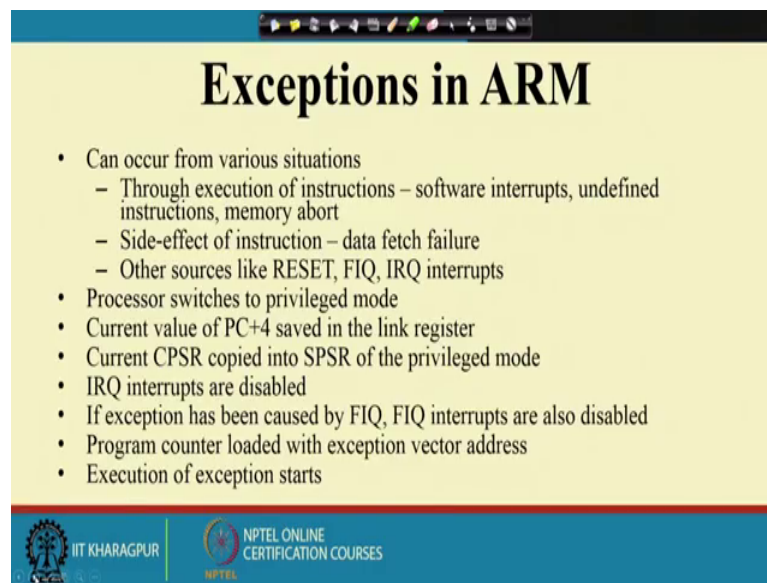


**Microprocessors and Microcontrollers**  
**Prof. Santanu Chattopadhyay**  
**Department of E & EC Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 48**  
**ARM (Contd.)**



So, next, we will be talking about exceptions in ARM.

(Refer Slide Time: 00:19)



**Exceptions in ARM**

- Can occur from various situations
  - Through execution of instructions – software interrupts, undefined instructions, memory abort
  - Side-effect of instruction – data fetch failure
  - Other sources like RESET, FIQ, IRQ interrupts
- Processor switches to privileged mode
- Current value of PC+4 saved in the link register
- Current CPSR copied into SPSR of the privileged mode
- IRQ interrupts are disabled
- If exception has been caused by FIQ, FIQ interrupts are also disabled
- Program counter loaded with exception vector address
- Execution of exception starts

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

Now, in as far as exceptions are concerned so, we can have the situations which are not occurring in the normal execution of programs. So, when the programs are executed like computations and all that they are not occurring, but these are some um extraordinary conditions, ok. So, in it is a while discussing about 8085 and 8051 we are talked about interrupts. So, this is basically same as those exceptions that we call here. So, they are all exceptional situations.

So, they can occur from various different situations like one is that through execution of instructions; so, some instruction is executed and as a result exception has occurred, like the software interrupts. So, we have got this SWI instruction or for Intel familiar processors we have got 8085 we have got RST instruction 8086 there is another instruction called INT.

So, those instructions so, they are called software interrupts and when they occur some special way service has to be invoked. So, the processor needs to switch from user mode

of op operation to some privileged mode so that this operating system for routines will be running in that mode.

So, this is the, this is a software interrupt. So, this is one source another source is undefined instructions; like when the processor is fetching instructions and executing them. So, if it happens like that that opcode part the opcode part of the instruction that the processor has got does not match with any of the known opcodes of the processor or the coprocessor, ok. So, coprocessor means that many a times we have got some additional processor that helps in executing programs like we can have some additional processor which will be doing this say the floating point operation. So, whenever that floating point operations are found so the task is given to the coprocessor. So, we will see those later.

So, this und[efined]- this undefined instruction means they are in the instruction opcode that it has got is neither the opcode of the processor nor the opcode of any of the coprocessors that it may have. So, that is one condition another condition is memory abort, like memory abort means it is trying to get the content from memory, but say the memory location is not available.

So, this happens that in system though I have got say 32 bit address bus so, I may not have that much of memory available in the system. So, as a result when it goes beyond that limits so, it is there will be memory fault.

There can be side effect of instruction like data fetch failure, like if we are trying to fetch some data so when one of the operand is the memory operand we are trying to get the data from the memory location and that memory location is not within the address space of the processor or the particular memory address is absent the memory chip is not there.

So, this type of situations there is a data fetch failure other sources which are actually the interrupt sources like the reset button that we have the reset pin that we have in the ARM processor or the FIQ pin IRQ pin so, if we are activating those pins there we interrupts and as a result so, these are the other sources for this exception.

So, for all these exceptions so that it happens like this is a processor switches to privileged mode. So, from user mode so, it will go to the privileged mode either it will go to system mode or it will go to say one of this FIQ mode IRQ mode like that. So, here

there are six different mode that we have seen previously. So, it will go to one of the privileged mode. Current value of PC plus 4 will be saved in the link register so, current value of PC plus 4 that is that in the next instruction. Every instruction is 4 byte long so, the plus so, it will be this next instruction address will be saved in the link register.



Then the CPSR register value will be copied into SPSR of that privileged mode. So, there are several SPSR registers one for each mode of operation. So, this CPSR value will be copied on to SPSR register. IRQ interrupts will be disabled, ok. So, and if the exception is caused by FIQ then the FIQ interrupts are also disabled. So, you need to enable the interrupts separately in the interrupt service routine to start to able to make those interrupts again usable.

And, the program counter will be loaded with some exception vector address. So, in case of ARM all interrupts are vectored interrupts. So, their addresses are fixed they are defined by the hardware the target address where it will the ISR address is fixed. So, the processor with the program counter will get that value and the exception execution will start. So, this is the sequence of operations that occur when we are having exceptions in the ARM processor.

(Refer Slide Time: 05:17)

### ARM7 Vector Table

<u>Exception type</u>	<u>Mode</u>	<u>Vector address</u>
Reset	Supervisor	0x00000000
Undefined instruction	Undefined	0x00000004
Software interrupt (SWI)	Supervisor	0x00000008
Prefetch abort (instruction fetch abort)	Abort	0x0000000C
Data abort (data access memory abort)	Abort	0x00000010
IRQ (interrupt)	IRQ	0x00000018
FIQ (fast interrupt)	FIQ	0x0000001C


IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

So, next we see that the vector table so, as I said that all the all these exceptions they are vectored interrupt they are they are vectored. So, this reset the mode it switches is to supervisor mode and the program counter value becomes all 0. So, that is it comes to the

very fast location of the memory. So, from that location we can have the OS code for loading the essential portions of the operating system from secondary storage to the main memory. So, that is a reset mode.

Then there is one undefined instruction mode. So, undefined instruction exception if that occurs then it branches to the address 044, ok. So, it goes to the address 4. So, you will see that between 2 such modes so, we have got only 4 bytes. So, you can have only one instruction in the in the vector location and unlike other processors like 8051 or 8085 so, where we have got at least 4 to 8 bytes available so that the small interrupt service routines can be held there itself, but in general so, that is found to be non working solutions. So, most of the interrupt service routines that we have are more than 8 bytes so, there is no point putting those only 4 putting only 8 bytes or so, in the between to interrupt addresses vector addresses.

So, in case of ARM so, that has been that has been called discontinued and then what happens is that you have in this 4 byte so, you can put a 4 words so, you can put a sorry 4 bytes or one word. So, you can put a jump instruction only. So, that jump instruction will take you to the actual inter service routine. Then after undefined instruction we have got the software interrupts SWI and if the mode is supervisor mode and the vector address is 8. So, all the software interrupts.

So, you remember there are there are 2 power 24 software interrupts that can occur in ARM processor and for all of them the control will be coming to this particular address 8. So, from here we should go to the SWI service and there I should analyze the value of n to figure out what which interrupts has really occurred so that this whole instruction is available in the instruction register so, the from there it can see we can figure out like which instruction which service has been asked for.

Then there can be prefetch abort. So, the during instruction fetch if there is an abort so, the operation mode is abort mode vector address is C and in case of data abort that is it is trying to access the data and it is not getting. So, data abort. So, in that case the address is 1 0 and then IRQ. IRQ interrupt has got this address 1 8 and FIQ or first interrupt it has got the address 1 C. So, this is the distribution and we will see that this distribution it also helps the op operation of this processor in some sense.

(Refer Slide Time: 08:30)

The slide is titled "Issues Related to Exceptions" and contains the following text:

- Vector at location 0x0000014 is missing to ensure a backward compatibility
- FIQ has been given the highest address
  - Interrupt service routine can start from this address directly
  - Eliminates the necessity of another jump instruction from the vector address to the ISR, thus saving time needed to start the routine after the fast interrupt has occurred

Handwritten notes in red ink are present:

- ① Separate Reg.
- ② JMP not needed

The slide also features logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a presenter in the bottom right corner.

So, first of all this vector location 14 is missing. So, if you look into this after one 0 the next address should have been 1 4, but that is not there. So, that is 1 4 is missing and it is for a backward compatibility. So, with some previous version of ARM, so, if you want to for keeping the compatibility, 14 is missing.

Another interesting thing to note is that you see we have got this FIQ interrupt which is at location 1C and after that there is no other entry for the vector table. So, what happens is in you can you can start this interrupt service routine from these address directly. So, from once you onwards so, you can start writing the FIQ service routine. So, that will not overwrite any other service interrupt services.

Whereas, if you for other cases for other interrupts so for say this software interrupt or undefined instruction like that so, those ISRS so, if they start at this address they will overwrite these locations of this vector table. So, as a result other interrupts will not work properly other exceptions will not work properly, but for FIQ this is the last entry and there is nothing after this. So, we can start this interrupt service routine from that address onwards.

So, this saves time because otherwise other instructions so, other inter exceptions what you have to do is put a jump instruction. So, every ISR code in though for those exceptions so, they have got an overhead of one extra jump instruction. So, that way it will take some more time for activation compared to the FIQ interrupts. So, we can see

that the FIQ interrupts are faster in ARM processor because of two reasons; one reason is that we can have we can have this thing your interrupt service this ISR at the exception address is the last one so, 0 0 that 1 C. So, the this FIQ interrupt is faster because of two reasons one reason is that we have got we have seen that they have got a separate set of registers the separate register set.

So, that is one reason and another reason that we have now is due to this highest address. So, that jump instruction jump not needed at the beginning of the ISR. So, because of that the FIQ is faster than other interrupts.

So, this helps in this helps in the operation like. So, we can have it is that critical interrupts put into this you can have critical interrupts put into this ARM into this FIQ so that we can get that service first.

(Refer Slide Time: 11:24)

The slide displays the following ARM assembly code:

```

EOR R1, R1, R1      ;clear R1 to store the largest
CMP R2, #0
BEQ Over           ;if block is empty, done

Loop
LDR R3, [R0]      ;get the data
CMP R3, R1       ;do comparison
BCC Looptest     ;skip if R1 is bigger
MOV R1, R3       ;else get the larger in R1

Looptest
ADD R0, R0, #4    ;increment pointer R0
SUBS R2, R2, #1   ;decrement number of elements left
BNE Loop

Over
;R1 holds the largest

```

Handwritten annotations on the slide include:

- A red arrow pointing from the word "Loop" to the `LDR R3, [R0]` instruction.
- A red arrow pointing from the word "Looptest" to the `BCC Looptest` instruction.
- A red arrow pointing from the word "Over" to the `BEQ Over` instruction.
- A red box on the right side representing an array, with a red arrow pointing to its top edge labeled "R0" and a red arrow pointing to its right edge labeled "R2=Count".

The slide footer includes the IIT KHARAGPUR logo and the NPTEL ONLINE CERTIFICATION COURSES logo.

Next we look into one program this is an example program for finding the maximum of a set of numbers. So, that it is so, we first the R1. So, it will R1 register we first clear it. So, that it will hold the largest number in it. So, you xor R1 with R1 and the result is also R1. So, it will be ze it will be there. So, R2 register. So, I assume that the numbers are stored in an array and R0 register points to the beginning of that block, ok.

So, it is assumed that the R0 register is holding the if this is the memory where this values are there then the R0 register is pointing to the beginning and this block is ending

with a this R2 register. So, this R and R2 register is holding the count, like how many numbers are there in that array, ok. Then, we first we compare R2 with 0. So, if R2 is 0; that means nothing to be done. So, it will be over so, branch on equal to over.

Otherwise, it comes to this statement and in this statement it is LDR R3 within bracket R0. So, the first number that you have is loaded into the R3 register then we compare R3 with R1, ok. So, R1 was R1 was R1 is holding the current maximum. So, we compare R3 with R1. So, it is assumed that all the numbers are positive if there are negative numbers then of course, this algorithm may not work. So, it is you compare R3 with R1 and if it is carry is there that means, if R1 is bigger then there will be a carry. So, we will be coming to this loop test. So, otherwise this new number R1 is new number R3 is bigger so, we move R3 to R1, ok.

And, then after that so, now, I should go to the next number and since these individual numbers are 32 bit number. So, it is assumed to this R0 should be implemented by 4 so, that we come to the next number. So, increment point are R0 then we reduce this R2 by 1. So, we reduce R2 by 1 and if it is not same so, branch on not equal to loop. So, if there if this R we subtract R2 one from R2 and store the number in R2 and then if it is not equal that is a 0 flag is set then this equality flag will be there. So, if branch are not equal to loop so, we will come back to this point again load the next number and compare.

So, there can be different ways by which this program can be written and the objective is also not to say that ok, we have to we have done a very good survey of this all the instructions ARM, but we are trying to look into the features of ARM and the types of instructions that are there, ok. So, this program is just an example on how to use those instructions for doing some operation. So, next we will look into another program look into another program.

(Refer Slide Time: 14:47)

The slide is titled "Comparing two null terminated strings". It contains the following assembly code:

```
Loop
LDRB R3, [R0] ;get next character of string 1
LDRB R4, [R1] ;get next character of string 2
CMP R3, R4 ;compare
BNE Notsame ;if not same, strings do not match
CMP R3, #0 ;check if end of string reached
BEQ Same ;if equal, same
ADD R0, R0, #1 ;increment pointer to string 1
ADD R1, R1, #1 ;increment pointer to string 2
BAL Loop ;branch always to check next character

Notsame
MOV R2, #-1 ;mark not matched
BAL Over

Same
MOV R2, #0 ;mark matched

Over
;R2 holds the match
```

Handwritten annotations on the right side of the slide show two vertical strings of characters: 'A', 'B', 'C', '0' and 'X', 'Y', 'Z', 'W', '0'. To the right of these is a vertical stack of five memory cells. The first cell is labeled 'R0' and the second cell is labeled 'R1', with arrows pointing to them from the right.

The slide footer includes the IIT Kharagpur logo and the text "NPTEL ONLINE CERTIFICATION COURSES". A small video inset of a presenter is visible in the bottom right corner.

So, you want to compare two null terminated strings. So, null terminated string means I have got so, if the first string is say A, B, C the first string is A, B, C. So, they are stored in successive locations A, B, C followed by a 0 this character is A character B and character C and then a null character and the other string may be some X, Y, Z, W like that and then 0.

So, this is the other string now while we have this type of thing this is 0. So, we have to compare so, LDRB R3 R0 it is assume that R0 points to the beginning of first string. So, if this is my memory and maybe in this region I have got the first string so, this is pointed 2 by R0 and in this region we have got the second string which is pointed to by R1.

So, we get a so, LDRB R3 within bracket R0. So, it will get the next character from string one and LDRB R4 within bracket R1. So, it will get the next character of string 2 into R 4 we compare R3 with R4. So, if they are not equal then naturally they are not same we just check whether the strings are same or not. So, if they are not same then we come to this not same location and set this we said this R2 to minus 1 we said this R2 to minus 1 to say that they are not matching.

If it is so, if they are same if the first two characters the both of them happens to be say a then we compare with the next character. So, for that purpose R0 is implemented by one R1 is also implemented by one they are now pointing to the next two characters of the



strings and then we again compare within branch 2 plus BAL is branch always. So, it comes to this loop and it will again B moving the next character into R3 and next character of second string on to R4 compare them.

So, if any of those characters are not same then this BNE not same this will be true. So, it will come here and R2 will be getting minus 1 and if R2 is if any point of time we find that the strings are not same we do not compare the remaining characters. So, we just go to this just skip over the same part and we come to branch always over. So, if R2 is getting minus ones after that we are coming to this point, the program ends or if it is always same then we move this move R2 comma as 0. So, the R2 register gets the 0 value. So, if the strings are same then with R2 will get a 0, if strings are not same then R2 will get a minus 1.

So, this is a comparison of these a this program will just compare between two strings whether they are same or not. So, you can extend this program for to check whether the strings are whether the strings one string is greater than the other and all that. So, those types of checks can be incorporated.

(Refer Slide Time: 18:09)

**ARM Cortex Processors (v7)**

- ARM Cortex-A family (v7-A):
  - Applications processors for full OS and 3<sup>rd</sup> party applications
- ARM Cortex-R family (v7-R):
  - Embedded processors for real-time signal processing, control applications
- ARM Cortex-M family (v7-M):
  - Microcontroller-oriented processors for MCU and SoC applications

The diagram on the right shows a hierarchy of processors. At the top is Cortex-A15 (x1-4, 2.5GHz). Below it are Cortex-A9 (x1-4) and Cortex-A8 (x1-4). Cortex-A9 is connected to Cortex-A5 (x1-4), which is connected to Heron (x1-2). Below Heron are Cortex-R4 and Cortex-M4. Cortex-M4 is connected to SC300™. Below SC300™ are Cortex-M3 and Cortex-M1. Cortex-M1 is connected to Cortex-M0 (12k).

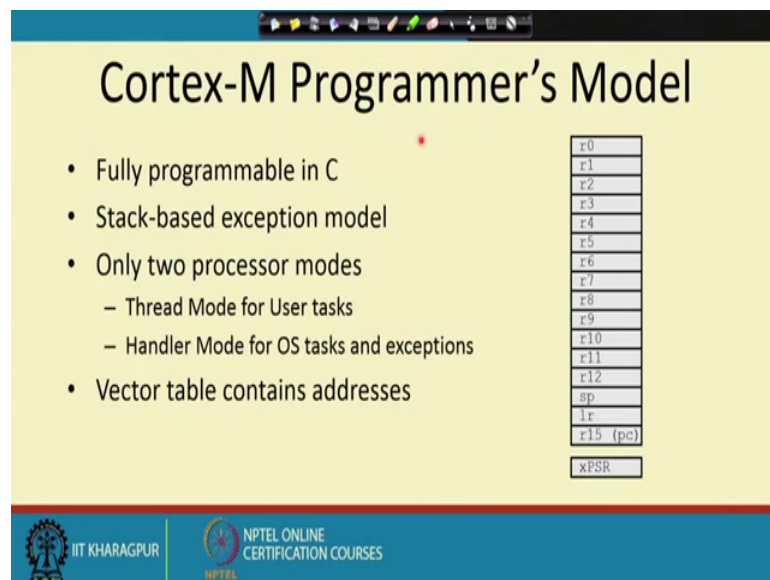
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, next we will next we look into the advanced processors of ARM. Like as I said that now ARM processors; so, they have been developed into the cortex family of processors and this ARM cortex. So, it is divided into three different families, ok. So, one is the A family cortex-A family, so, application. So, this is these are for the server or you can say

it is for the application processors so, for supporting their new operating systems and all that. So, this is for basically computational jobs if you are there lots of computations a server type of application. So, there this cortex-A series of processors are used.

Then for embedded processors and real time application, signal processing and control application we have got R cortex-R series of processors and for microcontroller oriented applications and processors. So, we have got this cortex-M series. So, this microcontroller and system on chip type of application. So, this cortex-M series of processors are advocated. So, for our discussion so, cortex-M processors so, they are more they are close to our discussion.

(Refer Slide Time: 19:19)



The slide titled "Cortex-M Programmer's Model" lists the following features:

- Fully programmable in C
- Stack-based exception model
- Only two processor modes
  - Thread Mode for User tasks
  - Handler Mode for OS tasks and exceptions
- Vector table contains addresses

On the right side of the slide, a vertical list of registers is shown in a table-like format:

r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
r12
sp
lr
r15 (pc)
xPSR

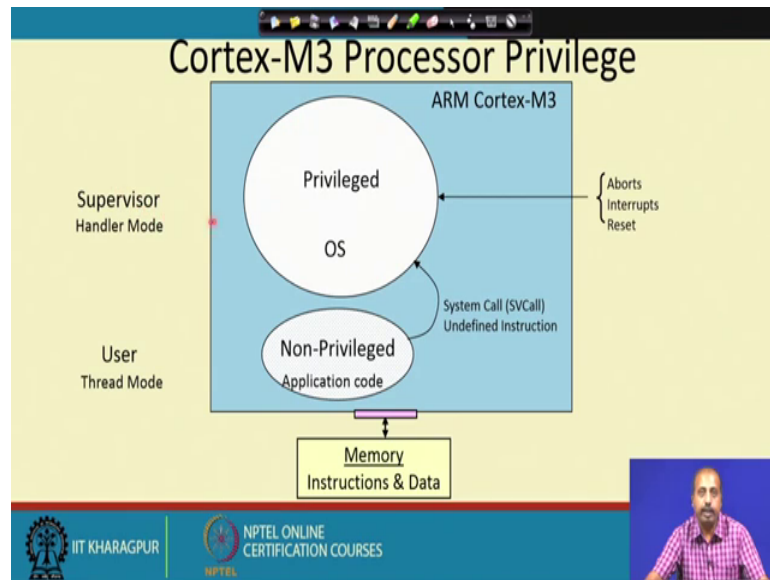
The slide footer includes the logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES.

So, we will look into those cortex-M processor. So, cortex-M processor so, the register set is same that R0 to R15 as we have then the PSR program set register CPSR, SPSR is there the good the since it is difficult to learn the complete assembly level programming of cortex-M. So, what they have done. So, they have developed very good compiler C compilers so, you can write your specification in the c language and then through this compiler. So, you can generate the code for this cortex the processor cortex processor.

Then stack based execute exception model. So, whenever some exception occurs then these values are saved into stack. So, the automatically this register values will be saved and there are only two processor modes; one is the user mode and another is a thread mode that thread mode or user mode. So, higher it will be user task will be executed and

the handler mode for OS tasks and exception. So, instead of that six mode so, we have got only two modes and vector table will contain the addresses for those exceptions.

(Refer Slide Time: 20:30)



So, in cortex-M3, we have got this non-privileged mode or the user mode and privileged mode or the OS mode. Then this so, in the non-privileged mode this application code will be running and this so and in the privileged mode we will have the system call. So, when application code is running sometimes we make a system call. So, it will go to the privileged mode or if it is the undefined instruction so, it will go to the privileged mode.

So, all the aborts interrupts and resets so, they are all handled in the privileged mode of operation, ok. So, this is the supervisor or handler mode and this is the user or thread mode.

(Refer Slide Time: 21:13)

**Cortex-M3 Interrupt Handling**

- One Non-Maskable Interrupt (INTNMI) supported
- 1-240 prioritizable interrupts supported
  - Interrupts can be masked
  - Implementation option selects number of interrupts supported
- Nested Vectored Interrupt Controller (NVIC) is tightly coupled with processor core
- Interrupt inputs are active HIGH

The diagram shows the INTNMI signal and 1-240 Interrupts (INTISR[239:0]) connected to the NVIC, which is tightly coupled with the Cortex-M3 Processor Core.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

For interrupt handling so, this has it has incorporated one special thing like we have got one non-maskable interrupt. So, that. So, this is this NMI INTNMI. So, this is one non-maskable interrupt that is supported and then we have got 240 prioritizable interrupts. So, 1 to 240 prioritizable interrupts are supported and these interrupts can be masked ok.

So, unlike say your 8085 or say 8051 where we have got a limited number of interrupts so, here the number of interrupts are really large and this is useful like, if you are having some embedded application then there may be different devices connected to the system and also so, you may need to have all these interrupts.

So, so, there are 240 such prioritizable interrupt. So, we can have this interrupts can be masked also and implementation option selects the number of interrupts supported. So, the original ARM code will tell that it will support up to 240 maskable interrupts, but you can tell that in my system I do not want. So, many I want only say 20 of them.

So, naturally this NVIC module that is not nested vectored interrupt controller NVIC module so, this is a part of this cortex-M3 processor now. So, you can simplify the design of this NVIC, if the number of interrupts are less than the NVIC design it also be will also be simpler and interrupt inputs are active high all are active high interrupts going to the NVIC. So, this interrupt controller has been added in the M3 processor.

(Refer Slide Time: 22:55)

**Cortex-M3 Exception Handling**

- **Reset** : power-on or system reset
- **NMI** : cannot be stopped or preempted by any exception other than reset
- **Faults**
  - **Hard Fault** : default Fault or any fault unable to activate
  - **Memory Manage** : MPU violations
  - **Bus Fault** : prefetch and memory access violations
  - **Usage Fault** : undef instructions, divide by zero, etc.
- **SVCcall** : privileged OS requests
- **Debug Monitor** : debug monitor program
- **PendSV** : pending SVCalls
- **SysTick Interrupt** : internal sys timer, i.e., used by RTOS to periodically check resources or peripherals
- **External Interrupt** : i.e., external peripherals

The slide includes logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES. A small video inset shows a presenter in the bottom right corner.

As far as the exceptions are concerned so, reset is there. So, that is power on or system reset then the NMI non-maskable interrupt. So, this is the cannot be stopped or preempted by exception any exception other than reset. So, that INT, NMI line is there by on which you can connect some non-maskable interrupt. Then the faults we can have hard fault, we can have memory manage, we can have bus fault or usage fault.

So, this is the, these are the different types of faults that are that are there that have that is recognized by the processor. So, default fault or any fault unable to be a to activate. So, so we can we cannot operate I can handle that fault so, they are called hard faults.

Then memory management so, memory management processor they same some violation the memory access there is some violation bus fault is prefetch and memory access violation and usage for undefined instruction divided by 0. So, while executing the instruction some exceptional condition occurs. So, divide by 0 some the divisor becomes 0. So, as a result the division is undefined.

Then there is supervisory call SV call. So, these are the OS requests. So, we have got that SWI instruction. So, they are mapped onto SV calls here. Then there is a debug monitor; so, that will be handling this you can monitor the operation of the chip. So, and through a debugger you can just check the values of those then the there is a pending supervisory calls like pendSV. So, this will this will give you the pending supervisory call. Then there

is a SysTick interrupt, so, SysTick interrupt is for internal system timer. So, this is useful like if you are willing to implement some software some timer in your program.

So, you do not need to do anything else you just use the SysTick interrupt from the M3 processor directly and there are external interrupts all those 240 interrupts that we have, so, all of them can be utilized.

(Refer Slide Time: 25:04)

The slide is titled "Conditional Execution" and contains the following content:

- If – Then (IT) instruction added (16 bit)
  - Up to 3 additional "then" or "else" conditions maybe specified (T or E)
  - Makes up to 4 following instructions conditional

Diagram illustrating the IT instruction format:

<del>ITTT</del> <del>EQ</del>
<del>Inst 1</del>
<del>Inst 2</del>
<del>Inst 3</del>
<del>Inst 4</del>

Diagram illustrating the conditional instructions:

<del>MOVE</del>
<del>ADDE</del>
<del>SUBNE</del>
<del>ORRE</del>

- Any normal ARM condition code can be used
- 16-bit instructions in block do not affect condition code flags
  - Apart from comparison instruction
  - 32 bit instructions may affect flags (normal rules apply)
- Current "if-then status" stored in CPSR
  - Conditional block maybe safely interrupted and returned to
  - Must NOT branch into or out of 'if-then' block

The slide footer includes logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, along with a small video inset of a presenter.

So, another very interesting feature that this ARM processor cortex-M3 has is the conditional execution. So, it is the if then else type of instructions. So, if then instruction added are it has 16 bit instruction up to 3 additional then or else conditions can be specified, likes a let us look into this statement. So, this so, this statement it says that so, INT INTT. So, this is if then.

So, you read it like this if then else then EQ; that means, the first instruction one will be executed if the condition is true, so, if so, then. So, this then corresponds to the first instruction, then the next then also corresponds to the second instruction. So, second instruction will also be executed if that condition equality was true at this point. This instruction 3 will be executed if the condition was false that is the else part and the last instruction will be executed again if the condition was true.

So, in this way you can have up to 4, if then else, instruction. So, basically if so, if the basic insta instruction is like move, add, sub and or the or then the these EQ since if the

first second and fourth instruction they were then part true part. So, for them we have got the qualifier EQ and for the third instruction. So, this was else E part. So, it is not of EQ so, that is NE. So, instead of so, this is similar to this block. So, we do not need to talk about in a single statement or we can tell all these flags and then for the remaining instruction. So, I need not tell it explicitly whether it is EQ or the flag conditions the conditional execution flags need not be explicitly told separately. So, this helps in writing programs where with the if then else type of constants.

So, any normal ARM condition code can be used and 16 bit instructions in block do not affect condition code flags. So, they will not affect the condition code apart from the comparison instruction. 32 bit instructions may affect condition flags; so, that is the ARM instructions they will affect the condition flag. The current if then status is stored in the CPSR register. So, conditional block may be safely interrupted and returned to and must not branch into or out of the if then block.

So, that is so, you cannot take a branch instruction from here before completing this if then else. So, you cannot have the third instruction as a branch out instruction or you cannot just branch into a block. So, you cannot say that I will come to the mid jump on to the middle of this block, so, that is also not possible. So, those are there.

So, this way these M3 processor so, it has got many interesting features and they are many more features are there, so I would suggest that you look into this ARM manual for their details, but whatever we do. So, that is actually the ARM processor is one of the very powerful processors and it has got the advantage because we have got low power consumption and it is quite powerful in the sense that it is comparable with many of the high speed processors; that we have now with the limited amount of hardware requirement because you can take only the essential parts and others can other may not be taken as part of the processor.