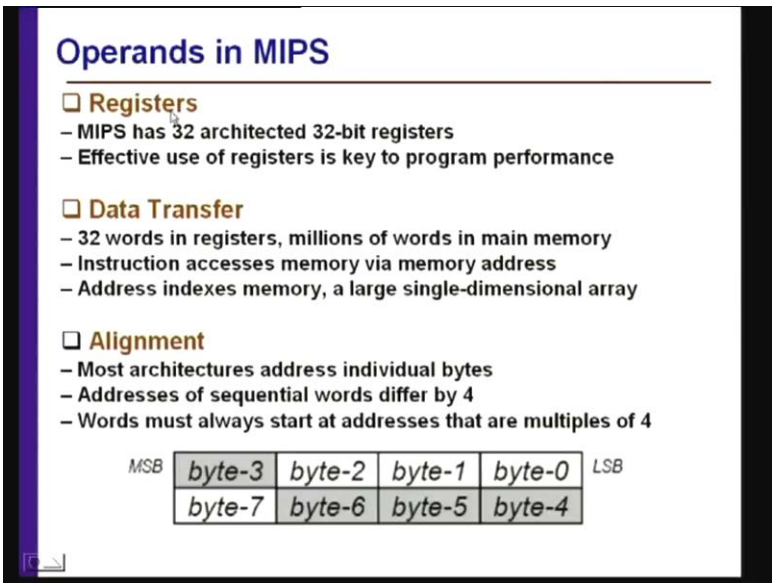
High Performance Computer Architecture
Prof. Ajit Pal
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 4
MIPS IPS and Processor

Hello and welcome today's lecture on MIPS and processor, so in two lectures I should cover this topic in the last lecture. I have discussed in detail what do you mean by instruction set architecture we have seen that the various components of a instruction e architecture and that instruction architecture access a specification for the design of processor. So, today I shall introduced you one processor that is known as MIPS, do not confuse with MIPS that is million instructor per second that is different this MIPS is name of the processor. So, I shall introduced you instruction set architecture and then I have discuss how you can realize the processor using that instruction set architecture as an input as an specification.

(Refer Slide Time: 01:53)

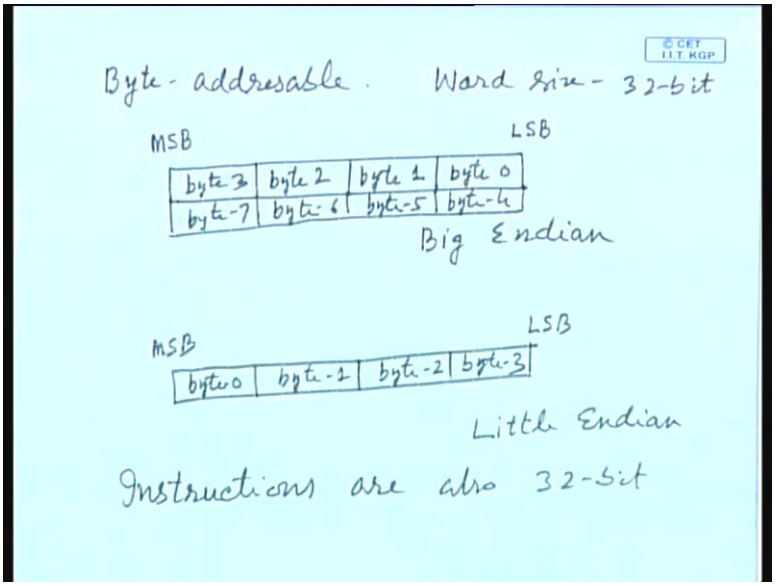


So, as I told the various components that you required for storing data number one is registers, so MIPS are got 32 architected 32 bit registers. We shall see these 32 be 32, 32 be registers are used for different purposes and effective use of the registers is key to program performance because how you are able to exploit how you are able to utilize.

This register will play very important role in deciding the performance of program that you write.

Then, you have to see the data transfer we have seen that 32 words in registers and the million of words in main memory. So, we will see it has got 32 bits address and as a consequence the number of words, I mean it is again by the addressable and the 2 the power 32 is the size of the main memory and instruction. Accesses memory via memory address and address indexes memory a large single dimensional array. That means the memory is accessed as a large single dimensional array and for that purpose you have to do some kind of alignment is most architectures at this individual bites as I told like any other processor mix also is byte addressable.

(Refer Slide Time: 03:31)

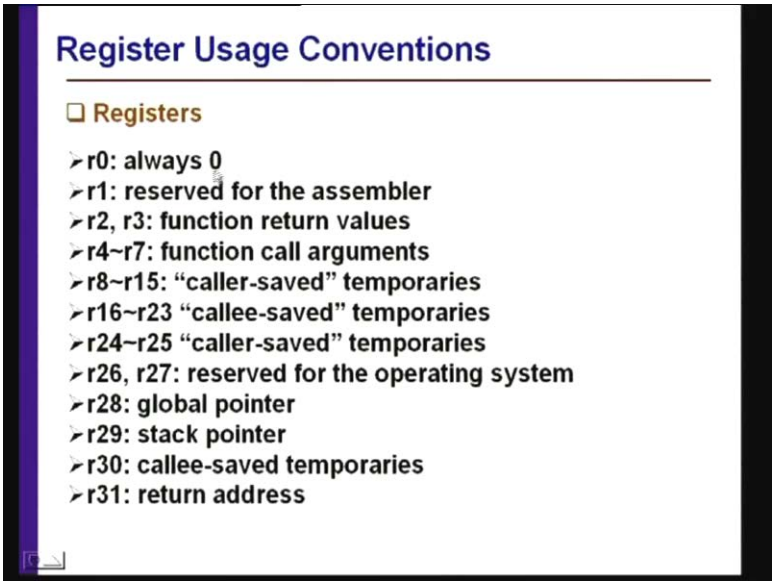


Now, whenever you do make it byte addressable your word size is now 4 bytes, so they will be 4 bytes present in a single word how do you really use be convention of aligning devise. For example, this will definitely will be this significant bit and this is the most significant bit, now this can be byte 0, this can be byte one this can be byte 2. This can be byte 3 and this approach is known as big ending and there is another possibility of course, in this case this is the first byte. Second byte will be like this that means it will again start with 0, 1, 2, 3 and byte 4 will be here in the second word and byte 5 byte 6 and byte 7, so this is your big Endean approach.

There is another approach which is known as little Endian in which case this 4 bytes are alignment in a little different way for example, what will be done there here it will byte 0 this will be byte 1, this will be byte 2 and this is byte 3. So, this your this is the LSB and this is the MSB, so this is known as the little Endian approach, so these two conventions are used, but you can see in case of your MIPS, they are using big Endian approach.

So, addresses of sequential words differ by four obviously 0, 1, 2, 3, and then the second one will be start from 7, so that means this will start from 0 and this will start with 4. So, this is and words must always start at addresses that are multiplies of 4 because of these obviously the address will started multiply of 4. That means 4 n 4 n plus 1 4 n 2, 2, 4 n plus 3, so n can carry from 0 to something whenever you accessing different addresses, so have been memory different words of the memory.

(Refer Slide Time: 06:25)



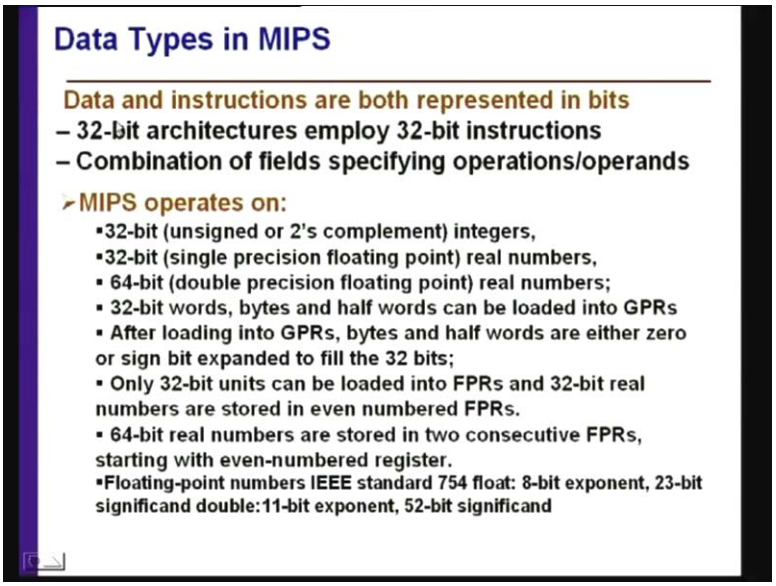
Now, as I told there are 32 bit registers and they are architected in a different way as you can see perform different purposes. For example, r 0 is always 0 you will be asking what the form of putting all 0 in a particular register the reason for doing it. You see they will be many situations query have to initialized either a memory location or a register with 0 when you do how keeping of a loop. There are many situations this will help you to write 0 in a particular memory location or in a register very quickly because already 0 is return in a register. So, if you copy those values into another register, it will take very small time, on the other hand if you write 0.

I mean if you read 0 by using imitate at the same, then you try to do that it will take longer time. So, that is the reason why a resister is a expressively a assign with value 0 and which can be use in many situations then register r 1 is reserved for the assembler r 2 and r 3 are the function return values. We will see that the registered are used for you know parameter parsing whenever you do in sub routine sub routine for function task. So, r 2 and r 3 is used to return the values by the function or the sub routine and r 4 to r 7, they are used by the by the sub routine to pass the parameter to the function.

That means main program can pass parameter to the function using this four register are 4 by 6 an answer, then you have got other registers like r 8 to r 15, 8, 9, 10, 11. There are 8 registers they are caller saved temporaries, so they are used to save temporary values in this registered then r 16 to r 23, there is called saved temporaries then r 24 and r 25 called saved temporaries. So, you can see these register can be used either by the main program or by the sub routine to stores some temporaries values and which can be which will make it very convenience to use register for storing intermediary results.

Then, r 16 and r 17 are reserved for the operating system and r 24 is used for global pointer and r 29 is used for stack pointer then r 30 used as called saved temporaries again and r 31 is used as return address sometime. It is called r a register that means the return at this whenever you do sub routine call that return at this same. You know the program counter value as same to be in a register in most of the situation it is step in stack, but in this particular case it is step in a special register that is r 31.

(Refer Slide Time: 09:55)

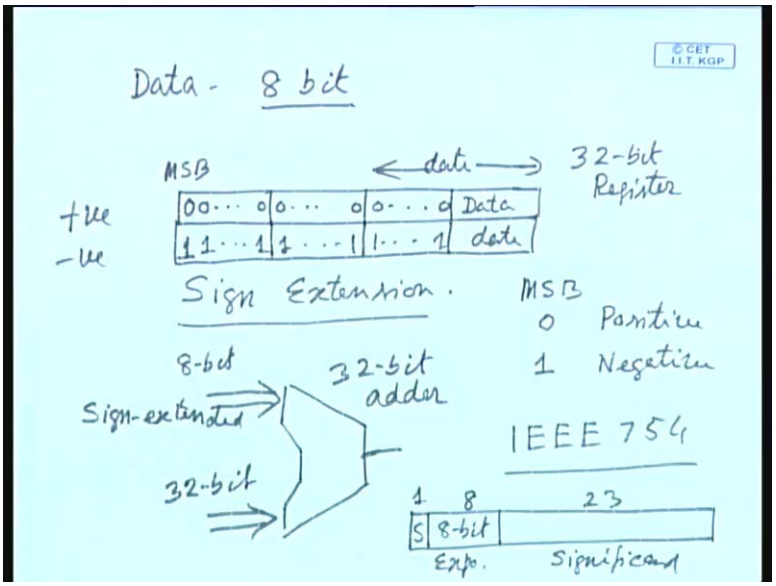


So, return and let us look at the different types of data types that MIPS can handle data and instruction are both represented in bits 32 bit architectures employ 32 bit instructions. So, not only your words size is 32 bit instructions are also 32 bit and all the instructions are 32 bit that means you have got a single size for instruction it is not variable length. So, the fix length instruction each of 32 bit and this is one of the important characteristic of risk processors and obviously MIPS processor and an consequence.

You know it uses a single format for not format single size not variable size of instruction and combination of fields specifying operations or operands. Later on, I shall discuss how 32 bits have been used in different addressing mode, then coming to the data types MIPS operator on 32 bits unsigned or 2's complement integers. So, it can perform integer processing either on unsigned data are two complement data, then 32 bit real number single precision floating point and 64 bit double precision floating point real numbers.

Then, 32 bit words bytes half words can be loaded in two general purpose registers, so we already seen it is got the large number of general purpose registers. This can be consider as general purpose registers and they can be use to same this bytes half words are either 0 or sign bit expanded to fill the 32 bit, now you may be here another concept you should understand.

(Refer Slide Time: 12:05)



Suppose, your data is 8 bit data is 8 bit now this is your 32 bit number data is 8 bit so this part will be field by data what will happen to the remaining by whenever you store an 8 bit data in a 32 bit register because your register is 32 bit a concept. It is known as sign extension is used as you know in case of sign tools complement number a whenever the most significant bit that is your MSB is 0, then it is a positive number, MSB 0 positive number and MSB is 1 the number is negative. Now, obviously this be doing 0 if the number is positive it is a positive integer, but what about the remaining bits.

So, you have got twenty three other bits what is done all this build up with 0s if it is a positive number. On the other hand, if it is a negative number, then most significant bit is one and these are all field of with ones and this can be your 8 bit data. So, this concept is known as sign extension that means your extending sign bit to fill up the remaining bit of the data so if this is the case for when data is 8 bit that is byte if the data size is 16 bit that is your half word in that case. Also, the same thing is done the concept of sign extension is used to fill up the remaining bits; you may be asking why is this necessary this is necessary.

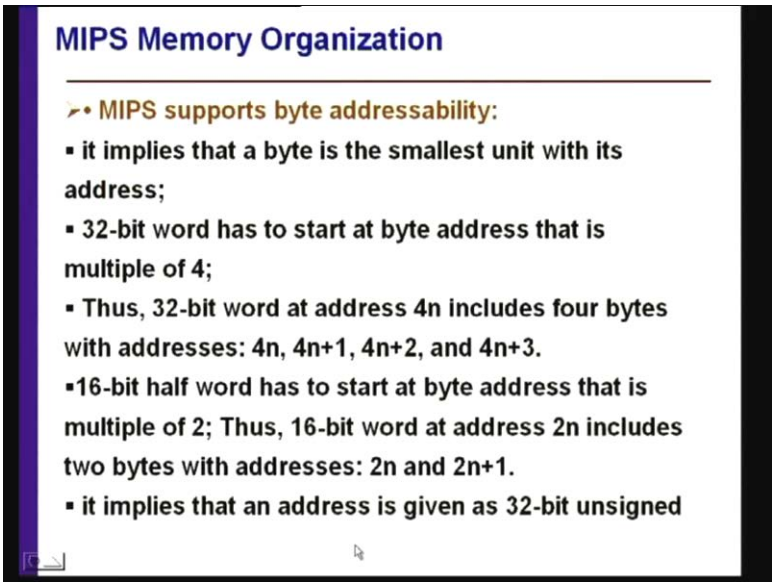
When will be doing say addition with the help of an adder you will applied one operand to one and another operand to another if this data is said 8 bit, but the adder is of 32 bit. So, obviously you can at add an 8 bit data with another operand it is 32 bit, so what we have to do we have to apply sign extended data to these so that you can apply both the

operands as 32 bits. You will get a 32 bit result, so that is the reason why sign extension is used so sign bit extended to fill the 32 bits. Now, only 32 bit units can be loaded into floating point registers and 32 bit real numbers are stored in even numbered floating point register and 64 bit real number s are stored in two consecutive floating point register.

So, we have seen it is suppose not only 32 bit single position floating point real number, but also 64 bit double precision floating point real number. That is the reason why you will require either a single register or two register depending on whether it is a single precision or double precision. There is some convention for representing floating point numbers this is this is known as IEEE 754 standard for representing floating point number, I request to capitulate this standard of floating point number where we will see how it is used for example are 32 bit that is a single precision.

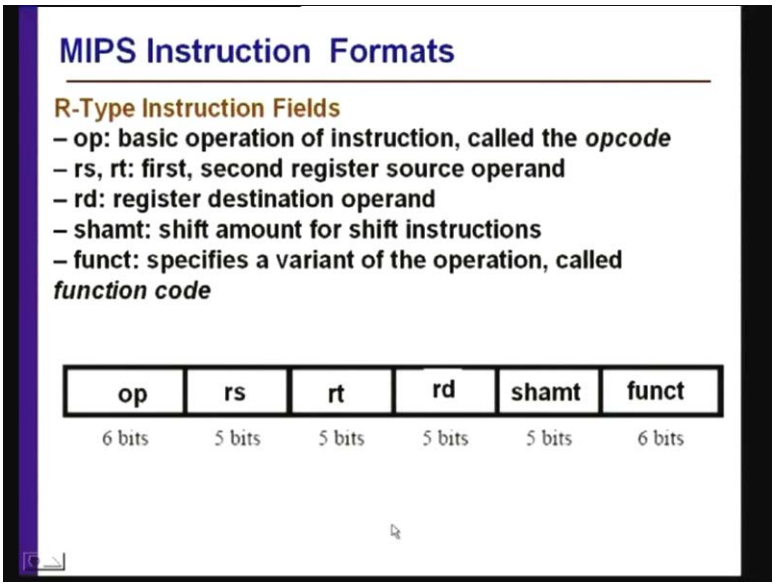
It uses 8 bit exponent and 32 bit significant and for double precision it use a 11 bit exponent and 54 bit significant. So, that is how the floating point number are represented with the help of IEEE standard 754, IEEE floating point number that means what we are doing we are we are dividing 32 bit. Here, we got sign bit and then 8 bit significant and remaining, sorry this is your exponent and then remaining bits are 23 bits significant. So, this is 23, this is 8 this is 1 that makes 32, so this is how the floating point number represented. The processor can do processor on the make processing can do processing on these different types of data by it word single to double precision single position floating point numbers.

(Refer Slide Time: 17:37)



So, this is the data types in MIPS and I have already explained MIPS supports bite addressability, it implies that a byte is the smallest unit with its address and 32 bit words as to start at byte at this address that is multiple of 4. I have already explain the thus 32 bit word at address 4 n includes 4 bytes with addresses byte 4 n 4 n plus 1, 4 n plus 2 and 4 n plus 3 and 16 bit half words to start at byte address. That is multiple of 2 thus 16 bit word at address 2 n includes 2 bites at this addresses 2 n and 2 n plus 1 at it implies that an address is given as 32 bit unsigned, so however although we at this will start this way your number has to be 32 bytes.

(Refer Slide Time: 18:22)

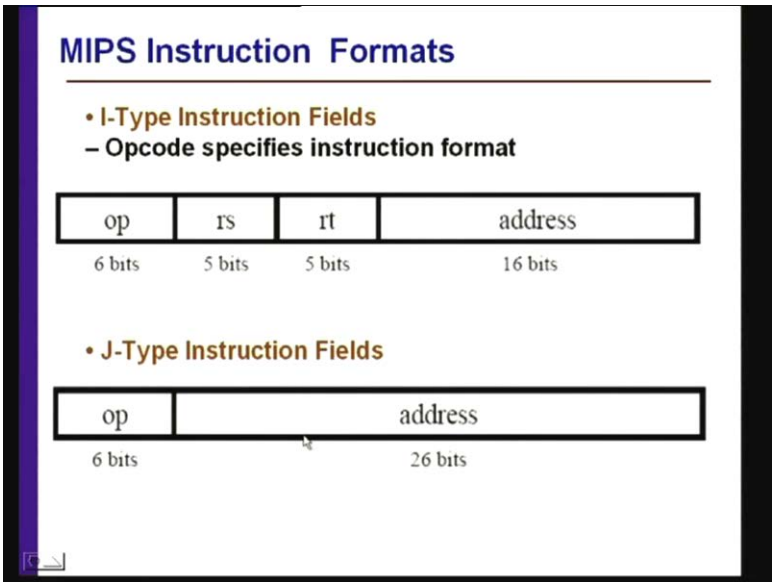


Now, let us come to the instruction formats I have already told that the instruction size is also 32 bit and how 32 bit instruction size has been used to represent different formats. Let us see I have discuss about different addressing modes in general term, obviously all these addressing modes that have discuss are not supported by MIPS. A subset of MIPS is supported MIPS, let us see what are the addressing mode that is supported by MIPS with the help of different instruction formats. Number one is known as r type of instruction fields, r types stand for register types that means your operands are in the register. Here, you can see the 6 bit is the op code field operation code and then r s r t, these are the two first and second register source of operand.

So, that means you will get two operands from to register r s and r t that r s and r t can be those 32 bit one, I mean two of the 32 bit register that I have already discuss and r d is again another general purpose. Again, it is the destination register at store, the result will be stored and this shift amount for sift instruction this one is only used when you are performing shifting.

So, you can not only do shifting by one bit you can shift you can shift a number is there you can shift by 2 bits by 4 bytes by 8 bites that can be done with the help of 5 bit number that can be specify with 5 bit number. So, you can shift up to 32 bit and as you can see this is the shift amount which is provided as part of instruction, but this is applicable only when you are using shift instructions. So, this field is not use for all other instructions is used only for shift instructions, then function specifies a variant of the operation called function code, so it is normally not use, but only in case of special situation this is used.

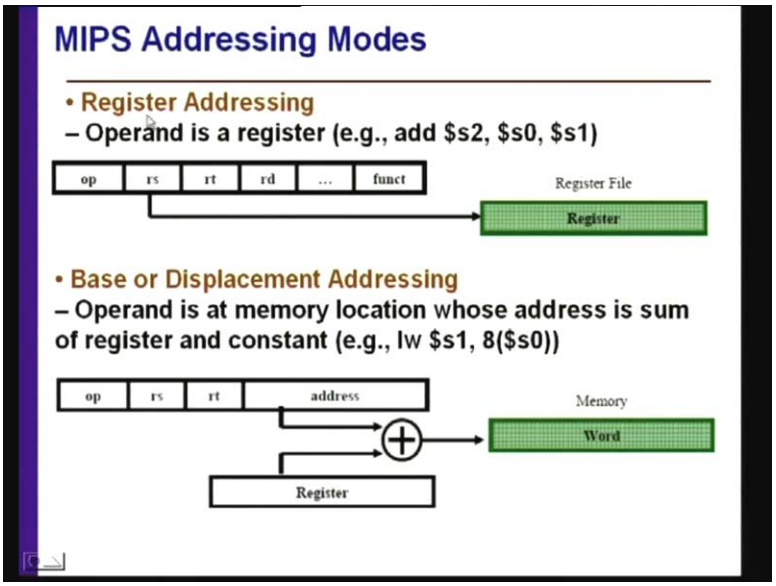
(Refer Slide Time: 20:41)



This is one type of instruction format second type of instruction format is known as I type instruction format. So, in case of I type of instruction format as you can see your op code part is remaining 6 bits and you have got two registers where the operands are available that is 5 bits. It is available here 5 bits, it is available here, then this will represent some memory address. We shall see how different addressing modes are implemented with the help of this shall come to this again.

So, how to this instruction format is use to generate and memory at this with the help of this 16 bit then there is a j type of instruction field which we got six bit op code followed by 26 bit address. So, it has got only three types of instruction formats, but all of them a 32 bit r type I type and j type. So, they are different fields are shown here and we shall see how they are used to generate the effective at this in different situation for example.

(Refer Slide Time: 21:47)



Register addressing in case of register addressing these three registers are really performing the source of operands and destination of result that means r s source of operands. So, it is pointing to one of the registers you have got a 32 bit register file, so it will point to one of the register for example, it can be s 2 dollar s 2 that is register dollar is 0 dollar s 1. That means the we can add the content of s 0 s 2 and s 1 s 0 and s 2 stored the result in s 1, so that will be the that will be the register addressing. So, where all the operands are involve the register and obviously here we using art type instructive format and coming to base or displacement addressing.

As you can see here operand is at memory location whose address is some of the register and constant, so here that 16 bit address field. You know this format we have seen 16 bit is provided here that 16 bit is added with the content of registered that is less register. So, it is called the base or displacement addressing, so there is a base register is 0 which is used so here your using s 0, but you can do some other register. Then, you are using this load word dollar s 1, so in your loading with the value of I mean content of a memory location into a register s 1, so this is essentially load instruction and load instruction will load some value from memory.

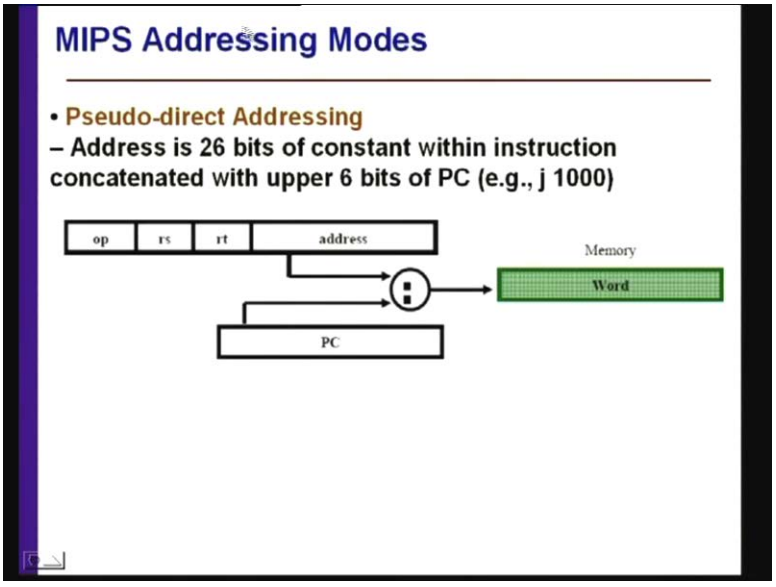
It will load into a registered and you can see the addressing how the effective generated addressing effective generated by adding the content of a registered with this displacement which is providing as part of the instructions. Now, let us come to

immediate addressing as you know an immediate addressing you provide the operand as the part of the instruction. So, operand is constant within instruction, so it is add I stands for immediate dollar s 1 comma dollar s 0 comma 4. So, 4 is the 4 is the constant, so you are adding the content of a register and this will be storing the result in the memory in the register, so this is immediate addressing.

So, we have seen that r s r t and this part is the that constant valued that is provided, then you can have p c relative addressing relative addressing is very useful particularly in situation where you will be doing let us say jam branch. This type of an instruction for example, branch if equal that means the content of s 0 and s 1 are same we will branch to memory location where that 16 which is provided here that as here I mean as the part of the instruction.

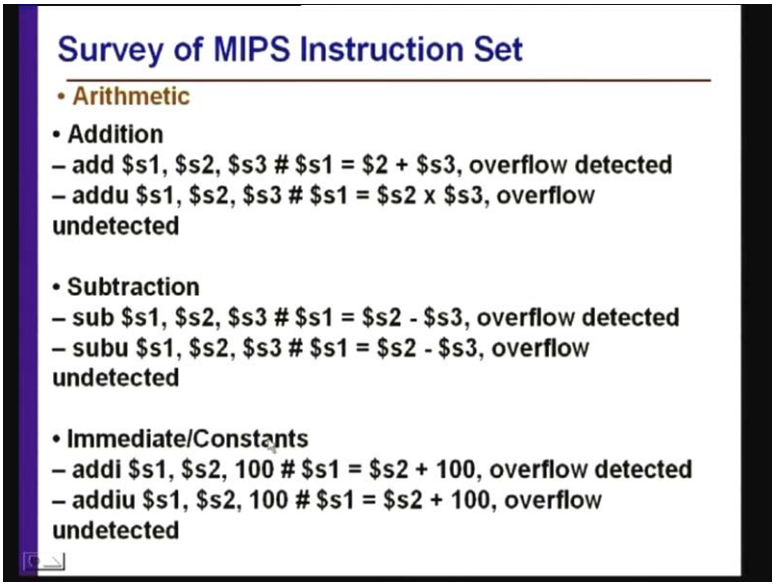
That will be added with the content of the program counter to generate effective address and i will branch to the memory location that is content of p c plus 16, 16 is provided here, I mean as part of the instruction. So, this is how p c relative addressing used to generate the effective address with respect to the program counter. So, this is useful in writing loops the program will be looping within a from one location to another with respect to the present value of program counter you can specify the displacement and accordingly it will be doing the looping.

(Refer Slide Time: 25:54)



Then, it has got pseudo direct addressing, so pseudo direct addressing why it is call pseudo direct addressing as you know in direct addressing that address has to be full address in this particular case full address is 32 bit. So, you know we do not want to increase the size of the instruction beyond 32 bit, so op code you have to leave 6 bit for op code. So, you are left with only 26 bit, so you have got address is 26 bit of constant within instruction is concatenated with 6 bit of the program counter. So, the six bit that 26 bit is concatenated with the program counter value 6 bit is the program counter value 6 bit will come from the 6 bit program counter concatenated and remaining 26 will be taken from instruction. That will generate the effectiveness, so this is how pseudo direct instruction addressing is done.

(Refer Slide Time: 27:09)

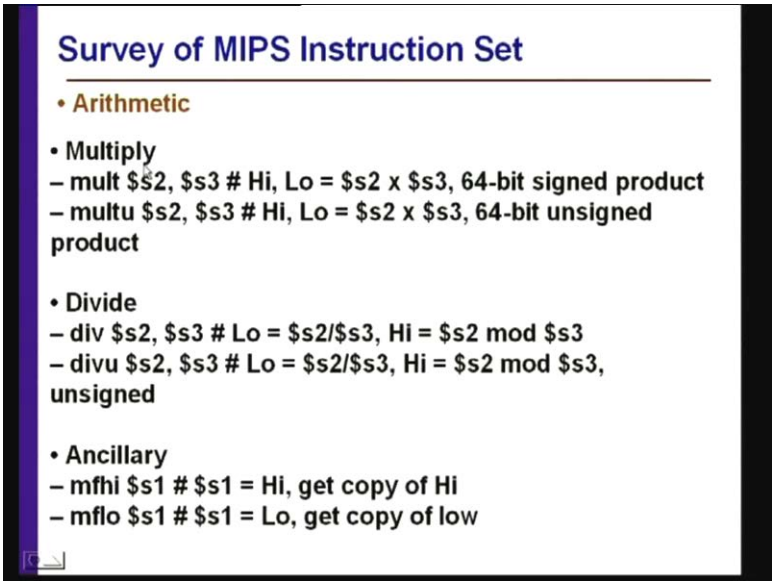


Now, let us look at the survey of MIPS instruction set I have discussed about the various data types I have discussed about different format, I have discussed about the various addressing mode. Let us discuss some represent instructions and from the instructions set for example, addition simple add with overflow detected add u overflow un detected. So, you can perform addition without detecting overflow and you can perform addition without detecting overflow.

Similarly, you can do subtraction detected and subtracted without detection of the overflow. Then, you can add with some immediate data as I have already told and that also you have got two a variation addition with a constant with over flow detected are

addition with a constant immediate value with over flow undetected. So, these are various arithmetic operations, but these are some of the representing you have got to multiply.

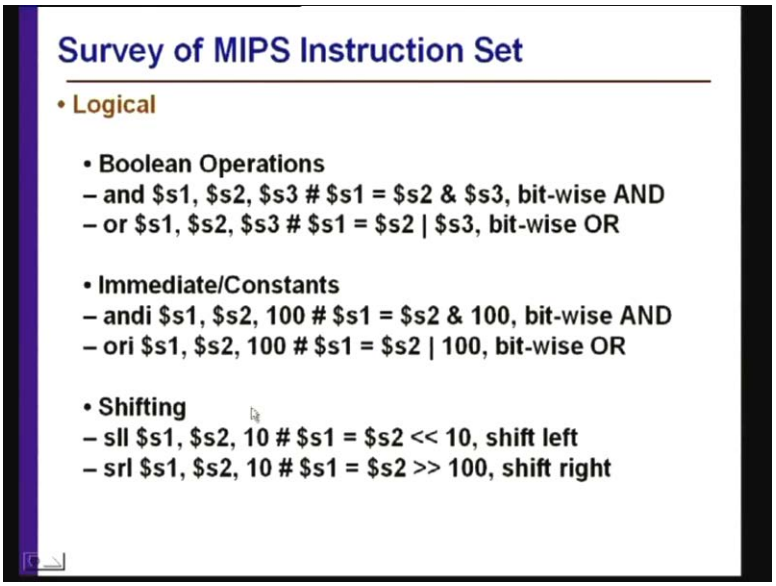
(Refer Slide Time: 28:24)



So, here we will see that two register that means s 2 and x 3, so s 2 and s 3 are having the operand and result will be store in a s 2 a because your result is 64 bit. Whenever you will get 64 sign product in the same register s 2 and s 3 by which is provided as a part of the unsigned product, then multiple unsigned product. Here, it is same except you are the result is unsigned the numbers are unsigned, then you can do division again two register involved two register s 2 and s 3 and you will be dividing that s 2 by s 3 so that higher byte it will be dollar s 2 mode s 3.

So, this is how it will be perform the defection and defection unsigned will be done in the similar way. There are some ancillary instruction m f h i is just essentially to get a copy hi get copy of hi that means you are higher order bit. That means copy lower adder bit getting copied, so these are some ancillary instruction.

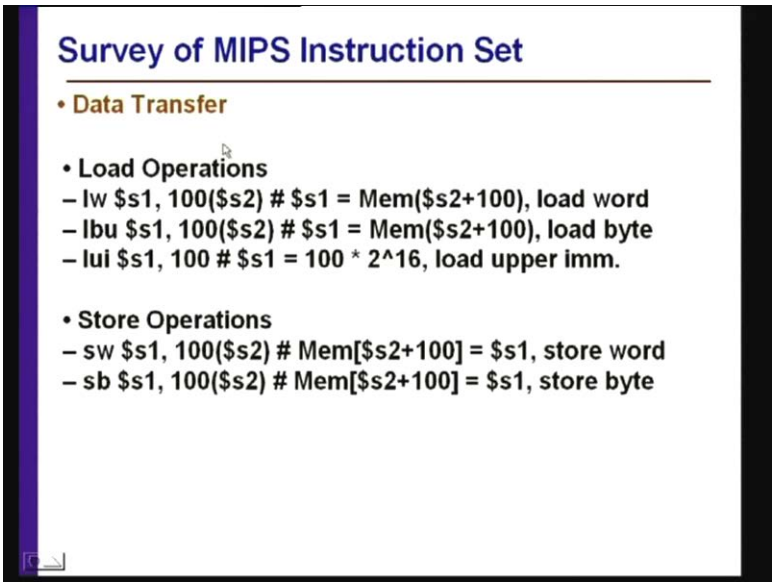
(Refer Slide Time: 29:55)



So, these are the arithmetic group then you add logical group and or it will perform bit wise operation on the content of register s 1 and s 2 and sorry s 2 and s 3 results will be stored in s 1 or dollar s 1, s 2 and s 3 that is here. It is or operation or dollar s 1 comma dollar s 2 comma s 3, it means that it will perform operation of the content dollar s 2 and s 3 and result it will be storing s 1 based on one. So, bit wise and or you can perform addition with immediate data again it will perform operations with the content of with the value that is provided as constants will be using here instruction format.

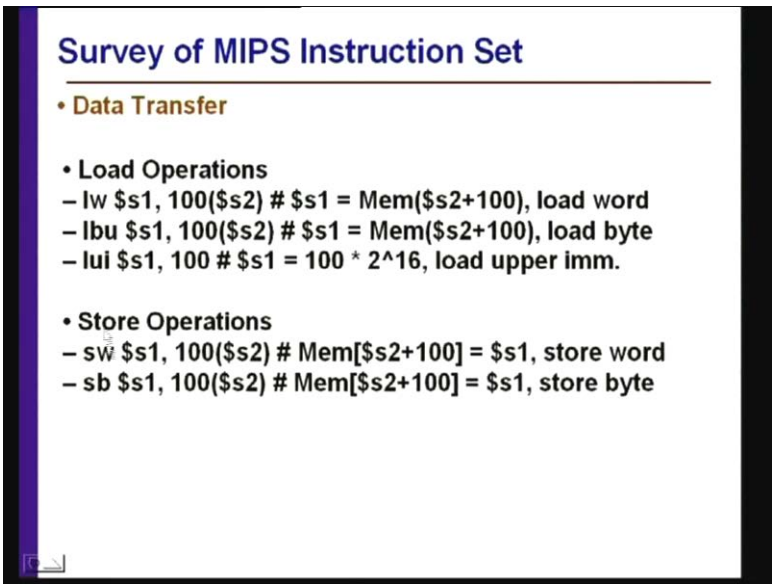
So, this hundred is provided as a part of instruction and it will do bit wise and or operation store the result in s 1 and the content of s 2 result is storing s 1. Similarly, you can do dollar s 1 comma s 2 comma 100 is the constants that is provided in the part of instruction. So, you will be doing dollar in the s 2 will be or bit wise is or operation in the hundred obviously inside extends of form and then it will store the result in s 1 and you can shifting. Here, you may recall that 5 bits where use expressively left for shifting operation so here you can do the shifting, so this ten the five bit is representing 10. So, you are doing shifting constant of s 2 by 10 bit and shift left your doing storing store the result in s 1. Similarly, here that 5 bit will providing 10, so it will do the shifting right shifting of the content of s 2 and then it will store the result in a s 1.

(Refer Slide Time: 32:05)



So, these are the logical operation then an apart from those are data manipulation on instruction arithmetic and logical operations are data manipulation group instruction.

(Refer Slide Time: 32:25)



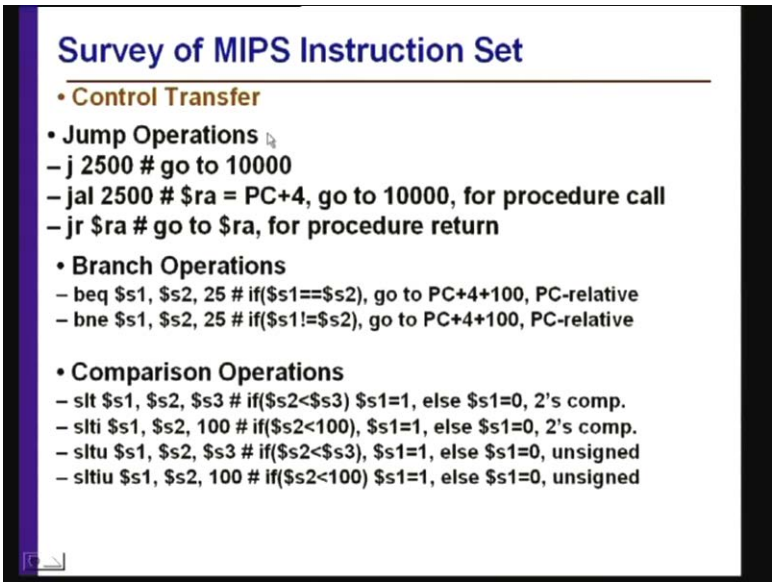
Then, you can do data transfer data transfer is essentially between memory to load and store that means from load means you will be loading the value from memory location to a registered and store is you will be storing value registered to memory location. So, you have expressive load store instruction and so load dollar s 1, 100 and within bracket dollar s 2 that implies that you are the content of s 1 will be taken from the memory.

Location effective address is generated by adding the content of register s 2 and adding that hundred is that 16 bit provided by using that I type instruction format.

So, this is load 1 and this is load byte you can load a bit same way and memory location is specify in the same manner. Similarly, you can perform load operation immediate, so lower and upper that means, so here you can load a byte you can load 16 bit you can load full word. So, all the three provided in this manner, so here it is load upper lower immediate and lower immediate. Also, there is another instruction, then store operations you can do store the content of a register into memory location.

Here, it is again using that I type instruction format, so the content of s 1 stored in the memory location by obtained by adding the dollar s 2 plus 100 is provided. The part of the instruction an immediate data, then you can to the byte storing here it is sword storing here, it is byte storing.

(Refer Slide Time: 34:19)

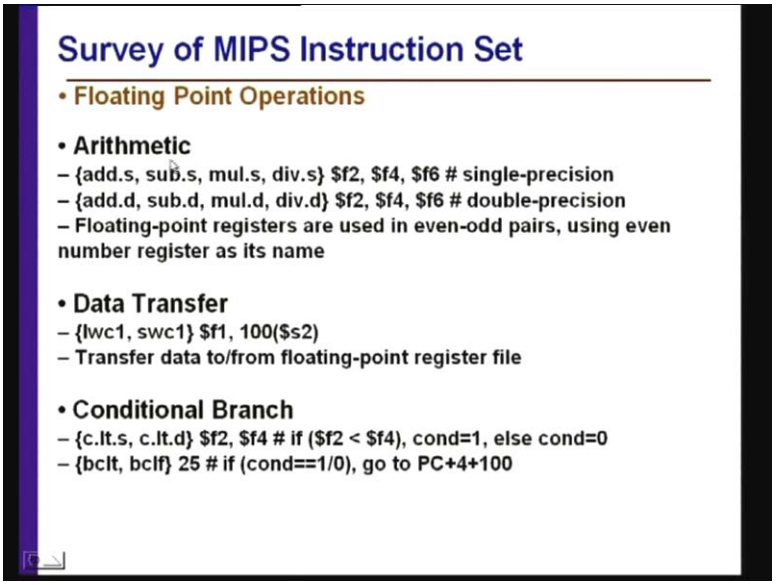


Then, you will be having several control transfer operation like jump branch, so jump 2500, go to 10,000, so you have to multiply with four we have seen these are bitted that is why your multiplying 2500 4 to go to this memory locations. Then, jump there is a procedure call again you are performing p c plus 4, so jump there is a procedure call go to ten thousand jump at memory location provided p c plus 4. So, r a you can see that last registered and that 31 that is acting as a return address, so you are adding the p c plus 4 that aggress. So, that is the return address whenever you are doing the procedure call.

Similarly, here also you can see go to return address for procedure return, so register r 31 is proving you the where the content of the providing store whenever you are performing subroutine call. Whenever you will return that time that value is taken from that register to jump to that memory location. So, it is not taking normally you know in many micro many processes it is taken from stack, so it is storing stack, but instead of stack here it is using special register which is part of an instructions.

Then, you can perform various comparison operations comparison operations, you can do, so different types comparison operations unsigned signed and so on. So, later on I shall give you some assignment for writing program in assembler language MIPS. So, you have to know about various addressing mode different types of instruction, so when you will writing program in a single language of MIPS.

(Refer Slide Time: 36:51)

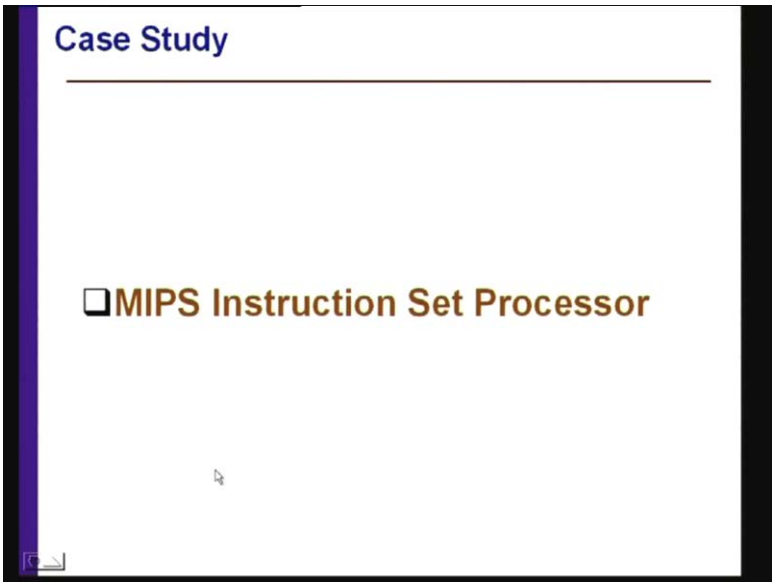


Then, it will perform various floating point operations, so various arithmetic operations single procession and double processions and floating points are used in even odd pairs, whenever you are using double precision you will be requiring two registers. So, even odd pairs are used and using even number register as its name, so you have to give the name of register, but it will involve the two register.

Similarly, you can do data transfer involving floating point operations, so transfer data 2 or from floating point register file that can be done and conditional branch can also be done based on floating point result. So, depending on this operation whether the

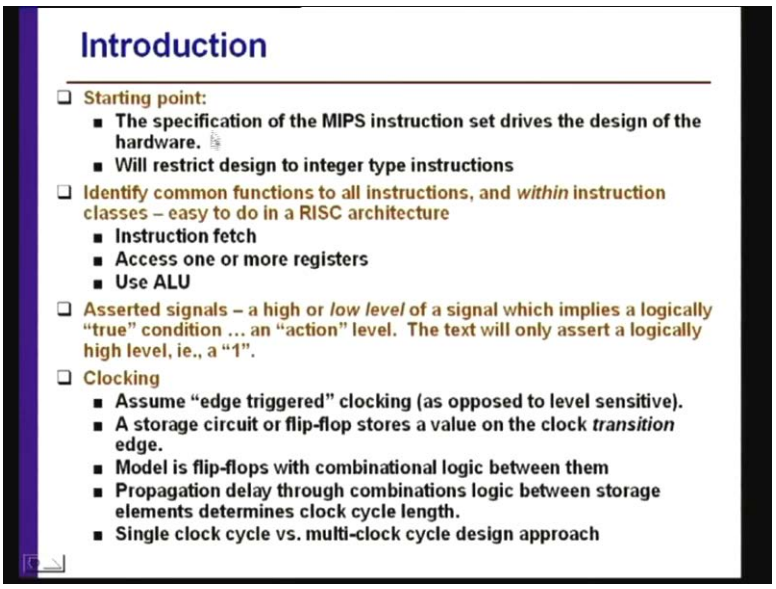
condition satisfied or not branch will take or not. So, this type of different conditional branches involving floating point operations is also provided part of the instructions.

(Refer Slide Time: 38:00)



Now, so far what I have done I have introduce to you instructions set architecture, now what I shall do we have to design a procedure which will implement the instructions that I have discussed, so I shall discuss about MIPS instruction set processor.

(Refer Slide Time: 38:27)



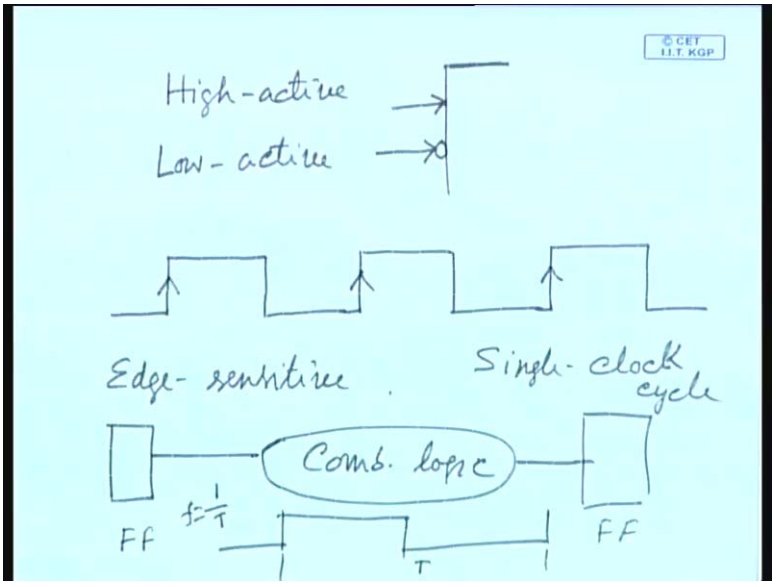
Here, it is the different points that you have to remember first of all starting point is the specification of the MIPS instructions set derives the design of the hardware. As I

already told instruction of architecture acts as a specification and that you have to the processor designer have to use those instruction to realize the to implement the processor. The second is and of course, to make the design simpler will restrict to integer type of instructions.

We have already seen MIPS, MIPS providing operations, so floating point we are not considering for the purpose of implementation to make to make the life simpler. We have only we shall only take into consideration integer type of operations not floating point then identify the common functions to all instructions and within instruction classes. Whenever you realize a hardware you have to do kind of optimizations and that is the reason you have to identify common functions to all instructions and within instruction classes so that you can do instruction fetch access one or more register use ALU.

So, particularly by using risk architecture it is easy to identify the commonality of the instruction and the various instruction classes and that will help you in simplify the implementation. Otherwise, you know if it is not a risk processor that implementation of the instruction will variable addressing i mean variable instruction format variable length of instruction and operations. It is extremely difficult an asserted signals a high or low level of a signals which implies a logically true conditions an action level that this here statement all I am trying to tell signals are high active.

(Refer Slide Time: 40:47)



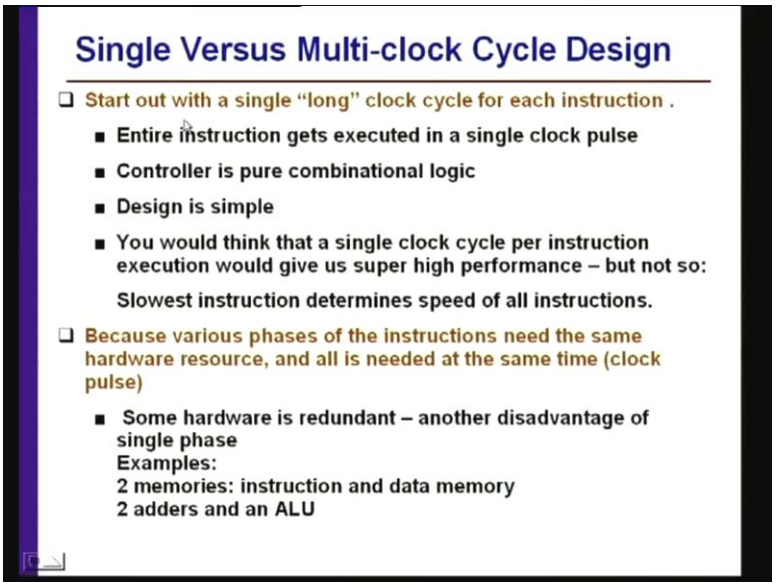
You know it can be make it low active we can make high active suppose this is your processor there is a input and that input whenever it is high active implies in this way. On the other hand whenever it is low active it is represented in this way there is a circle here. So, in this particular case it is assume that the inputs are signals are high active that means you will be using this kind of convention, whenever you will be generated statement instead of making low active.

Then, coming to the clocking we shall we are using edge triggered clocking as opposed to level sensitive you know by clock we may this kind of signal which will be generated by clock generated. It can be a crystal control accelerator which will generated fixed clock now it can be either level density or it can be in our case, it will be assume that this is edge generated that means all be changes will be taking place at the edges. So, this is edge sensitive because these conventions are assumptions are very important whenever you design a circuit because you have to use different components like flip flop. It has to be edge flip flop not levels flip clocks, so in storage circuit of flip flop value on the clock transition edge.

As I have already told model is flip flop is communication logic between them, so model is here you have got some combination logic. Here, you have got flip flop of storage element on either side, so your input is coming from flip flop combination circuit will be performing different things that is your processor that we shall realize and store the result in flip flop. So, that will be done in one clock cycles, now we assume single clock cycle that means the combination l logic delay of this will be such that it will be one clock cycle delay of the circuit will be same.

That means the clock frequency cannot exceed this, this is the time period clock frequency cannot exceed 1 by t because that is the delay of the combination logic if you use higher clock frequency, then you will get incorrect result here. So, that is the assumption delay made propagation delay through combination logic between storage elements determine the clock cycle length then single clock cycle verse multi clock cycle what I shall discuss. Here, we are restricted to single clock design, you can design the processor using multiply clock cycle and how it can be done, I shall explain later, but for the time being we are assuming that it is a single clock based.

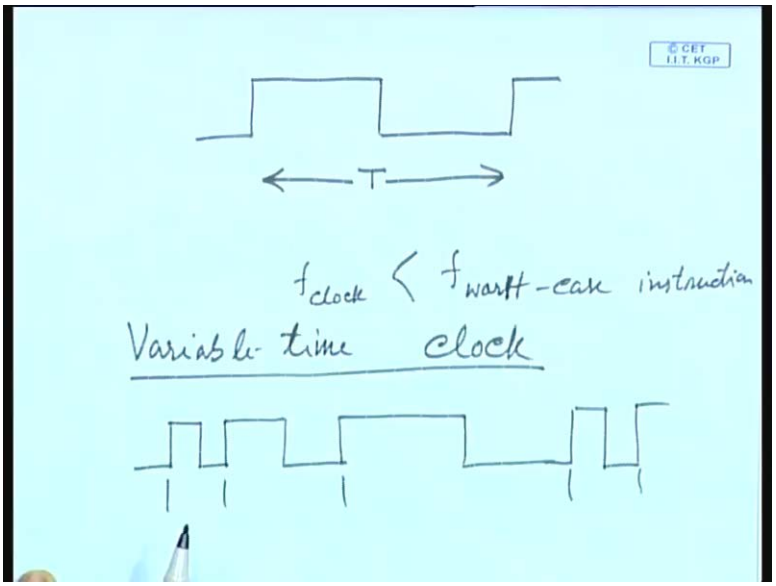
(Refer Slide Time: 44:20)



Now, we are starting with single long clock cycle for each instruction that means to perform each instruction respective of the type whether it is a data manipulation or data transfer or branch or control transfer. Whatever it may be all instructions we are assuming that they will perform in a single clock cycle and entire instruction gets executed in a single pulse controller is pure combinational logic.

Since, it is done in a single cycles o clock controller is a pure combinational logic and this is done to make the design simple. Later on, you know when we shall be considering other things like pipe lining more complex things. Then, you shall see design will be quite complex, but is the starting point to make the life simpler we are assuming that it is single clock pulse. Now, you may think that single clock cycle per instruction execution would give as super high performance.

(Refer Slide Time: 45:45)



So, what I am trying to tell you may assume that since everything each instruction executed in a single clock cycle we shall get very good performance, but unfortunately that is not true reason for that is the clock cycle frequency. This time period of the clock will be determine by the worst case delay of an instruction suppose you have got different instruction of different kinds and each of them will take different time to perform for example, if it is a simple data manipulation. Then, for example, that registered adding the content of register and storing the value in another register that may take some time.

However, when will be redeem load type of instruction we have to either load some value from memory to a register or store some value for register to memory. So, when it will involve memory it will take longer time so instruction are not of equal length in the sense execution will be different and your clock frequency f will be restricted a clock has to be less than the f. You know the case instruction in other words the slowest instruction will design the clock frequency.

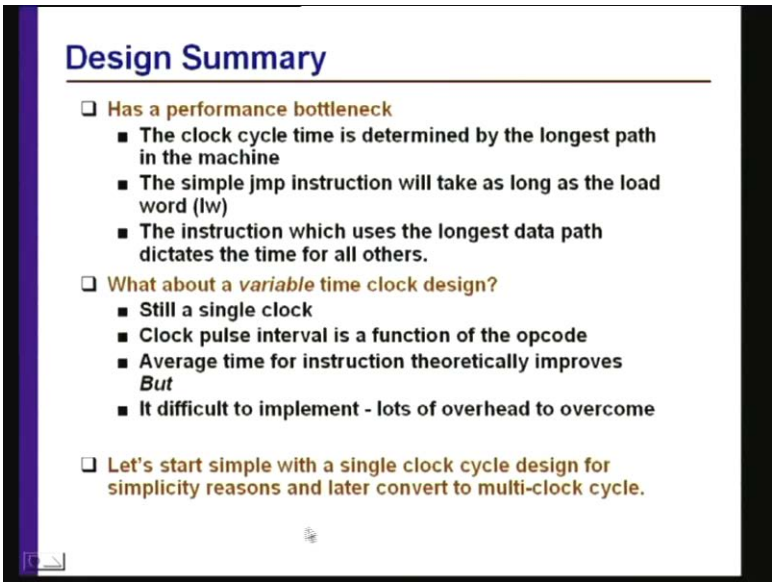
So, you are not really achieving a much and frequency, you know the slowest instruction determine speed of the instructions and because various phases of the instructions need the same hardware another important aspect is you know. Since, you are doing everything in one cycle you will require more hardware if you doing multiply cycle than there is a possibility of reusing some of the functional units for different purpose. For

example, for the calculation of address to calculate for the performing addition. So, these things can be done by using a single functional unit adder whenever you use multiply clock cycle, but if you use single clock cycle.

Since, it is done in one goal you have to duplicate adders and will see the redone more that means the hardware is redundant other disadvantages of single phase for example, we shall be using two separate memories, and one is your instruction memory. Another is your data memory one memory from where instruction should be fetch another memory from fair data will fair whenever your performing load and store.

So, if you use multiply cycle multi cycle design then you can have a single memory, but whenever it is a single cycle design you have to use two separate memories because you have to fetch instruction and also load or store data. So, you have to in a single cycle, so two separate memory will be required and as I already told 2 adders and an ALU.

(Refer Slide Time: 49:30)



So, here is our design summary, it has some performance bottleneck the clock cycle time is determined by the longest path in the machine the simple jump instruction will take as long as the load word. The instruction which uses the longest data path dictates the time for all others, so this slowest instruction will decide the clock frequency what about variable clock design. So, to overcome that there is one possibility that is your variable time clock, so earlier we are assumed that each clock is same duration this one this one

all are of same duration. Instead of that, suppose you are performing each step smaller time, so this is your cycle if it takes longer time will be like this.

It takes still longer time the clock cycle can be like this if it takes a smaller time another clock cycle smaller duration. So, what you are trying to do you will be generating clock cycle of different time period, so this is variable time clock. So, if you do that you are performing each single cycles, but each clock cycle is different duration you may be asking why not use this, but this will definitely complicate the design of the clock circuit.

Normally, you know you are using a crystal accelerator, so which will generate a fix frequency, but whenever you go for this kind of variable clock then the clock frequency is not fixed so your clock generated circuit will be very complicate. Also, you identify what will be your exact timing, so clock pulse interval is a function of the op code, so op code will design what will be the duration of this clock t 1 or t 2 or t 3 this will be decided the op code.

That means after you use the op code, the op code will specify that this is the clock frequency for this particular time for this particular instruction and accordingly the clock has to be generated. So, this will make the design pretty complex, so let us start simple with a single clock cycle design for simplicity reason and later consider multiple clock cycle and incorporate more complexity like pipelining and other things. So, like this let us stop here, let us stop here and we shall assume our discussion on the processor design in a next class.

Thank you.