at all. So, for protected mode operation indicates the privilege level 0 to 3 at which your code must be running in order to execute any I o related instructions. So, meaning that and below. So, 2 means 2 1 0, 3 means 3 2 1 0, 1 means 1 and 0, 0 means 0. So, this is very, very important for you to understand here.

(Refer Slide Time: 10:17)



### Details of the flags

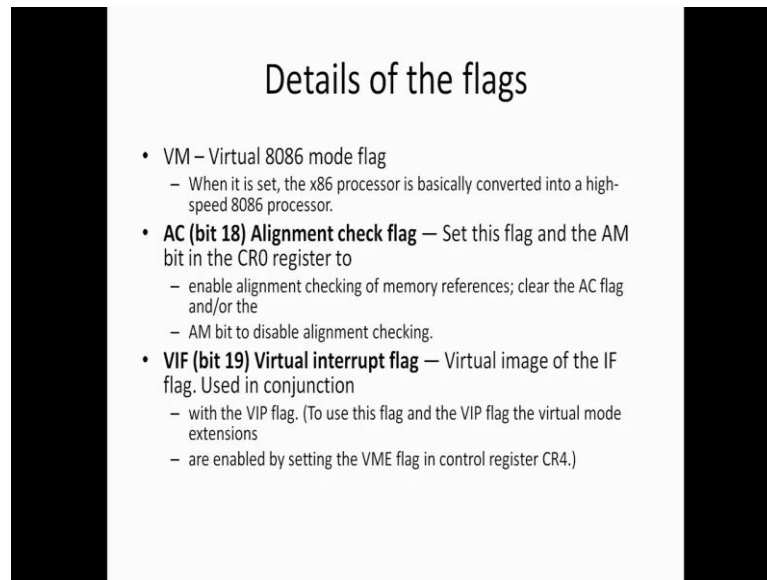
- NT – Nested Task Flag
  - When set, it indicates that one system task has invoked another through a CALL instruction as opposed to a JMP. For multitasking this can be manipulated to our advantage
- RF – Resume Flag
  - It is related to Debug registers DR6 and DR7. By setting this, you can selectively mask some exceptions while you are debugging code

And then, there is something called a Nested Task Flag. When set it indicates that one system task as invoked another through a call instruction as suppose to a jump. So, what is nested task I call you after you finish test come to me, not to something else. What is jump I call you then you execute there is no necessity for you to come back. So, that is a difference between jump and call and this nested for multitasking this can be manipulated is very important for Nested Task. This flag and there is called something called Resume Flag it is related to d burg registers, DR6 and DR7 by setting this you can selectively mask some exceptions while you are debugging code. This is basically used for debugging, but then can we exploit this to gain some information about the code. So, I have some code. For example, I have a licensed code and it is accessing some license file correct, to validate that it can execute can I use this debug registers to go and get



some details of how to break this license. So, very interesting exercise, right, there are some positive answers to that question.

(Refer Slide Time: 11:28)



### Details of the flags

- **VM – Virtual 8086 mode flag**
  - When it is set, the x86 processor is basically converted into a high-speed 8086 processor.
- **AC (bit 18) Alignment check flag** – Set this flag and the AM bit in the CR0 register to
  - enable alignment checking of memory references; clear the AC flag and/or the
  - AM bit to disable alignment checking.
- **VIF (bit 19) Virtual interrupt flag** – Virtual image of the IF flag. Used in conjunction
  - with the VIP flag. (To use this flag and the VIP flag the virtual mode extensions
  - are enabled by setting the VME flag in control register CR4.)

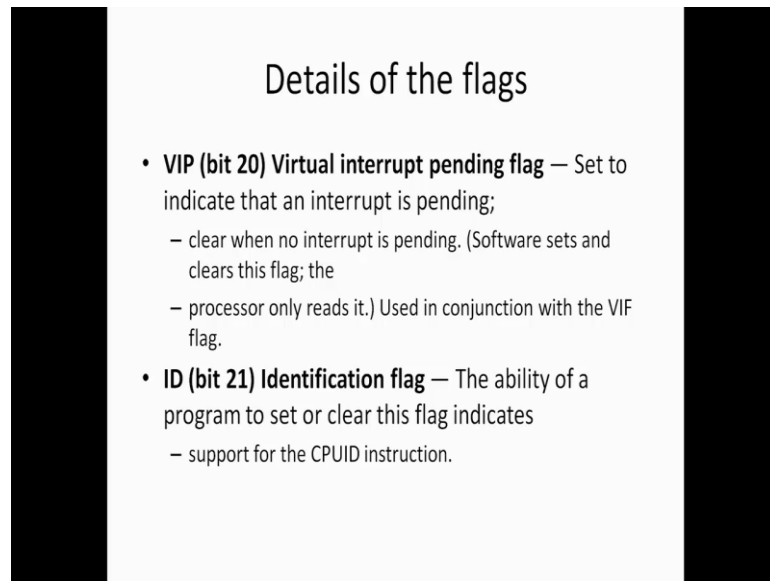
Then, there is Virtual 8086 mode I think nobody uses it but, still it exists for this thing. Now, comes two more flags that are dictating about one flag that is dictating about performance it is called an Alignment Check. So, when you look at a complex instruction set computer it has 8 bit appearance, 16 bit appearance, 32 bit appearance. From the memory when I am getting instruction, I read it as 4; 4 bytes when I get data I read it as 4 bytes, 4 byte junk. So, if I want to read byte 1, I read 0, 1, 2, 3 I cannot just read only 1. If I want to read by 2, what do I read 0, 1, 2, 3 only, in that 4 byte segment to which it belongs, I read all those 4 bytes if I want to read by 3 then I read 0, 1, 2, 3. So, suppose my byte is stored. I now want to read a double word which is 4 bytes and that byte is stored in 2, 3, 4 and 5 memory locations. So, to read this what will I read, first I have to read 0, 1, 2, 3 then 4, 5, 6, 7 So, to read one operand for me it takes 2 memory reads. You understand this? Please understand this. So, always the data that comes from the memory to the cash is 4 byte blocks. So, some instructions can become really slow. Do you understand this because instead of reading one byte I am reading 2 bytes. So, essentially it becomes slow.

So, if you had stored that byte as 0, 1, 2, 3 instead of 2, 3, 4, 5 the worst case is 1, 2, 3, 4 say I did that, if I had done it like that then it would have been very easy. So, imagine I have 1 million variables and every variable is not aligned then what will happen, your memory performance actually screws up very badly right. So, this is and they are on different locations quite spread that. So, this becomes an extremely complex. So, what do we do is called there is a flag called Alignment Check. That Alignment Check essentially tells if it can talk about bit aligned, it can talk about word aligned but here it talks about word align if it is not in this 32 bit boundary that is it should be if a variable starts it should be 0, 1, 2, 3 or 4, 5, 6, 7 or it cannot be between half of this and half of this, if it is going to be in the half it immediately give us an exception.

So, when you compile the code you compile the code; make the Alignment Check Flag as 1 and run this code. And wherever you are finding this misaligned access it will flag an exception then you go and do some changes. So, that you align it finally, you get an aligned code and the movement you have an aligned code your performance will be very fast. What do you mean by performance tuning of architectural level performance tuning of your code this is what you mean by that, right. So, you see to that that the alignment is proper so that when the code executes no memory fetch is actually become 2 fetch it becomes a single fetch. So, this a way by which you can do performance tuning on your code. That is the alignment check flag and you go and clear it after everything is done you can clear it. Then but your code will really become faster. So, what do you lose by this? If I have some 2 byte variable if I have 4, 2 byte variables in a non-align setup I can put it in 8 bytes in an aligned word setup I should put it in 16 bytes. Your memory actually doubles.

So, I go back to my communist science, you lose something you gain in performance you lose in memory. You gain in memory, you lose in performance. Are you able to follow this bit 19 is called a Virtual Interrupt Flag, which a virtual image of the Interrupt F flag this is used in conjunction with the VIP flag. So, there is something called VME flag in the controlled register 4. So, there are. So, many things, but I should tell you that all these things are much related to what you call as the system management mode. This gets so. We will talk about s m m possibly sometime down the line.

(Refer Slide Time: 16:35)

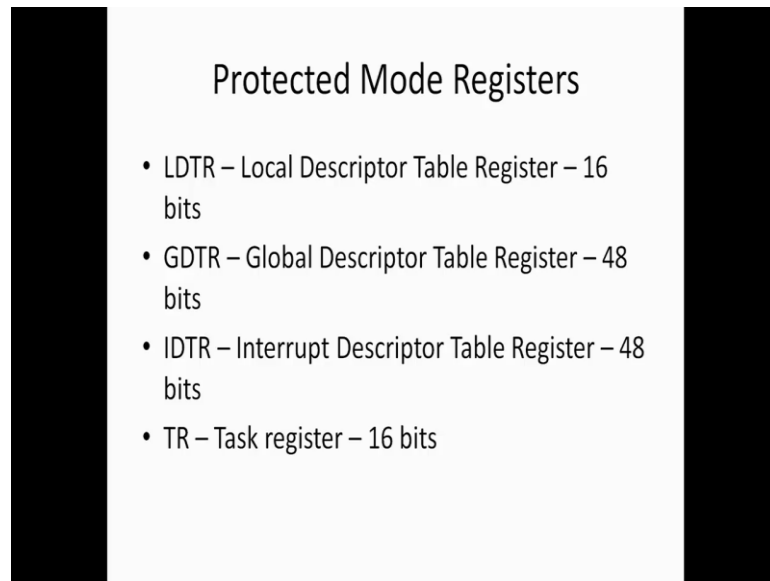


### Details of the flags

- **VIP (bit 20) Virtual interrupt pending flag** — Set to indicate that an interrupt is pending;
  - clear when no interrupt is pending. (Software sets and clears this flag; the
  - processor only reads it.) Used in conjunction with the VIF flag.
- **ID (bit 21) Identification flag** — The ability of a program to set or clear this flag indicates
  - support for the CPUID instruction.

Probably part of this course or part of the operating system course, where it will become so, do remember there is a VIF flag and VIP flag which basically are used for certain operating system related functionalities. And the last one is the ID flag which is the Identification Flag the ability of a program to set or clear this flag indicates, support for the CPUID instructions. So, every CPU as produced as a unique ID and you can get that ID.

(Refer Slide Time: 17:17)

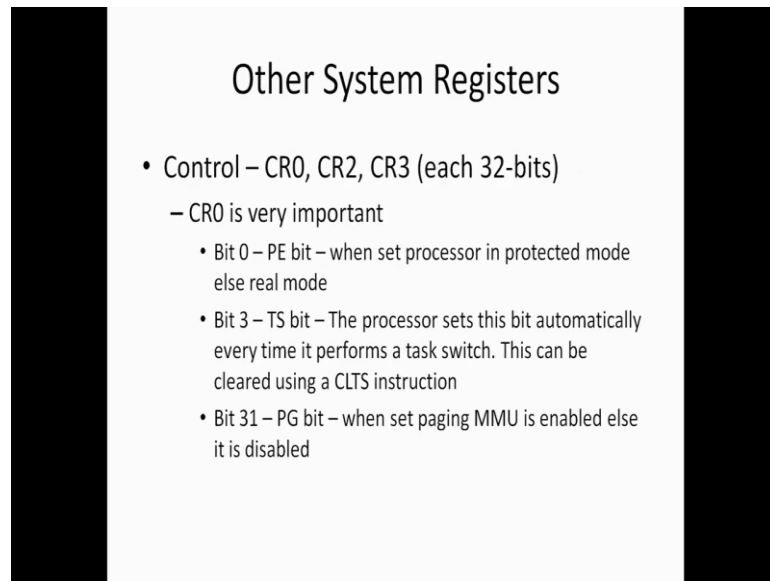
A slide titled "Protected Mode Registers" with a list of four registers and their bit widths. The slide has a white background with black text and is flanked by two vertical black bars on the left and right sides.

### Protected Mode Registers

- LDTR – Local Descriptor Table Register – 16 bits
- GDTR – Global Descriptor Table Register – 48 bits
- IDTR – Interrupt Descriptor Table Register – 48 bits
- TR – Task register – 16 bits

Then there are certain protected mode registers we will these things we will become clear tomorrow, but do understand that there is a Local Descriptor Table Register, a Global Descriptor Table Register, an Interrupt Descriptor Table Register and a Task Register of 16, 48, 48, 16 bits respectively. Just have just note it down and tomorrow when we deal with segmentation in greater detail, we will understand where these are used.

(Refer Slide Time: 17:48)

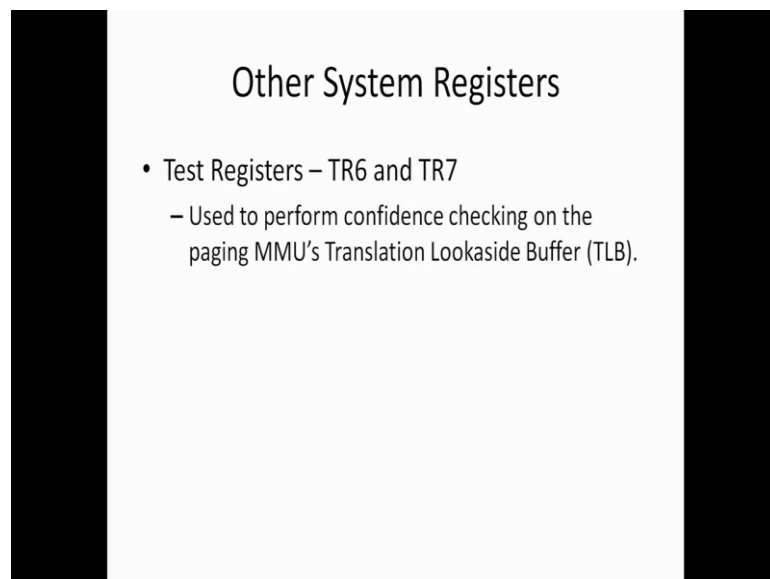


### Other System Registers

- Control – CR0, CR2, CR3 (each 32-bits)
  - CR0 is very important
    - Bit 0 – PE bit – when set processor in protected mode else real mode
    - Bit 3 – TS bit – The processor sets this bit automatically every time it performs a task switch. This can be cleared using a CLTS instruction
    - Bit 31 – PG bit – when set paging MMU is enabled else it is disabled

So, we have talked already about Control Registers and Debug Registers. And then there are test registers, which are used to perform confidence checking on the paging memory management unit translation which look aside buffer.

(Refer Slide Time: 18:03)



### Other System Registers

- Test Registers – TR6 and TR7
  - Used to perform confidence checking on the paging MMU's Translation Lookaside Buffer (TLB).

What is translation look aside buffer? What do you mean to do confidence checking on that? Those things we will see as we go through the virtual memory concepts, but there are some registers called test registers. So, what I have done today is to tell you the infrastructure that is available for computing the local storage infrastructure that is available for computing. Namely, the registers and some of the registers which are used for controlling the program, namely the flag registers.

So, with this we come to the end of session-9.