

## **Computer Organization and Architecture: A Pedagogical Aspect**

**Prof. Jatindra Kr. Deka**

**Dr. Santosh Biswas**

**Dr. Arnab Sarkar**

**Department of Computer Science and Engineering**

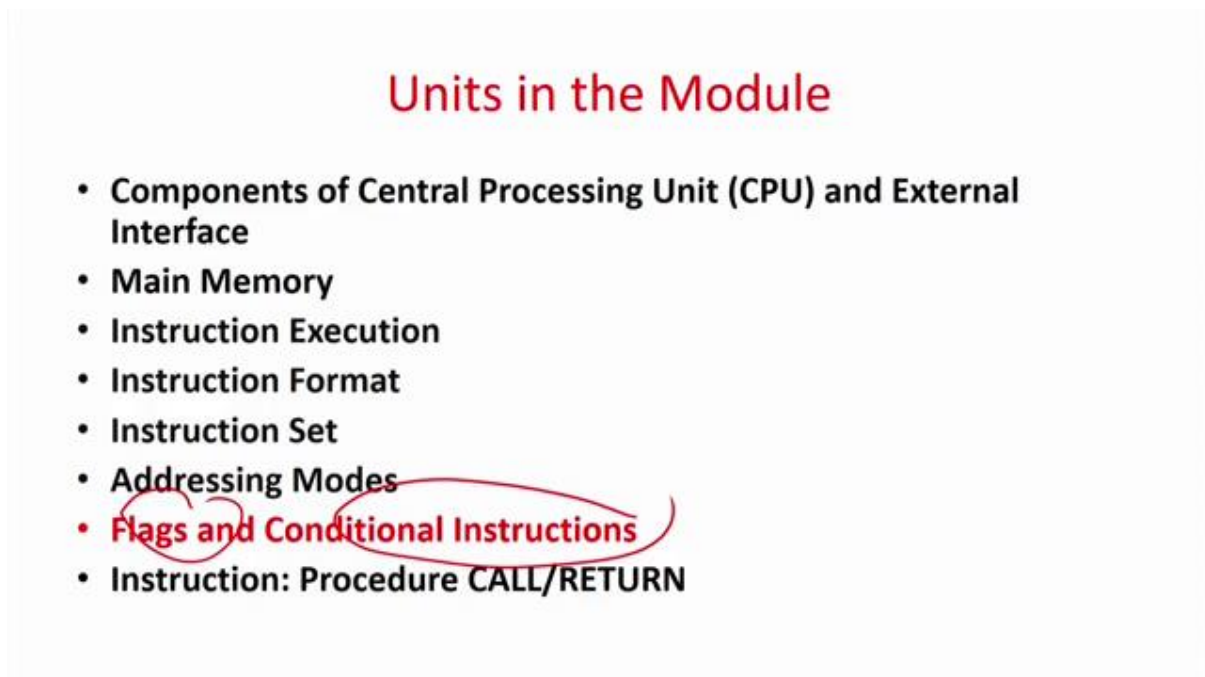
**Indian Institute of Technology, Guwahati**

### **Lecture – 13**

#### **Flags and Conditional Instructions**

Ok. So, welcome to the next unit on the module on addressing mode, instruction set and instruction execution flow.

(Refer Slide Time: 00:30)



**Units in the Module**

- **Components of Central Processing Unit (CPU) and External Interface**
- **Main Memory**
- **Instruction Execution**
- **Instruction Format**
- **Instruction Set**
- **Addressing Modes**
- **Flags and Conditional Instructions**
- **Instruction: Procedure CALL/RETURN**

So, ah till now whatever we were discussing basically what are the different type of instructions? How it looks? What are the formats? What are the different components of an instruction? And in the all the cases if you observe we were just assuming or thinking that the instructions would execute in a very sequential manner.

That is first instruction may load something from the memory then it may do some memory operation, addition operation, subtraction operation and then it will again write to the memory. So in fact, we are assuming that everything would go in a very sequential flow, but as all of us

have written some c code or any high level language code in our life. So, we are its very obvious to us that there is nothing which is very sequential in a code.

That means, every time you will have some logic and it will depend on some input conditions and based on that you will either jump to another instruction or even continue; like a loop based on some instruction you will go back to the loop and based on the exit condition you will exit out of the loop.

In other words a very very important concept which we need to understand; when you are looking at the instruction format and execution of instructions is that; we always have some conditions and based on the condition some of the instructions will execute the next instructions or jump to some other instruction, which will be executed once the conditions are satisfied. In fact, they are called as the conditional instruction and without a conditional instruction no coding paradigm is complete.

So, along with conditional instruction in this unit we are going to look at flags and conditional instructions. So, conditional instructions are very similar to our like or if then else statement while conditions, jump, loops, etcetera, but whenever we say something like ah if  $x > y$  then do something.

So, in high level language we have a condition like  $x > y$  and how a condition is internally checked in a hardware or in your CPU is basically depending on certain kind of flag registers or the registers are there and there are some bits which are actually set or reset depending on some conditions. And your conditional instruction actually checks those flag and then decide what to do. So, this unit will be dedicated to such conditional instructions and what are the flags and how they coordinate and how they are executed. So, this is about the unit number 7 of this module on flags and conditional instructions. So, as we are doing looking at hardware the term flag is actually coming with the conditional instructions.

(Refer Slide Time: 02:56)

**Module**  
**Addressing Modes, Instruction Set**  
**and Instruction Execution Flow**

**Unit-7**  
**Flags and Conditional Instructions**

But in a high level language version we generally talk about conditional instructions like if then else, for loops etcetera.

(Refer Slide Time: 03:04)

**Unit Summary**

The most common way of generating the condition to be tested in a conditional branch instruction is the use of flags. The flag code register consists of individual bits that are set or cleared depending on the result of an operation that is carried out by execution of an instruction. These bits are used to remember the effect of computation after executing the current instruction. These bits are then used as conditions for next conditional instructions.

Sign, Zero, Carry, Auxiliary carry, even parity, overflow, equal etc. are some of the most common flags.

So, as the whole course is ah based on pedagogical aspect; so, what is the unit summary? So, or what we are going to look in this unit. So, basically this unit will be mainly focusing on that there are certain instructions whose executions are not sequential, they actually depend on

conditions. So, actually they as you all know there is something called program counter which will help us to know what is the next instruction.

So, if there is no conditional instruction as such then the program counter increments by 1, but whenever there is a conditional instruction based on the truth of the condition; the program counter value changes and it jumps to the required instruction.

So, there are two type of conditional instruction that is conditional branch and unconditional branch. Conditional branching means from statement  $x$  you go to statement  $y$  or you just execute statement  $x + 1$  next instruction depend on some condition like if  $x > y$ , then you execute the next instruction and if it is false you jump to some other memory location and execute the instruction there.

But there are some unconditional jumps also ah unconditional statement that you come here and then jump to such and such memory location execute the instruction there without waiting for any condition or without validating any condition; that is jump unconditional that is just like a function call. So, you come over here without any condition you just execute the instruction which is corresponding to the function which is being called.

So, they are actually unconditional jump instructions. So, in this unit basically we are going to look at ah what are the conditional instructions? How they actually change the program counter? Where they are stored? What are the internal dynamics of it? And basically there are two types of conditional statement branch and unconditional branch and conditional branch. And then we will look at the basic heart or basically what actually ah determines this condition.

In high level language we say  $x > y$  or the means say loop continues from 0 to 10 and at every point we increment the counter by 1, but when the counter which is 10 then we exit. In hardware how actually it is reflected? So, it is reflected in terms of certain flag bits which are some registers called the flag registers and inside the flag register there are certain bits allocated for some important parameters like sign, zero, carry, parity, overflow, equality etcetera.

So, what they are basically? When some conditional instructions are executed like say add and then certain flags will be set in the flag register depending on the value of the operation say for example, if I subtract two numbers and the answer is 0.

So, in that case the flag bit corresponding to 0 will be set if I add two numbers and the sum is say 5. So, in this case the zeroth flag which will be reset because the answer is more than 1. Then there is something called even parity so, if the answer is 5; 101 it's an odd parity number. So, this even parity bit will be reset and so forth.

That means, in other words in this unit we are also going to look at certain flag bits. And how they are set and how they are reset depending on the arithmetic operation just before a just after going for this instructions corresponding to addition, subtraction, equality checking etcetera. And then we will see how the jump conditional jump instructions basically execute by looking at the value of this bits in the flag register. So, what are the objective of the unit?

(Refer Slide Time: 06:25)

## Unit Objectives

- **Comprehension: Discuss:--**Discuss about flag bits and how these flag bits get set or reset.
- **Synthesis: Design:--**Uses of flag bits to design conditional statements.

This is a small unit and the objectives are basically to discuss that after this object code after this unit as a comprehension you will be able to discuss flag bits and how this flag bits are set and reset. The flag bits are heart of any kind of a conditional instruction and you will be able to discuss in as a comprehension that what are the bits? How are this bit set? And how are the bits reset? And then you will be able to design this is a synthesis objective using this flag bits you will be able to design conditional statements.

That is based on the flags because no conditional instructions can be executed without the flag bits. So, using these flag bits; so, what other conditional instructions can be designed? Like for

example, if you have a zeroth flag then we can have instructions corresponding the equality checking that if two numbers are compared or if I subtract two numbers and then if the answer is 0; then the zeroth flag will be set then I can have a subtraction instruction just before it and then I can say jump on 0.


That means, say for example, I have a counter and I actually want to increment the counter till 10 and just after executing of the each loop, I will decrement or I will increment the value of the index by 1. And as I check as I check I will subtract the index ah with 10. So, whenever the answer is 0; that means, the loop has reached up to 10 and it should exit.

So, in that case I will use the jump conditional jump on Z, but just before that I will subtract the index variable with 10 and if it is zero the zeroth flag will be set and I can have a special instruction called jump on 0. So, this is actually a synthesis objective just by looking at the flag bits, you can decide what other instructions can be designed for this conditional codes.

(Refer Slide Time: 08:09)

### Program status word

- The program status word (PSW) is a part of memory or registers which contain information about the present state of a program. By storing the current PSW during an interruption, the status of the CPU can be preserved for subsequent executing after returning from the Interrupt Service Routine.
- Error Status of the programs
- Pointer to the next instruction to be executed
- Sign bit of the result of the last arithmetic operation.
- Zero bit, which is set (reset) when the result of an arithmetic operation is 0 (no zero).
- Carry bit, which is set (reset) if an operation resulted in a carry (no carry) out of the MSBs of the operands.



So, before we start off this one. So, we know that whenever we talk of a jump instruction then what basically happens you are executing certain set of code or you are in a certain temporal part of a code. And as a jump instruction you generally you can go and serve a procedure or a function. So, before we jump from the main program to some other function or from one location to some other location the current context of the code has to be saved. So, therefore,

that is certain what are the temporary memory location, what are different register values at this time, what was the accumulator value at this time. Say for example, you have loaded some variable you have added with something and stored the value in the accumulator and just after that before saving it to the main memory, you got a procedure call and you jump.

So, whenever I come back and re execute from this one. So, what basically happens? The program counter say is that memory location 5 in each case what I have done I have added something with accumulator and stored the value in accumulator now the sixth location was storing the accumulator to the main memory.

But the sixth memory locations what happened? Just before that ah sorry the fifth location may be a accumulator operation, then there is a function which may be a conditional operation that depending on something you execute the next instruction or jump to a function.

The next instruction may be to store the value of the accumulator to the memory just, but just before that there was a conditional instruction based on something you jump. So, just after the accumulator operation at memory location fifth; you add something else to the accumulator, but next was a conditional instruction. So, without saving it to the memory you have a executed a procedure call.

But then what happens the accumulator will also be used in the procedure code; so, the intermediate value of this sum which in the accumulator will be lost. So, what do you have to do? Before you go to the executing the function you have to store the value of the accumulator, you have to remember the value of program counter that now I am in 5<sup>th</sup>, sixth was the conditional instruction.

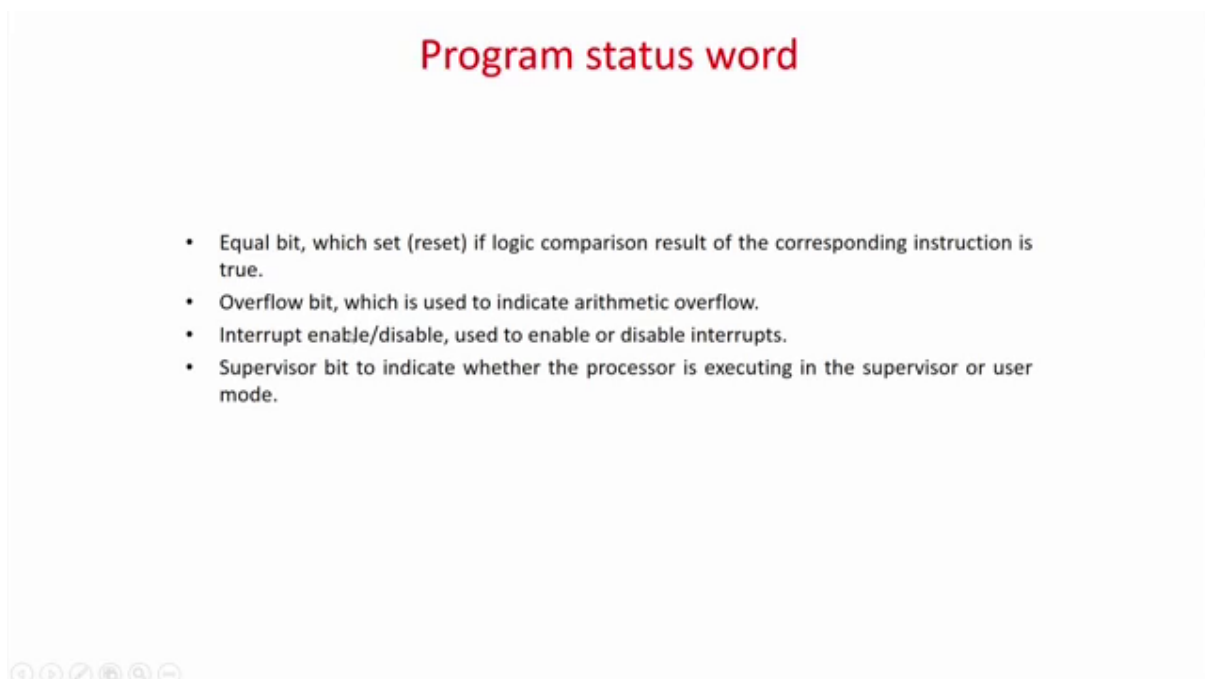
And after I come back I have to start executing from memory location number 7 the instruction that will save the accumulator value to the same memory. So, you have to also remember that what position that is if I jump from this location to this location. So, I have fifth is accumulator operation sixth is your conditional jump, seventh is again you are storing back the value of accumulator whatever you have done over here in the main memory. So, after doing this function you have to again come back and execute from memory location 7. So, when I am jumping you have to remember that I have to come back to memory location number 7 whatever intermediate accumulator value has to be stored in the memory register memory. And so many temporary variable or context of the program has to be saved. So, that when I come after executing the function I can recollect everything back and reload the registers accordingly.

So, there is something called a program status word which is a part of the memory or registers which contains information about the present state of the program by storing the current *PSW* during interruption or procedure call then everything is saved and then you come back after the executing your function you can get back the whole code and all the intermediate values refill and start executing from we have left.

So, what are the why you are so, *PSW* is so important when you are studying about jump instructions because jump means you may leave the context of the current code and execute some other context and again come back and re execute from where I have left. So, the whole intermediate state of your program is saved as a *PSW* and it we recollect that.

So, what the *PSW* has? It has lot of components some of them I have listed error status of code, pointer to the next instruction to be executed like in this case it is 7, where I have left sign bits, zero bits, carry bits, reset bits, overflow bits and so many other things which is listed over here.

(Refer Slide Time: 11:58)



### Program status word

- Equal bit, which set (reset) if logic comparison result of the corresponding instruction is true.
- Overflow bit, which is used to indicate arithmetic overflow.
- Interrupt enable/disable, used to enable or disable interrupts.
- Supervisor bit to indicate whether the processor is executing in the supervisor or user mode.

So in fact,. So, when we slowly advancing in different other newer modules on interrupts, but then ah actually on memory operations then we will be looking at more in details about what exactly the *PSW* stores, but for the time being you can see what they are, error status pointer pc is next value, all the flag bits current value of accumulator, if there are some if there are

some registers R1, R2, R3 which are the user defined registers all those variables you will store it as a program status words.

So, whenever I come back I can recollect everything and I can reuse. So, that is one very important thing that just before executing a jump to a function or an interrupt service routine we store the program status word.

(Refer Slide Time: 12:34)

The slide is titled "Condition codes or flag bits" in red. It contains two bullet points. The first bullet point is circled in red and describes the condition/flag code register. The second bullet point describes the Zero flag and the "Jump if Zero" instruction, with "Jump if Zero" circled in red. Handwritten red notes include "SWB" and "307" at the top right, and "3MB" at the bottom right, with arrows pointing towards the text.

### Condition codes or flag bits

- The condition /flag code register consists of individual bits that are set or cleared depending on the result of an operation that is carried out by execution of an instruction. These bits are used to remember the effect of computation after executing the current instruction. Also, these bits are used as conditions for next instructions.
- For example if a subtraction instruction results in a zero, then the Zero flag is set. This bit remains set until another instruction that affects the zero bit in the condition code register executes. Now, if the next instruction is "Jump if Zero", then it checks the zero flag. As the zero flag is set the condition of the instruction holds and jump takes place.

Now that is about the background that before I ah I am doing some work now I just go and do something else. So, before that actually I save my work in some intermediate memory and when I come back I can do that.

Now we are going to the real crux of these ah I means conditional instructions that is flags. So, this was about bookkeeping; so, once you have more details about bookkeeping etcetera we will be doing when you are doing the module on interrupts that is on I/O then ah because the whenever there is an interrupt you have to leave the main code and you have to go and service the interrupt. So, you have to store the context of the main code ok.

So, leaving aside that that is just a pointer which you have to keep in mind for the time being, then we are going to something called the heart of ah this jump instruction that is the code or flags. The code or flag is a register basically it is something called a flag register it has

individual bits which are set and reset by some execution of an arithmetic operation or a logic operation just before the setting of it.

In other words there is a register which is a flag register. So, whenever some arithmetic operation or logic operation happens; the corresponding bits are set or reset. For example, if a subtraction instruction leads to zero then the zeroth flag is set; that means, there are a lot of flags like zeroth flag, parity flag, sign flag, overflow flag etcetera.

So, based on some arithmetic operation of a subtraction addition; the corresponding bits are set and reset and for many cases basically some of the flag bits are set or reset, but they are not taken into picture. For example, if I am doing a subtraction operation  $a - b$  so obviously there will be no carry generated because unless I add two negative numbers or add two positive numbers the carry will not be generated. That is there is no overflow a carry can be generated as we will see in 2's complement arithmetic, but overflow is not generated. So, even if the overflow flag is set or reset if I am just adding two or one if I am doing a subtraction operation that is I subtract a negative number for positive number or one number is positive and one number is add negative we are adding it that is two sign numbers are there.

But of the opposite sign and we are adding it then the overflow flag is immaterial the flag will be set or reset based on different conditions as we will see, but it will be a don't care condition, but a very concrete example is given over here if I subtract two numbers and the answer is zero then the zero flag is set and then you can use it for an instruction like jump if zero there is jump on zero.

So, if there is an instruction; so, if I have an operation like say SUB say 30. So, what it is doing? It is taking the value of memory location 30 whatever is in the memory assuming it to be a direct instruction. So, ah it will go to the location 30; find out what is the value in memory location 30, subtract with the accumulator and store back; you see if the content of memory location 30 is equal to the accumulator you will get a zero over here.

And immediately zero flag will be set, now with the next instruction you can have a conditional instruction you can see that jump if zero. So; that means the memory location has the value of 10, 30 memory location has the value 10 and accumulator is an index of a counter which is also got 10 you subtract it then the loop has been completed and you have to jump out of the loop.

So, you can say have an instruction called jump on zero just after the abstraction instruction. So, it will jump and go out to some other memory location because the instruction has been satisfied conditional instruction will be satisfied because the zeroth flag will set.

(Refer Slide Time: 15:58)

Condition codes or flag bits		
FLAGS	RULES TO SET/RESET	
	SET	RESET
<b>S (Sign)</b> This flag is of importance if the arithmetic is signed	If the result of an operation is positive or 0.	If the result of an operation is negative
<b>Z (Zero)</b>	If the result of an operation is 0.	If the result of an operation is not equal to 0.
<b>C (Carry)</b> This flag is of importance if the arithmetic is unsigned	1) If the addition of two numbers results in carry out of the most significant bits. 1) If a subtraction of two numbers requires borrow into the most significant bits that are subtracted.	In all other cases.
<b>EP (Even Parity)</b>	If the result of operation has even number of 1's.	In all other cases.

So, now, we are going to see different type of ah what are the different type of ah typical in a typical CPU, what are the different types of flags? First is the sign flag; so, this is flag is of importance the arithmetic is sign; that means, if I am using unsigned arithmetic for the time being. So, this flag is of no importance if the operation that is if I am using the 2's complement arithmetic. So, if I know the MSB is 1; it's a negative number and if the MSB is 0 it is a positive number.

So, ah the sign bit is of importance if the arithmetic is signed; so, if the answer is 0 that is positive if the MSB is 0. So, it is set that is a positive number and if the MSB is a 1 then it's a negative number and it is reset 0. So, if you do some operation and the answer is 0; so, it is set and if the answer is not equal to 0 user is set.

So, of course, for any operation everything zeroth flag is very very important. Unlike if it is a unsigned arithmetic then the sign has no meaning; carry flag if two numbers if the addition of two numbers result in a carry out of the most significant bit it's obvious if I am having two

numbers say example 0111 and if I have a number 1000; unsigned arithmetic I am considering sorry if I take 11 sorry.

If I add 7 with 12 we are going to get 1100 and then the carry generated. So, in case of unsigned arithmetic such a carry is generated the more significant bit; so, the carry flag is set. So, if I am taking two if I subtract two numbers and there is some carry is borrowed.

Then also a sign carry flag will be set and in all other cases it is reset. So, in very simple words as I have given you an example if some carry is generated by adding two numbers at the MSB or if the subtraction of two numbers required a borrow at this then carry flag is set. Similarly even parity the results are generated; so, if the number of 1's is even the parity is set in another cases it is reset. Like in this case the carry is 1, but the 4 bit answer is 0011. So, it's a number of 1's are 2; so, it's even, so even parity is set.

(Refer Slide Time: 18:05)

Condition codes or flag bits		
<b>O (Overflow)</b> This flag is of importance if the arithmetic is signed	1) If the sum of two <u>numbers positive</u> (with sign bit 0) yields a <u>negative</u> number (with sign bit 1.) 2) If the sum of two <u>negative numbers</u> (with sign bit 1) yields a <u>positive</u> number (with sign bit 0.)	In all other cases.
<b>(E) Equal</b>	If the result of a comparison instruction is true	If the result of a comparison instruction is false
<b>Interrupt enable</b>	If the flag is set to 1, mask-able hardware interrupts will be served.	If cleared (interrupt enable set to 0), such interrupts will be ignored. However this flag does not affect the handling of non-mask-able interrupts.
<b>Supervisor mode</b>	If this flag is set, it indicates that the processor is executing in the supervisor mode. Certain privileged instruction can be executed only in supervisor mode and certain area of memory can be accessed only in supervisor mode.	If this flag is reset, it indicates that the processor is executing in the user mode.

Several others overflow flag; so, they tell you what is an overflow. Overflow will happen if the two numbers are positive or the two numbers are negative because a negative and a positive number will never generate a carry.