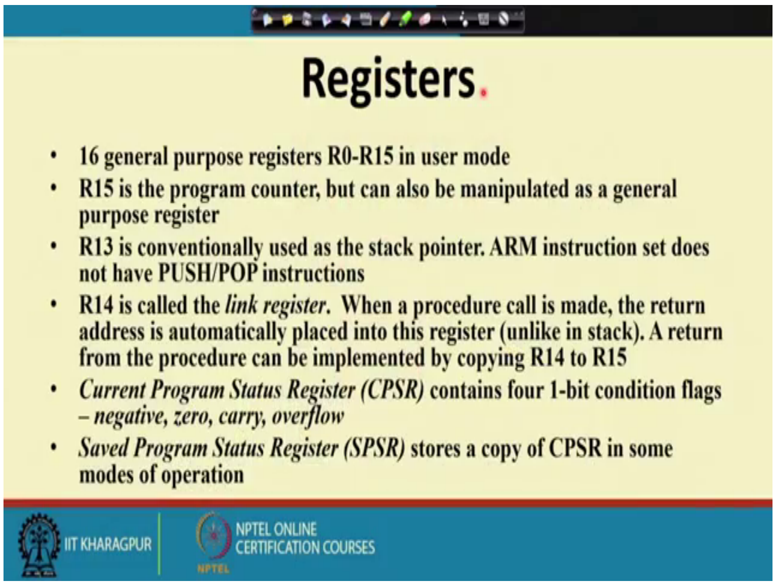
Microprocessors and Microcontrollers
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture – 44
ARM (Contd.)

In the registers, this another interesting feature that this ARM processor has. So, there are 16 general purpose registers, mark R-0 to R-15 in user mode ok.

(Refer Slide Time: 00:20)



Registers.

- 16 general purpose registers R0-R15 in user mode
- R15 is the program counter, but can also be manipulated as a general purpose register
- R13 is conventionally used as the stack pointer. ARM instruction set does not have PUSH/POP instructions
- R14 is called the *link register*. When a procedure call is made, the return address is automatically placed into this register (unlike in stack). A return from the procedure can be implemented by copying R14 to R15
- *Current Program Status Register (CPSR)* contains four 1-bit condition flags – *negative, zero, carry, overflow*
- *Saved Program Status Register (SPSR)* stores a copy of CPSR in some modes of operation

So, user mode is basically the mode in which the processor is generally in when it is executing some program. So, it is, it has got R-0 to R-15. So, as an user, so I can use this R-0 to R-15 registers.

Out of this 16 registers R-15 is the program counter ok. So, unlike other processors that we are familiar with where the program counter was a special register and there are only a few instructions by which you can access the program counter. For example, in case of 8085 you remember there is an there was an instruction PCHL by which this program counter was accessible or you can make some call to a subroutine by which a program counter value will be available in the stack and do the manipulation there.

So, that way we can do it, but the you can just in case ARM processor it is not there. So, R-15 is available to the user for as a general purpose register as well.

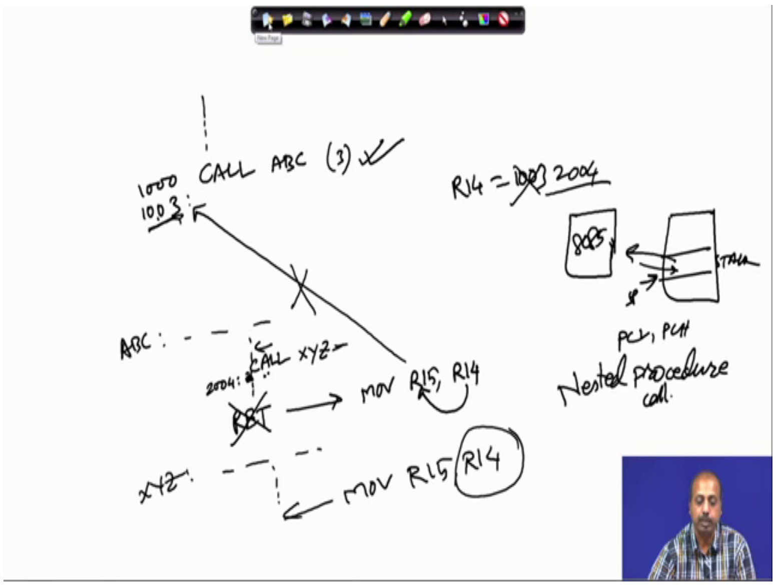
Then the R-13 register is conventionally used as stack pointer. So, please note the term conventionally. So, means that there is no hard and first rule that I have to use this, I have to use this thing R R-13 as the stack pointer. So, I can use that R-13 as simple register also as general purpose register also, but in general it is used as stack pointer. And this is another thing to note that the ARM instruction set it does not have PUSH, POP instructions. So, there is no push pop type, PUSH, POP instructions in ARM. So, you do not have this stacks available in ARM.

So, it seems a very surprising because the program that I am writing. So, there may be sub programs and of that. So, the stack is not there. So, how am I going to save the written addresses of the subroutines and how am I going to pass parameters like that, so there are many concerns. So, what this ARM people thought possibly is that it depends on the program. So, some many programs may not need to call a subroutine.

So, it may be that it is just written in a flat fashion. So, the entire code is a single routine ok. So, there is no sub program. So, as a result I do not need any stack in that, so why should I have that burden of stack. So, if I need stacks. So, I can use it I can define in it some in my own way that we will see later, but by default this ARM does not have any stack pointer ok.

Now, this ARM, so if you are using a stack if you are defining a stack in that case this R-13 is used as the stack pointer and, but that is a conventionally. So, if this is not there if the stack is not there then the biggest disadvantage that we have is the calling and returning from sub program. So, what this ARM has done is that they have dedicated this register R-14 called the link register. So, whenever a procedure call is made the return address is automatically placed into this register and a return from procedure is simply by copying R-14 to R-15.

(Refer Slide Time: 03:45)



So, let us take an example say I have got a program and somewhere I am calling subroutine call say subroutine ABC and this subroutine ABC is somewhere here ok. So, here it is doing at the at the end I have got the return from subroutine.

Now, what this; if this is say the address is this 1000 and this instruction is say 3 byte instruction, so next location is 1003. What the ARM people will do the ARM processor will do is that in the link register ok, so in the link register that is in R-14 it will put this value of 1003, in R-14 it will put the value 1003, while this call is being executed it will be done like this. Now when it comes to this point return exclusive return is not necessary. So, what you can do R-15 is the program counter.

So, you can simply say MOV R-15 comma R-14. So, what will happen? This R-14 will be copied into R-15 and as a result the program counter is now modified, so you are back to this point. So, the program will continue executing. So, it is like the return from the subroutine. So, this way it is fine.

So, what is the what may be a possible advantage? The possible advantage that we have is you see there is no memory access. So, if we if we consider say 8085. So, if this is the 8085 processor and you have got this as the memory a part of the memory is defined as a stack and the top of the stack is point to two by the stack pointer. So, whenever if subroutine call is made, so the PC values are saved on to this registers the PC low and PC high, they will be saved on to this stack and for returning.

So, this values will be popped out from the stack. That means, whenever it is going for a procedure call or subroutine call it is doing a memory access and returning also it is doing memory access. So, that is time consuming that is power consuming. So, external bus activities will be there address bus data bus clients they will be activities, so will have those time and power consumptions will come into picture.

But here what is happening is a say R-14 R-15 they are all internal registers ok. So, they are within the CPU. So, there is no difficulty. So, they are simply register to register data transfer so that is very fast. So, that way it is a very convenient way of doing this subroutine call when we are calling a single subroutine. But of course, there is a catch like if you I have. So, whenever you are within the routine ABC if there is another call called to another routine XYZ. Then what will happen? So, if this is a starting at say location say this is suppose this the next instruction location say two 1004, then while this call is being executed this R-14 will get overly 10 with the value 2004 now coming back.

So, coming back whenever this XYZ routine is over, suppose this is the routine XYZ and then when you are whenever you are trying to go back. So, here you have to put that instruction MOV R-15 comma R-14. So, where will it come back? So, it will come back to this location, but by that time your R-14 register content is lost, so this value that you put here this 1003 that was put there. So, by that by this time this return value is lost so you will not be able to return this one properly. So, this return will not work.

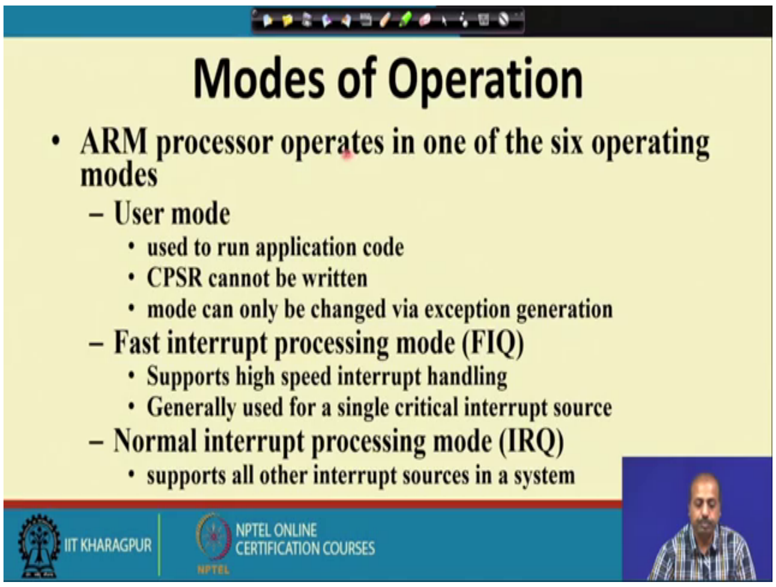
So, basically you need to do something extra to take care of this type of situation. So, whenever you have got this nested procedure call, nested procedure call we have got difficulty ok. But we in a program I may not have nested procedure calls. So, as a result this may not be necessary. If it is necessary then I have to take special care like I need to save the content of this R-14 register on to stack and while returning from here, I have to just pop out the content from after returning here I have to before calling this I have to push this as R-14 register value in to stack and after coming this point I have to pop out the R-14 register from stack.

So, somehow the stack will be necessary, but that is program specific and the program are if needed will be you implementing the stack himself ok. So, the basic processor will not support it.

Now, so that is the R-14 is the link register and it has got procedure call is made the return address is saved in this R-14 register and return will be done by copying R-14 to R-15. Then there is a program status register there is a current program status register and there is number of stored to saved program status registers there number of such SPSR registers are there. So, current program status register it has got 4 condition flags negative, zero, carry and overflow.

So, there is nothing like auxiliary carry and things like that. So, it has got only 4 carry, 4 bits negative zero carry and overflow and there is a SPSR register that can save the content of CPSR register in some modes of operation.

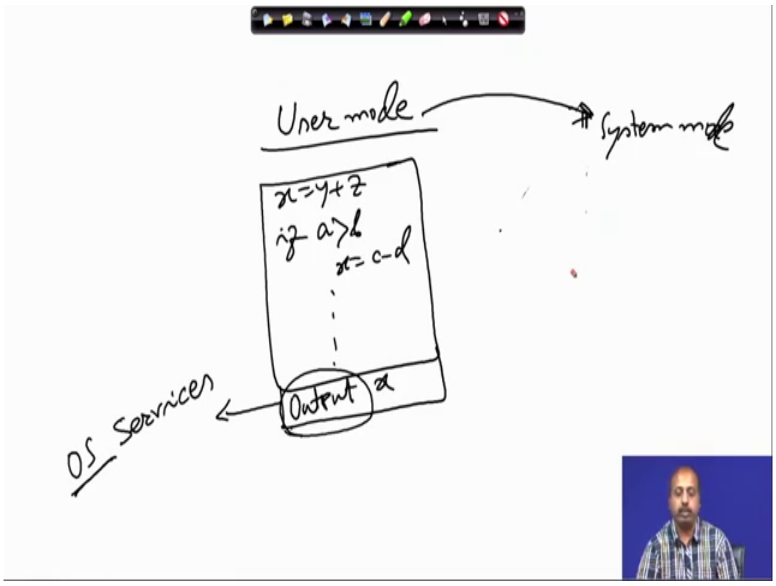
(Refer Slide Time: 09:22)


Modes of Operation

- ARM processor operates in one of the six operating modes
  - User mode
    - used to run application code
    - CPSR cannot be written
    - mode can only be changed via exception generation
  - Fast interrupt processing mode (FIQ)
    - Supports high speed interrupt handling
    - Generally used for a single critical interrupt source
  - Normal interrupt processing mode (IRQ)
    - supports all other interrupt sources in a system

So, if you look into the modes of operation then the ARM processor can execute in different modes ok. First one is the user mode when the program is executing some application code. So, in general I can say that whenever we are looking into a processor operation. So, the operations can be classified into different category. The basic mode is the user mode of operation.

(Refer Slide Time: 09:47)



Now, in user mode of operation, we have got this program statements like x equal to y plus z if a is greater than b then x equal to c minus d. So, like that we are writing program. So, as long as it involves only memory access and some arithmetic logic operations, so they are program will be executing in the user mode. But sometimes what is needed is that maybe we want to output the value of x.

Maybe we want to put it on to the screen or put it on to the printer the value of x has to be put on to the on some device. So, in that case the how is it implemented because if this access is unrestricted like you in a system I can have a number of user programs executing and if I do not have any control over this thing like which device program will access the device and all that. So, there will be a full chaos ok.

So, what is normally done is whenever a device whenever a program request this type of service. So, they are known as cyst as services from the operating system or OS services and whenever this operating systems services are necessary we go from user mode to something called user mode to system mode or kernel mode. So, this is called user mode to system mode or kernel mode and this execution is done in this mode ok.

So, that it is much more protected modes. So, it is you cannot; so in user mode you cannot access many of the this system resources, but in kernel mode or system mode you can do that. So, that way there can be two different modes which are common in most of the systems ok.

So, apart from that there are other modes as well like we can have this user mode to run application code. So, this user mode is restricted in the sense that it cannot modify the CPSR register ok. So, that will be affected only by this arithmetic logic operations, but you cannot modify it by program instructions. And we can change the mode only via some exception generation.

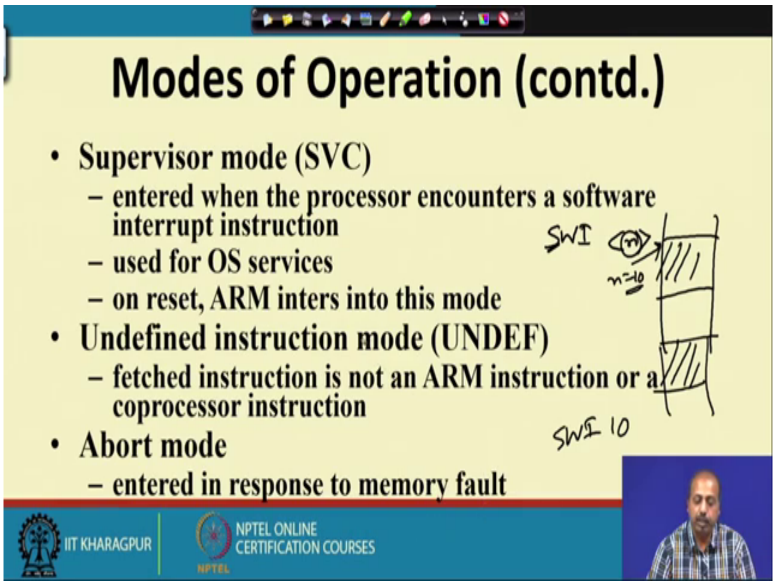
So, exception is like whenever you are asking for a device support so that is an exception to the system. So, that is given as a system call to the operating system. So, that is an exception. So, it that is, there can be different ways by which you can raise exceptions. So, you can change from user mode to other mode by raising exceptions.

Another mode of operation is the fast interrupt processing mode or FIQ mode. So, this is one inter processing mode, but it is high speed interrupt. Like whenever you are having say any embedded application then all the activities are not equally important ok, all the events that are not equally important. Like when we have got say a plant being controlled by say the ARM processor, now if they further regular operation they have some priority, but if there is a smoke detector detects a fire situation or smoke is detected. So, that is an emergency condition. So, the action has to take place immediately. So, that way, so that can be sent as a fast interrupt to the system. So, that the response is much faster.

So, generally a single critical interrupt source is connected to the FIQ pin. So, there is a dedicated FIQ pin in the processor and it is connected to that. So, this way we can have this may be the power supply line is power supply, power failure is detected on the FIQ connected to the FIQ.

So, that if there is a power failure the system will finish of the urgent works and then it will go to back up, it then it will go to shut down. So, and there is another interrupt processing mode which is known as IRQ mode where the processing. So, it is other interrupts in the system. So, it is a ARM processor it will have two categories of interrupt one is first interrupt processing by FIQ, and normal interrupt processing by IRQ.

(Refer Slide Time: 13:59)


Modes of Operation (contd.)

- Supervisor mode (SVC)
  - entered when the processor encounters a software interrupt instruction
  - used for OS services
  - on reset, ARM inters into this mode
- Undefined instruction mode (UNDEF)
  - fetched instruction is not an ARM instruction or a coprocessor instruction
- Abort mode
  - entered in response to memory fault

SWI (n)
n=10
SWI 10
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

So, other modes of operation like we have got supervisor mode, this is basically that system mode that I was talking about. So, this is entered when the process are encountered, software interrupt instruction. So, software interrupt instruction, these are basically the feature that is that are used by that are provided by the system by the processor designers.

So, these are interrupts, but these are some saw in a in a program you can put this type of instructions. Like we can have the instructions like SWI which is software interrupt and a number n is given. So, what the system does is depending upon the value of n there are some there is free specified memory location where it will branch ok. So, this may be for SWI, n equal to 10 may be it will come here ok.

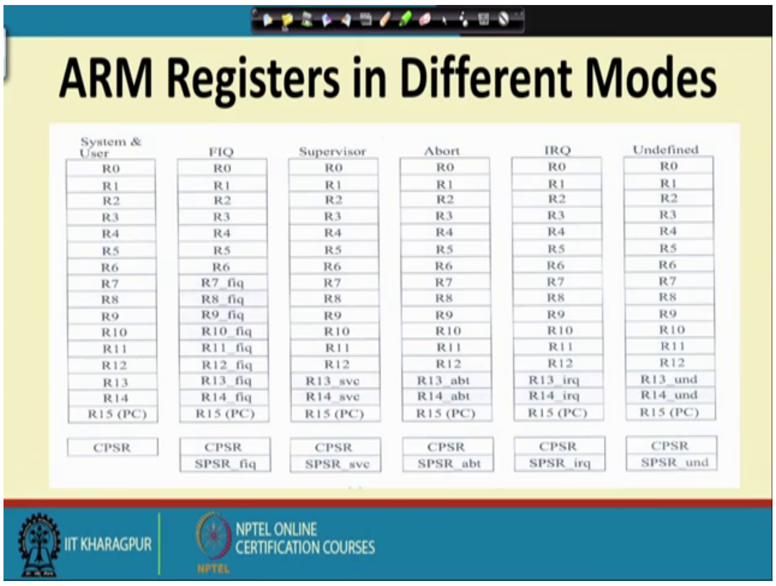
Now, what the operating system does is that for different services like accessing printer, accessing display, accessing file, accessing keyboard, etcetera it may define different different services and this corresponding service routines may be loaded in this regions, fine. So, now in a program suppose I need the printer service and the printer service I know the service number is 10.

So, in my program I call this SWI 10 ok. So, I will be accessing this routine which is the printer access. So, this is, so this SWI, when this SWI instruction is executed by a program. So, the system will go to the supervisor mode and in the supervisor mode. So, it will be it will be doing this OS service ok. So, normally this is, software interrupts are reserved for this OS services. On reset ARM will enter into this supervisor mode and then it will be going to user mode depending upon the operating system control.

There is one undefined instruction mode. So, undefined mode says that the this fetched instruction is not an ARM instruction or a coprocessor instruction. So, it is the processor is continually getting instructions from memory. So, if the program that is loaded in the memory it has got it is not loaded properly some of the some of the instructions are wrong in the sense that the they do not correspond to any of the valid opcodes in that case the op the instruction is an undefined instruction and then the processor will go to these undefined instruction mode and the accordingly it will do some operation.

Then there is an abort mode. So, this is if there is a memory fault. So, it is it has generated some address the processor has generated some address, but that address is beyond the range of the memory chip that we have in the system. So, that is a that is a memory fault ok. So, that way it will be going into the abort mode, so this all these modes are supported by the ARM processor.

(Refer Slide Time: 16:58)



ARM Registers in Different Modes

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7_fiq	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13	R13_fiq	R13_svc	R13_abt	R13_irq	R13_und
R14	R14_fiq	R14_svc	R14_abt	R14_irq	R14_und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_svc	SPSR_abt	SPSR_irq	SPSR_und

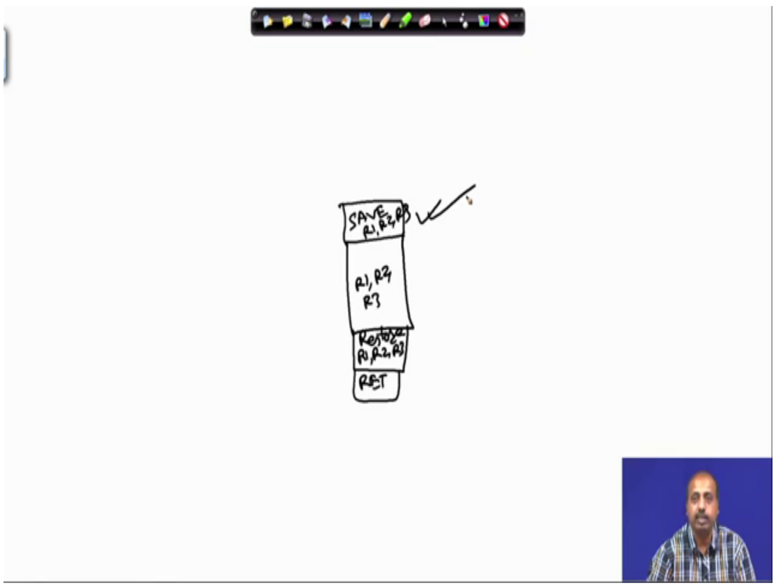
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, in different modes this ARM has got different sets of registers. So, this is the system and user mode and they share this registers R-0 through R-15. So, this registers are marked for the user mode and system mode. So, this registers are common ok.

Now, out of that I have already said R-15 is the PC R-14 is a link register R-13 stack pointer generally like that we have said. Now, if you look into the FIQ mode, in FIQ mode you have got the registers R0 through R6. So, that is fine, but this R7 to R14. So, these are new registers ok. So, this R0 to R6, this is these are the same as your user mode register R-0 through R6, but these registers are new they are not same as these R7 to R14, R15 R14 that we have here they are not same here they are new registers.

So, what is the advantage? So, it is similar the into the bank switching of say 8051. So, whenever you are writing the interrupt service routine. So, if you can restrict your inter service routine to use these registers only R7 FIQ to R14 FIQ then you do not need to save any of the registers. So, in any interrupt service routine when you are writing any interrupt service routine.

(Refer Slide Time: 18:24)



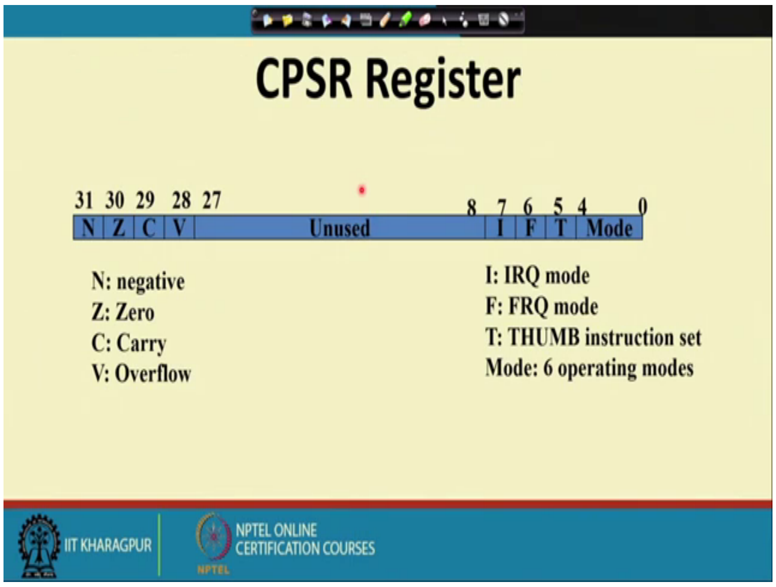
So, we know that if this is the body of the interrupt service routine then before that if this interrupt service routine say uses the registers say R1, R2 and R3 then before we start this interrupt service routine we have to save this registers save R1 R2 R3 and after this before returning. So, you have to restore these registers. So, restore R1 R2 R3 and then only I can do return. Why back because in the calling program this R1, R2, R3 might have been being used. So, in this inter service routine before using those registers we should save them and before returning we should restore them.

Now, this is the over rate you know that if it if a particularly for embedded application if I am working in a real time environment and that interrupt has occurred and now I have to save the registers before going into the actual operation. So, that way this is an over rate. So, if I if I use in the FIQ mode we have got a separate set of registers R7 FIQ to R14 FIQ. So, if you use those registers only then you know that they are different from whatever register the program the previous user program was you utilising. So, that way you can you need not do this saving part. So, you can forgo this saving and restoring part. So, your ISR becomes must simpler ok. So, this can be done.

Similarly, we have got this supervisor mode where this, only R-13 R-14 they are new registers then abort mode we have got R-13 R-14 as new registers IRQ also has got R-13 R-14 as new register same for undefined. Apart from that, so each of this CPSR registers every mode has got the CPSR register and there is a corresponding SPSR register which is saved program status register.

So, whenever you come to a particular mode, so previous CPSR register is saved automatically into the corresponding SPSR register. So, that while you are going back, so you can copy these SPSR register on to the CPSR back and you can continue with that. So, that way we have got a large number of registers in this ARM processor that helps us to write programs you using the register file that the programs will be faster.

(Refer Slide Time: 20:48)



CPSR Register

N: negative
Z: Zero
C: Carry
V: Overflow

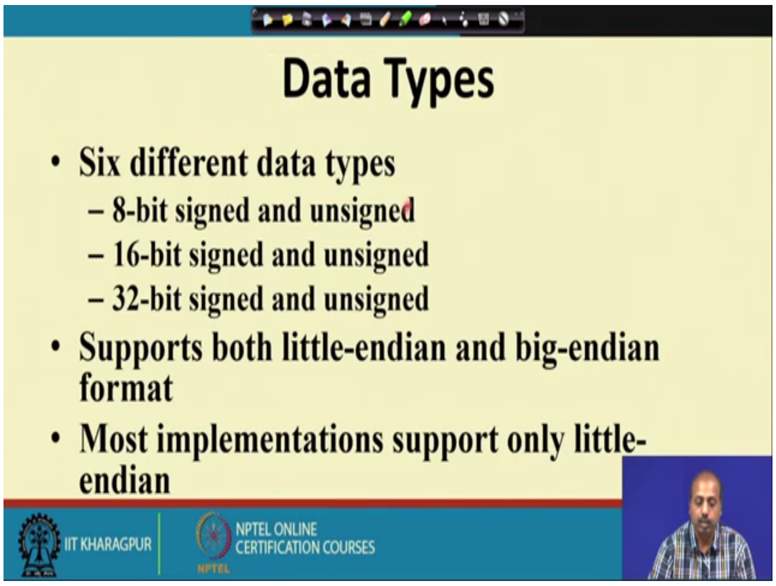
I: IRQ mode
F: FRQ mode
T: THUMB instruction set
Mode: 6 operating modes

Then the CPSR register. So, it has got the 4 important bits N Z C and V which are negative, zero, carry and overflow these are the most significant bits of this 32 bit register.

There are some more interesting bits ok. So, these bit number 0 to 4. So, they are reserved for the mode part you have seen there are 6 modes, but this mode register has got 5 bits in it. So, you can, so that may be for future extension then there is a bit number 5 which stands for the T or the thumb instruction set. So, whenever this processor 8 0 this ARM processor it is using the thumb instruction set then this T bit will be equal to 1 and when it is using the ARM instructions set then this T bit equal to 0. So, there are some specific instructions by which you can switch over between the instruction sets and by looking into this T bit you can understand whether it is using thumb instruction set or ARM instruction set.

Then there is the bit number 6 is the F bit which stands for that FRQ mode or FIQ. So, if the processor is currently in the fast interrupt processing mode then this FIQ bit will be set to 1 and if it is in the normal interrupt processing mode then this IRQ bit will be set to 1. So, this is the CPSR register structure. You see that this much much simplified compared to say other processors that we have seen so far.

(Refer Slide Time: 22:25)



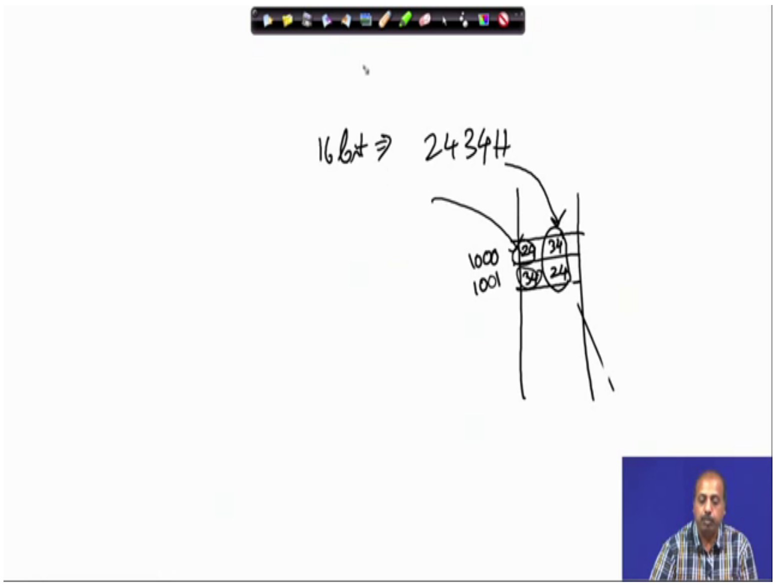
Data Types

- Six different data types
  - 8-bit signed and unsigned
  - 16-bit signed and unsigned
  - 32-bit signed and unsigned
- Supports both little-endian and big-endian format
- Most implementations support only little-endian

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Next we will look into the data types that are supported by this a ARM processor. There are 6 different data types 8 bit signed unsigned, 16 bit signed unsigned, 32 bit signed unsigned. So, we have got both signed and unsigned numbers it can be 8 bit, 16 bit or 32 bit, supports both little-endian and big-endian formats. In fact, there is a pin called big-endian. So, if you if you say this ARM processor has got a pin called big-endian. So, if that is activated, processor will take it as big-endian format otherwise it will take it as a little-endian format. So, what is it like? So it is a it is nothing, but a convention ok. So, if you have got a say an instruction.

(Refer Slide Time: 23:17)



Some data say one say 16 bit data or 2 byte data where the number is say 2434 hex; now if you are storing it at memory location 1000 ok. So, one location. So, if the memory is 8 bit then one location is not sufficient for saving the number. So, you need to have it stored in two locations 1000 and 1001.

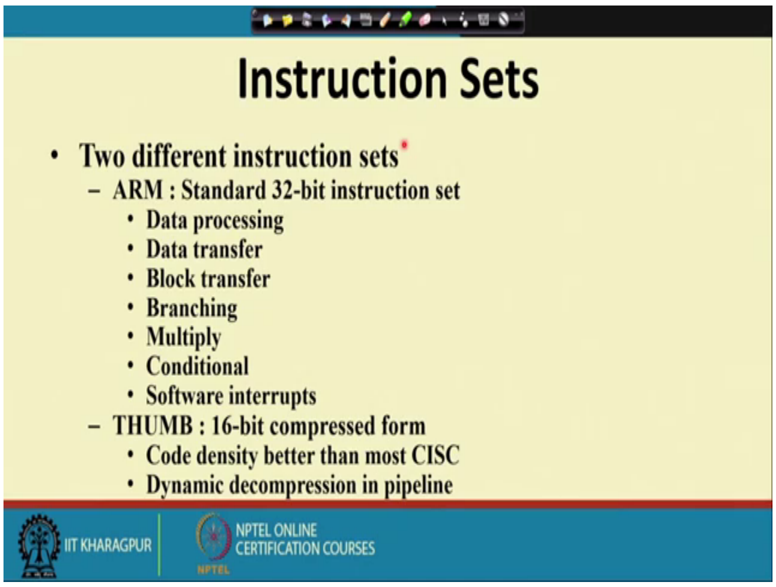
Now, in one convention, so we can put this 24 here and 34 here and in another convention I can store 34 here and 24 here. So, in this case in the second case what has happened is that this lower order byte it has been saved in the lower order memory location and this higher order byte it has been saved in the higher order memory location.

Whereas, in the first case in the first case this lower order byte has gone to higher order address and the higher order byte has gone to the lower order address, but operation wise there is no difference, but these are the two formats that are followed, so Intel follows this format and if you look into the Motorola series of processors. So, they will follow this processor. So, ARM has got both the facilities both the little-endian and big-endian.

So, this little-endian means it ends with the little one. So, this is that, so this is 34 will come to the lowest address and that way.

So, that way we will see some example later that will carry differentiate between this little-endian and big-endian formats. So, but ARM supports both and most of the implementation they support only little-endian because supporting both is of no use ok. So, and as I said that this ARM gives licenses to the manufacturers, so they will select one of these formats and accordingly it will be the data path will be synthesised for that.

(Refer Slide Time: 25:18)



Instruction Sets

- Two different instruction sets
  - ARM : Standard 32-bit instruction set
    - Data processing
    - Data transfer
    - Block transfer
    - Branching
    - Multiply
    - Conditional
    - Software interrupts
  - THUMB : 16-bit compressed form
    - Code density better than most CISC
    - Dynamic decompression in pipeline

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Next we will look into the instruction sets. So, as I said that there are two different instruction sets in ARM, one is the ARM instruction set which is a standard 32 bit instructions and we have got thumb instruction set which is 16 bit compressed form of instruction.

Now, these instructions can further be classified as data processing instruction where it is doing the processing like say add, addition, subtraction, multiplication etcetera. Then data transfer instruction transfer between say the register pairs, register and memory like that. Then block transfer you can move a block of data from one portion of memory to another portion we can do branching ok.

So, we can go from one location to another location branch from there. Multiplication then conditional execution and software interrupts. So, these are the various categories of ARM instruction that we have.

Thumb instructions are more close to the standard microcontroller and microprocessor instructions that we are familiar with. So, it is a 16 bit compressed format. So, in 32 bit we have got two instructions. So, if you are accessing the instruction so you will be getting two instructions at a time. So, that way the code density will be better and it is better than many of the CISC processors.

So, that way it is a very nice thing that code density means per. If I for a particular function, if I have to write say 100 lines of code in one processor and 50 lines of code in another processor in the second processor has got a better code density than the first one. So, it is measured in terms of say memory bytes how many bytes are needed for storing the program ok. So, that way it is the code density will be better.

And there is a dynamic decompression in the within pipeline form, so ARM thumb instructions are not executed directly. So, they are converted into ARM instruction and then executed. So, it is basically in the form of there is there is a decompression that takes place. So, that way thumb instructions are going to be slower than the ARM instructions.