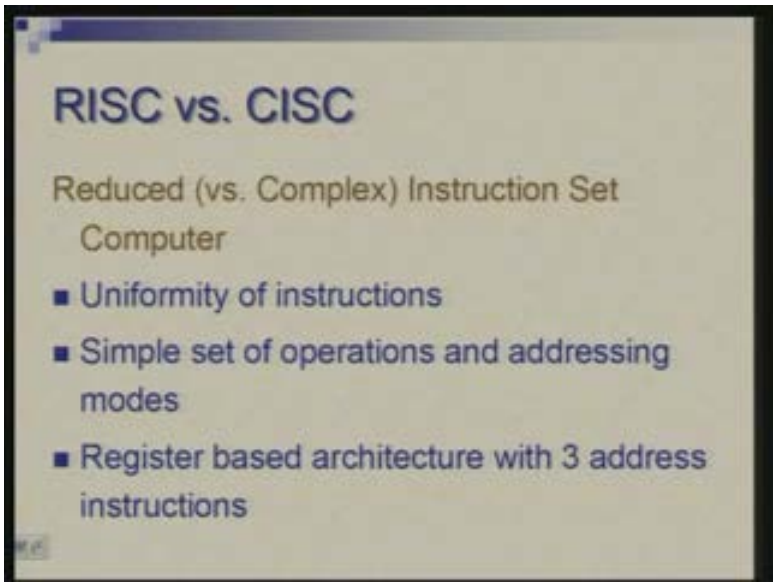
Computer Architecture
Prof. Anshul Kumar
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Lecture - 8
Architecture Examples

We began understanding instruction set architecture by taking an example of MIPS processor which is conceptually very simple. In the previous lecture, we tried to go beyond MIPS and saw in what directions architectural developments have taken place to bring in different variety of features. I also mentioned about RISC and CISC stacks of architecture where in one case the emphasis is on simplicity and efficient implementation whereas in the other case the emphasis on providing powerful features for the programmer.

Today I will take some specific examples. It is quite time consuming to take any architecture and study in detail. But what we are going to do is just look at a few salient features, particularly those features which are different from what we have studied in case of MIPS.

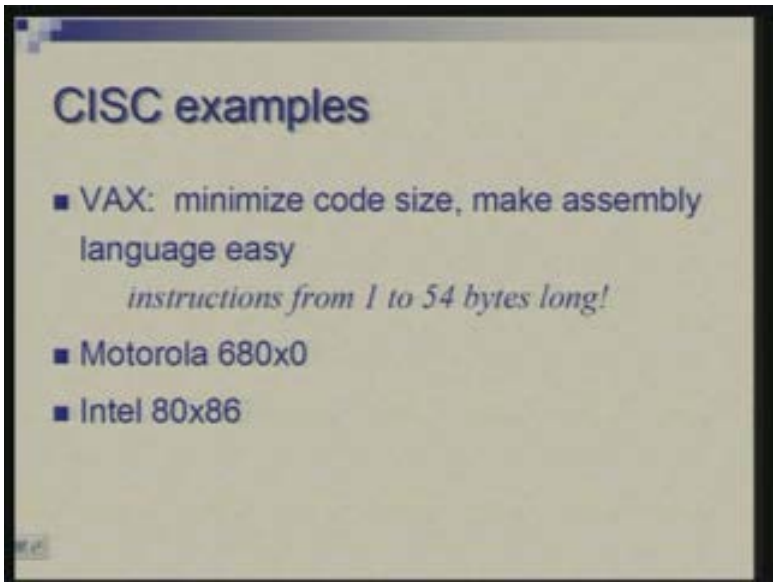
(Refer Slide Time: 1:57)



Just to recall RISC or the reduced instruction set computers basically deal with instructions which are very simple and emphasis on uniformity of instructions. There are very few formats and all instructions are expressed in terms of those; the instruction size is generally the same. Each operation tries to do one simple thing at a time and there is no attempt to do many different tasks within same instruction. The architecture typically provides several registers and each instruction performing arithmetic typically takes three operands; two for sources and one for the destination.

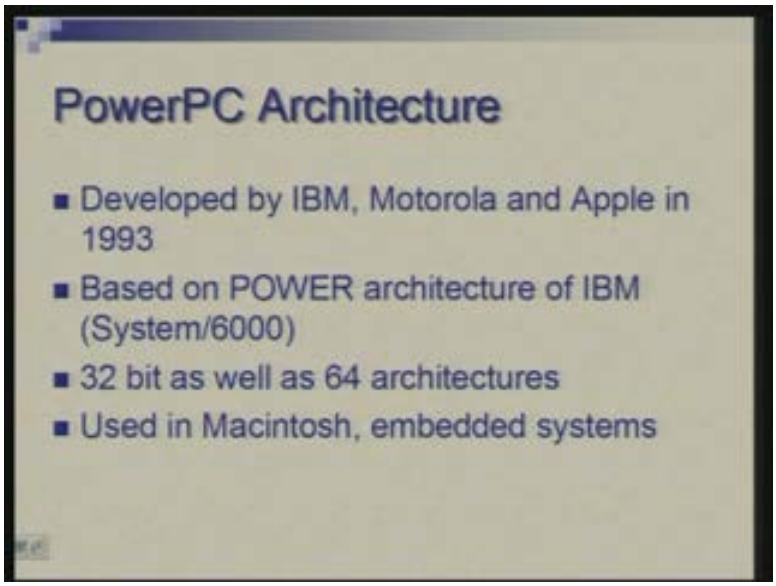
Some of the examples of RISC style of architecture i mentioned are SUN's SPARC, HP's PA-RISC, Motorola's power PC, DEC's alpha and even one earlier machine of 1960s namely CDC 6600 followed that kind of style. So out of these we will look at some more details of SUN's SPARC and also Motorola's power PC.

(Refer Slide Time: 3:22)



On the other hand, there are numerous examples of CISC particularly machines of earlier times 70s and also some of 80s; some of the 70s machine which grew to later machines. VAX is the climax of such development where the attempt was to have very compact code; minimize the code size and make it very easy for assembly language programmer. The variation and instruction size could be as much as 1 to 54 bytes. Another example is of Motorola 680x0. Again there is a series of processors starting with 68000, 68010 and so on and Intel's 80x86 starting with 80286 onwards and these machines have their origin where the idea originated from earlier 16-bit, 8-bit and 4-bit macro processors. So we will look at VAX and Intel 80x86.

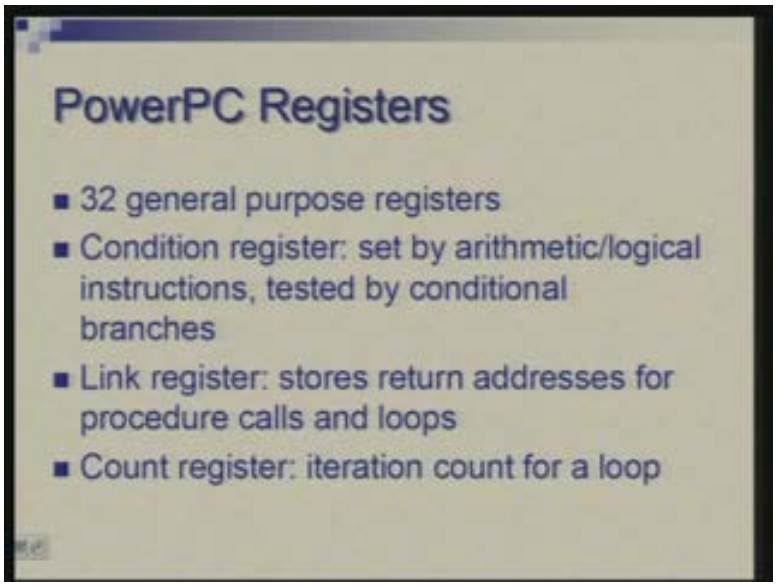
(Refer Slide Time: 4:36)



So, starting with power PC architecture this was an effort of more than one company including IBM, Motorola and Apple. So Motorola was primarily the semiconductor manufacturer; they have been along the Intel, they had been in the business of macro processor development right from early 70s and at that time they were the main rival of Intel. Apple started with small desktop personal computers and IBMs experience was in mainframes and later on PC's were based on x86 series. So these three major companies with somewhat different experience got together in 93 and defined an architecture called power PC. This had its origin in another RISC architecture which came from IBM system 6000 and that was essentially a machine used for servers.

So although initially power PC was 32-bit architecture their later versions in late 90s were of 64-bit kind also particularly 900 series. So this has been used extensively in Macintosh systems and also there are embedded versions which find a wide variety of applications. There are lots of similarities between MIPS and a power PC architecture; roughly similar principles are followed but power PC goes little further and has actually presented some instruction which are different from MIPS particularly in branch instruction area they have lot of variety.

(Refer Slide Time: 6:48)



So like MIPS they have 32 registers, apart from that there is a condition register. So condition register is essentially a register where different bits have different meaning and they are set or reset depending upon previous arithmetic or logical operations. In case of MIPS we have seen that there is an instruction slt which does comparison and the result is put in one of the general purpose registers. But in power PC and several other machines the result of instruction for example comparison or even arithmetic is put in special bits of a register or flags.

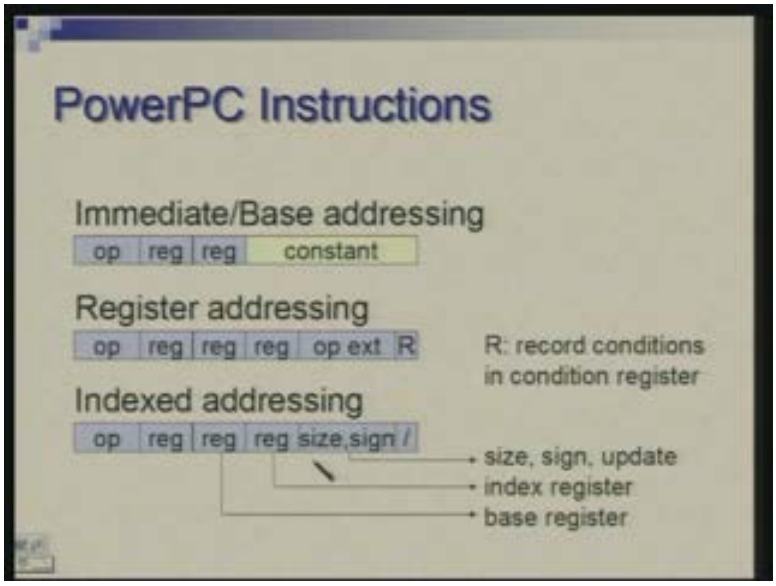
For example, when you are comparing one bit of the condition register would carry the information about the result whether A was less than B or not less than B or even if simply subtraction is done or addition is done whether the result was positive or negative or whether the result was zero or non-zero, whether the parity of the result was odd or even so there are lots of such conditions which are defined and they all combine together in a special register called condition register.

There are two other special registers which are worth mentioning one is the link register. So, unlike dollar ra which we have seen in MIPS there is a special register where link addresses are stored and this has a double purpose not only for procedural linkages it can also be used for loops. So for example, when you are beginning the loop, with the help of the instruction you can have that beginning of the loop address stored in this register and at the end of the loop you can simply use this information to jump back so you do not need to specify explicitly the address but you can simply refer to link register so it is also used to jump back to the beginning of the loop.

Then there is another register called count register which can be used to store iteration count for a loop. So there are special instructions which will allow this to be incremented decremented and tested so that facilitates execution of loop where the iterations are

driven by a count. So let us look at some of the instruction formats of power PC and also associated addressing modes.

(Refer Slide Time: 9:22)



So you would find again lot of similarity in the sense that there is a 6-bit opcode and 5-bit register fields. So, the first one which you see here is quite similar to the immediate addressing or base addressing which we see in MIPS, so it is exactly the same format. Again the base addressing is used for memory access and immediate addressing is used for arithmetic and logical operations to provide an operand. Then register addressing is slightly different in terms of format; the idea is same but the format is different.

Once again since the opcode the main opcode field is 6 bits you often need to extend this you need to have another few bits which will actually help you in expanding the number of instructions. So a group of instructions will have same opcode and they will be distinguished by opcode extension so that field is here and the size of this is…. you could see 16 minus 5 and minus 1 so this is a 10-bit field here.

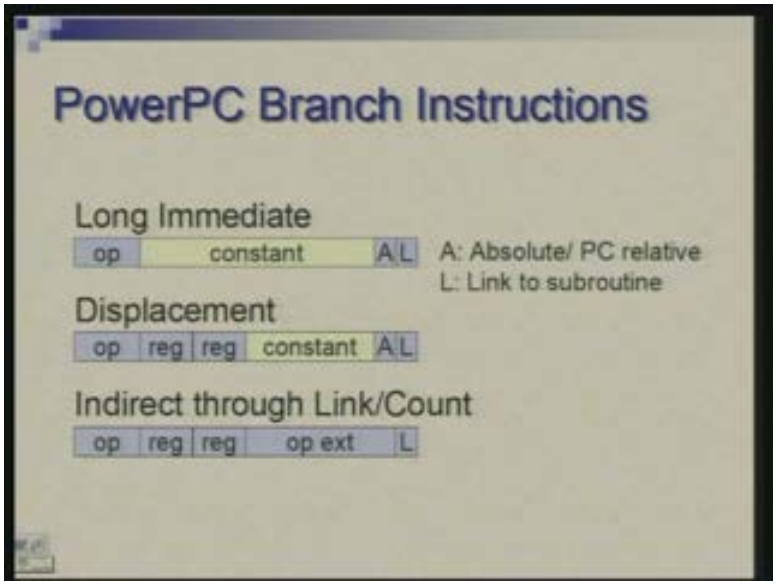
Now in this you notice a 1-bit field R which actually controls whether the instruction is to make an effect on the condition bits or not. you can… it is the same instruction, you can enable condition setting or you can disable condition setting by choosing one or zero in this particular bit. So what might happen is that you may set condition by some instruction and you may not be requiring to test it immediately there may be a few other instructions and then you may test it later. So what you would like is the instructions which are in between should not change the value of the condition codes. Therefore you need a control where you can precisely allow or disallow an instruction to modify the condition bits.

There is an additional mode which is used for accessing memory called indexed addressing. You would recall that I mentioned in context of MIPS that base addressing

and index addressing could only be a matter of different interpretation. Basically you have a constant part and something which is coming from register so basically the two are added but how you interpret the two components is different. The indexing here is taken little differently that you have base register as well as index register both specified and these two register contents are read. So one carries the base which for example could be starting address of an array and the other carries an index which would be corresponding to the index.

Now, since you may be accessing array of bytes or array of words you need further control so there is a field here which specifies what is the size of your data. When you are indexing you need to multiply it by certain constant indicating whether it is byte or half word or full word or double word and so on so information about size, the sign of the offset whether you want to add or subtract and after addition after adding this index do you want to update the base or not, so something like autoincrement autodecrement where address gets modified and the modified value goes back to the register so there is a control on that. This particular field (Refer Slide Time: 13:38) actually carries information about these three aspects and makes it very versatile.

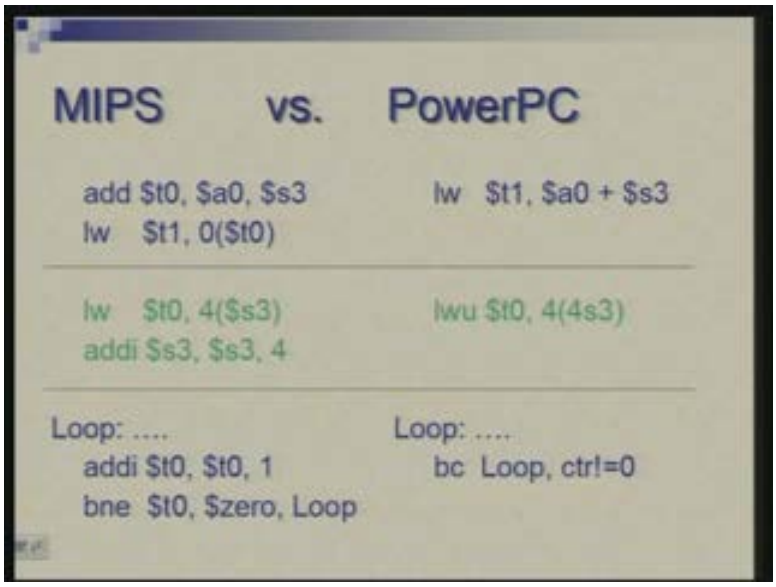
(Refer Slide Time: 13:46)



These are a few formats used for branch instructions. There is a long immediate in the sense that there is a long constant apart from 6-bit opcode and other two special bits, rest of it is available for a constant. That means a 24-bit constant can be used in instructions like jump so this is an this is used for unconditional jump instructions and this one, next one has a smaller constant so it is a fourteen bit constant all right so fourteen bit displacement can be specified; the bit A in both these cases indicates whether this displacement is absolute or it is PC relative. So whether this has to be added to PC to get the final address or this itself is the address and for that you have a control and L indicates whether you are making a link to a subroutine or not.

So basically transfer of control can be made to a subroutine by any of the branch instructions unconditional as well as conditional by setting this bit to one. So, as you have seen the difference between j and jl the only difference is that the return address gets saved in a specific register. Now, this facility is available with any of the branch instructions. For example, even with beq so you may make a call to a procedure if two values are equal or unequal. You may actually check a condition and make a call to a procedure. So this is indirect jump (Refer Slide Time: 15:36) if for example we had jr there so here you make a control transfer based on the contents of link register and count register. So count actually also…. Apart from having count in some context it can also be used as address. Here are a few examples which will show how some of these instructions will shorten the code which you write.

(Refer Slide Time: 15:58)



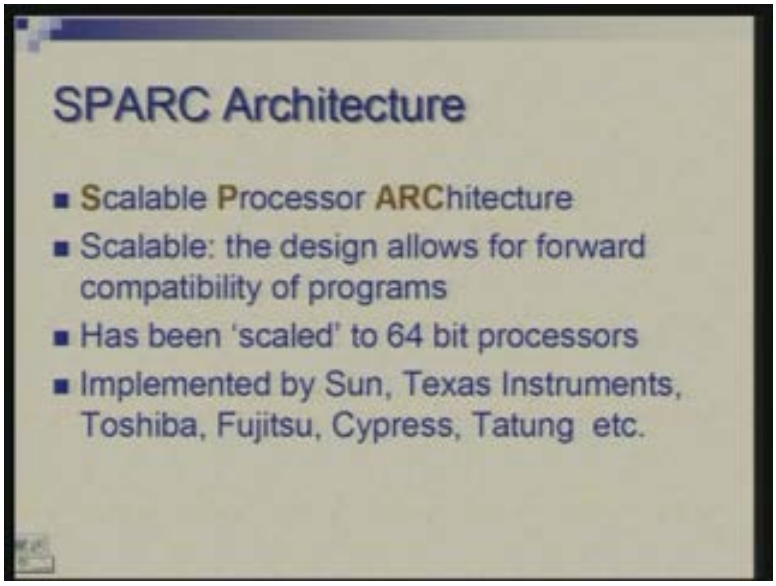
So, on the left side you see sequence of instructions in MIPS and on the right side corresponding instruction in power PC. In MIPS, for example, we often require address to be computed by adding two registers and then we use that in lw. So here we are typically we are wasting this constant field we do not use this in any case because the lw instruction does not have a mode where two registers can be added but with this index mode you can have a base and an index both added by the instruction itself.

Then next is the example of update. Here, for example, in MIPS you will say that you are accessing memory with s3, offset is 4, result is brought to t0 and then you update s3 by 4 possibily for the next iteration. But here with lwu which is update version…. there is lw and lwu means lw with update so this I am sorry this is dollar s3 (Refer Slide Time:: 17:30) so s3 is being used with this offset and that offset also updates it so next time s3 will be four more than what we have in this.

Here is an example of using a counter. In MIPS you will have a counter maintained in a register let us say t0 which you update and you are checking, for example you could start

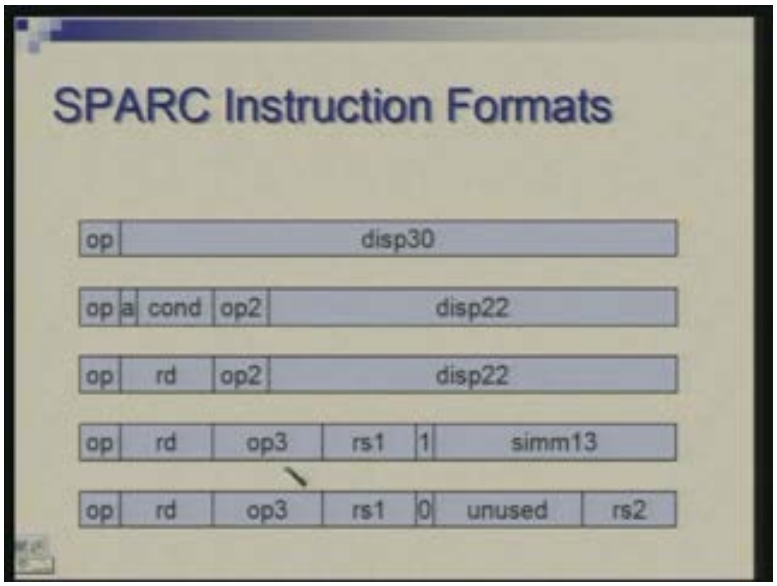
with a negative value and you could increment till it becomes zero. You are testing it and if it is not zero you go to beginning of the loop. In power PC there could be a single instruction which is branch and count to loop and the condition being checked is whether the count is not equal to 0. So, again lot of conditions could be specified here but this instruction in particular is saying that branching is to be done when counter is not 0.

(Refer Slide Time: 18:46)



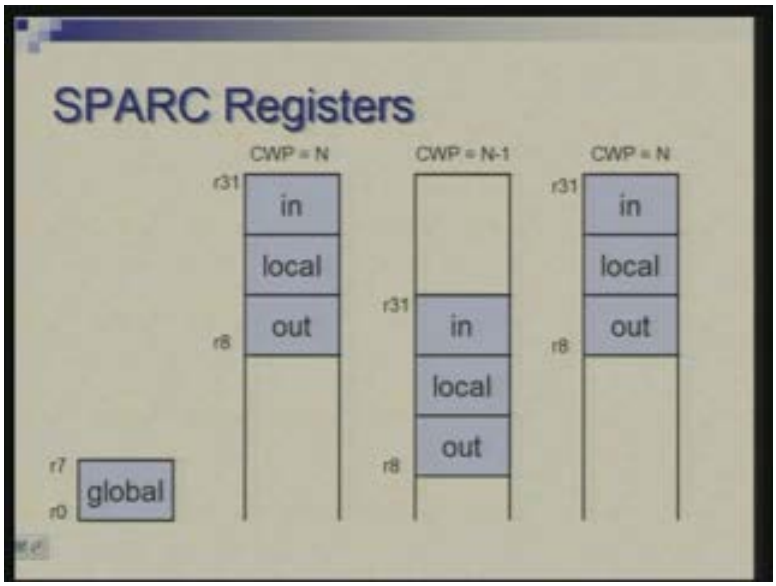
The next example of RISC I am taking is SPARC architecture. The term SPARC stands for scalable processor architecture and scalability here means that the same instruction set could be implemented in different ways with different technologies and the architecture would scale. So with same instruction set you could get different performances, also, scale to a higher word size. In fact this architecture also has been scaled to 64 bits and this is an open architecture which is been which is implemented by several companies; Sun was the main one, then Texas instruments, Toshiba, Fujitsu, Cypress, Tatung etc. This has again a limited number of instruction formats but certainly more than what we have seen in MIPS.

(Refer Slide Time: 19:38)



There is a 2-bit opcode which is a which is somewhat peculiar and clearly this would need extensions because you can only specify four different patterns here. There are opcode extension fields here as you see op2 here in this case and op3 in this case and in this case. So this is a 3-bit extension field and this I think is probably 5 or 6-bit extension field. rd rs1 rs2 these are again register fields and size is 5 bits. You would see that constants are of various sizes 13-bit, 12-bit in these two cases and 30-bit here so this would be an unconditional jump instruction and you could see that you get full-fledged 30-bit address here which means you can address…… if you take this as a word address you have complete addressability, you do not have to take bits from PC so in 30 bits you can address entire memory space. That is the reason why opcode field has been squeezed to two bits and then exceptions are made and extensions are made. This is for conditional branch. This is for also for a branch where there is a displacement of 22-bit with reference to PC. Here you have format for immediate instructions you are limited to 13-bit constant. If the operand…….. if the constant is larger then you have to load it separately in a register and then work. This is a typical three address arithmetic type of instruction.

(Refer Slide Time: 22:09)



The most interesting feature which was brought out by SPARC architecture was the concept of register windows. what they have is there is a large number of registers which could be as much as 128 to 256 or even more; in different implementations actually the size differs but a program sees only 32 registers at a time so to say it has a window of 32 registers at any time which can move up or down as you go from one context to other context.

We have seen that in MIPS when a procedure call is made you have to worry a lot about what register is used by caller, what register is used by callee so somehow you have to share the registers. If you have a larger requirement then you need to do lot of transfers to memory, save them and restore them. So it is this problem which is addressed by register windows that when you make a call to a procedure or function you get a fresh set of registers so that the caller and the callee can work independently on their own set of registers. But this is another extreme; you also want some linkages some flow of data between caller and callee in terms of parameters. So therefore the two windows are not totally disjoint they are overlapping and overlap is in a very regular fashion.

For example, suppose you are in the main program you will see total of 32 registers and these are divided into four parts: input, local, output, global. So as the windows change the global registers eight of them remain unchanged so you see same eight registers which are let us say out of array of 256 registers and the first eight are always part of the window and local are those which are totally exclusive to a given context whereas in and out are overlapping. So when, for example, from this context you make a call to another function then the set of eight registers which was out here become in for that. Essentially this is a way you pass parameters.

Whatever information up to eight values which you want to pass on you can load into these eight registers and then when you make a call the window slides, you get these 24

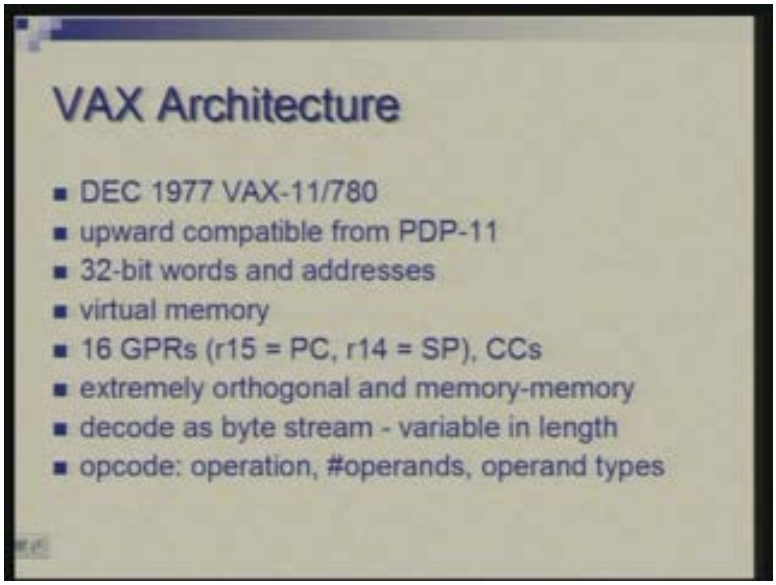
registers, the global remains the same and what was out has not become in because these are the registers which have carried information into it and what appears as out will become in for the next level. So out of these 24 registers 8 are used for getting information….. 8 are purely local, 8 are used for getting information out to the next level. Of course this is the convention. Once you have got some values in registers you can use them for local usage also. So at any given time you have these 24 or some other 24 registers and 8 global registers.

[Conversation between student and Professor……25:35 [what does CWP stands for] yeah I am coming to that.]

Now, to keep track of current window you need a current window pointer or CWP. Suppose CWP or the window pointer is pointing to this window next it will point to that window, when return occurs it gets back to the same thing. so suppose you have 128 or 256 whatever be the number there is a finite set of windows which are possible and if depth of calls become more than that then actually windows act like, I mean, this set of registers acts like a cyclic buffer. So you can keep on going deeper and the window keeps on sliding but of course as it loops back the old window the first one will have to be saved in memory otherwise it will be overwritten. So that saving and restoring has to be done but when you go to a depth of let us say 8 or so then you will have to worry about saving or restoring.

For most practical purposes the depth would typically not be more than 4, 5 or 6 and often these windows are fives so it is only in rare cases that you need to spend time and effort in saving and restoring the windows. Otherwise call to a procedure is speeded up by this facility.

(Refer Slide Time: 27:11)



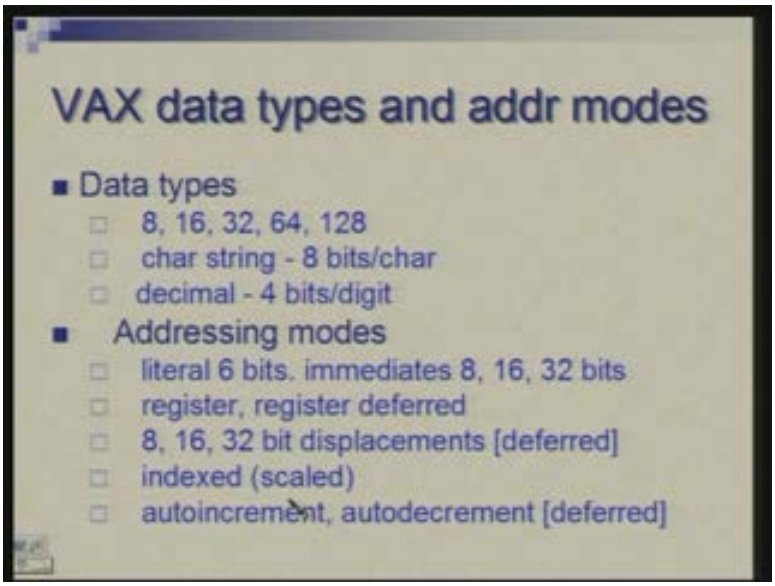
Next we take example of VAX architecture which was developed by DEC or Digital Equipment Corporation in 1977. This was a successor of PDP-11 a minicomputer which was let us see the most successful minicomputer and all the PDP-11 was 16-bit machine this was VAX which was a 32-bit machine. So this of course has been now discontinued, DEC started with architecture called alpha which is a RISC architecture. Of course the company DEC is no longer…….. it first was taken over by Compaq which was a predominant PC manufacturer, system manufacturer and then Compaq itself was taken over by Hewlett Packard so all that is history.

VAX brought in the concept of virtual memory which actually gives you illusion of a larger memory than what you have physically. We will discuss this topic in detail later on. it has 16 general purpose registers unlike 32 in MIPS or SPARC or power PC which we have seen this is only 16 and out of these 16 one of them is PC; program counter is out of these it is not a separate register, also stack pointer is one of these beyond this you have besides this you have condition codes similar to power PC.

Now the key feature here is that there is extreme orthogonality. That means the way you specify operands is independent of largely independent of the opcode. So you have different ways of specifying operands or different addressing modes and they can be independently combined with any of the instruction opcode. So, all instructions support all addressing modes whereas in MIPS we have seen that for each instruction there is a specific addressing mode. Here you have whole lot of instructions, whole lot of addressing modes and all combinations are possible.

What we will see next is Intel machine. There are lots of instructions, lots of addressing modes but only some combinations are valid some are not valid and it is a bit hard to remember which are and which are not. So the instructions are taken only as byte stream because as I mentioned the size could be varying quite a lot: 1 byte to something like 54 bytes so decoding is not all that easy. The opcode specifies many things. Opcode is typically a byte but sometimes it actually extends to another byte. So not only it specifies which operation is to be done but how many operands are there. So for example add would be available with the two operands, two operands, three operands, six operands different modes are there so number of operands and operand types whether it is working on integers, reals, double precision, bytes and so on so all that information is packed in either 1 byte or 2 bytes.

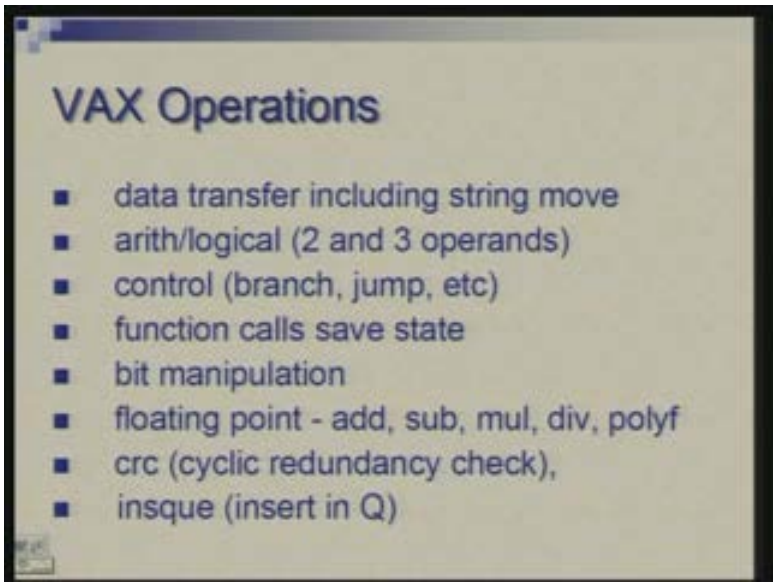
(Refer Slide Time: 27:11)



The data types varying from 8 to 128, it could deal with character strings where 1 byte represent a character or string of decimal digits one digit coded in 4 bits; there are lots of addressing modes, there are 6-bit constants, 8-bit constants, 16-bit constants, 32-bit constants, they are register addressing mode that means register carries the operand or you would find the term deferred with many of the addressing modes. Deferred refers to some kind of indirection. That means what you are getting is only an address and then you take that address and make access to memory to get the operand. So register mode means that register carries the operand; register deferred means register carries the address and you carry on one memory access cycle.

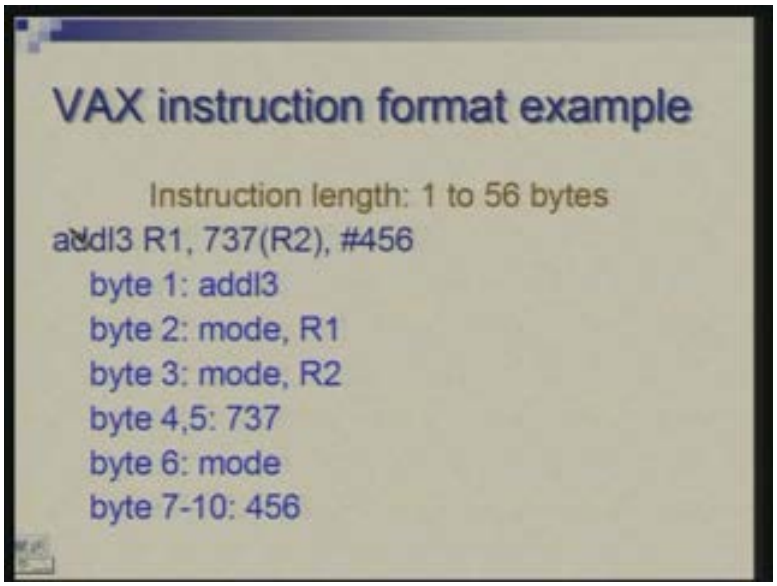
Displacement: Displacement with respect to registers and one of the register is as you know is PC, one is SP. PC relative and register relative are not two different modes, you could have 8, 16, 32 bit displacement with respect to any of the registers and its deferred version. That means register plus a constant can give you an operand it can also give you an address. So indexing is there, indexing with scaling that means the index which is getting added can be multiplied by 1 2 or 4 depending upon operand size; autoincrement autodecrement with deferred so all these addressing modes are available; variety of data transfer variety of operations I will not go through all of them some are common.

(Refer Slide Time: 27:11)



You could see peculiar things like operations which work on polynomials directly or operations which do insertion in queue. So here is an example of instruction. For example, addl3; 3 refers to three operands here. Well, when I say operand it actually means we are also counting the destination and this l stands for the datatype so l implies 32-bit long integer, integer here means 16 actually so 32-bit they call as long integer and the three addresses are R1which is a register addressing mode, this is a….. you could say a base addressing and this is immediate addressing so each of these could appear in any mode.

(Refer Slide Time: 33:28)

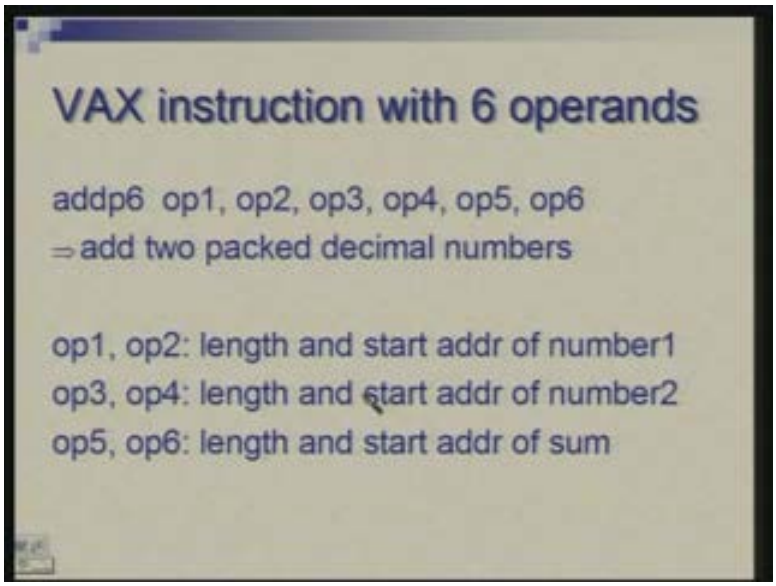


You are specifying two operands and a result; each one can be in any mode and there is a whole variety of addressing mode which is……. of course destination cannot be in immediate mode so that thing naturally do not make sense they are of course excluded. So the way instruction is represented is 1 byte will specify this opcode and it is operation, the datatype and number of addresses so those three information or three pieces are there. Byte two specifies the mode of one of the addresses and register R1. So actually four bits specify a mode and you can see that there are sixteen modes and four bits for specifying register number. So since they are only sixteen registers similarly for second one you have byte three which specifies the mode and say the register is R2.

The mode which has been specified is that it says base addressing mode. That means the constant needs to be specified. So next two byte carry the constant that is how that is one reason why instruction size varies. Depending upon modes and number of operands the instruction size varies. So, if all the modes were registers suppose it was addl3 R1 R2 R3 then the same instruction would occupy less number of bytes.

But here you have specified a base addressing mode so two additional bytes come in you specified a constant immediate mode for this. So four bytes come for that. Byte 7, 8, 9 and 10 carry this constant. So this will give you some glimpse of how instructions vary in size because of all this variety which is possible.

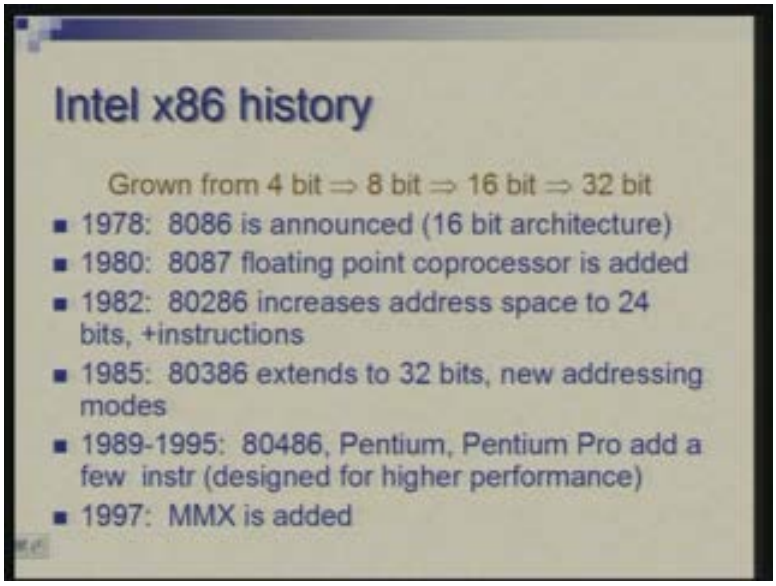
(Refer Slide Time: 35:00)



There is another example of instruction addp6, it has six operands and why do we need six operands it is actually adding two decimal numbers each decimal number is represented as a string of digits. Packed digits means that you are packing it in the smallest number of bits so it is four bits for every digit, unpacked would mean that you pack a digit in 1 byte or one word. So the two sources and one destination each require two operand specifications. So, op1 op2 are for first number, 3 and 4 for second, 5 and 6 for the result. And why you need two for each is one gives the length of the string and

the other gives the starting address so you could add long you could take a long numbers and not necessarily can find two 32 bits or 64 bits there could be long strings of digits and single instruction could add these.

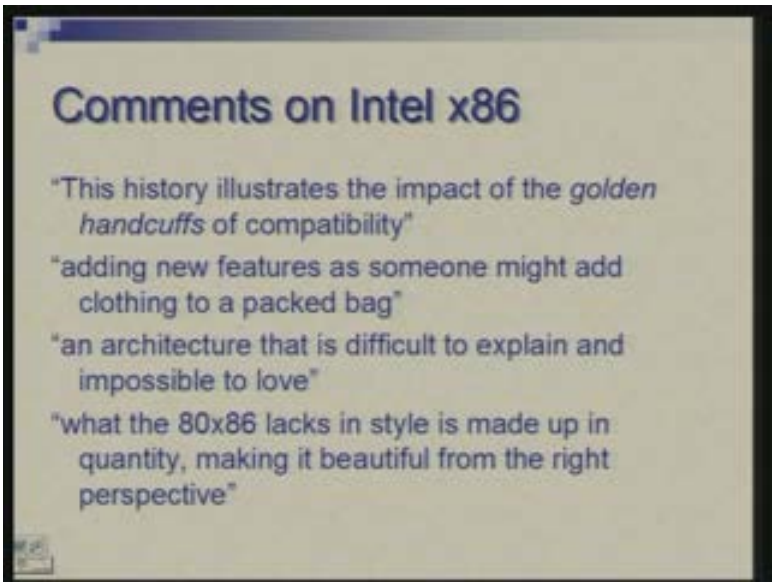
(Refer Slide Time: 37:28)



Finally let us talk of Intel x86 which has a very very long history beginning with 4-bit, 8-bit, 16-bit, 32-bit. Of course there were major breaks 4 to 8 and 8 to 16 but starting with 16-bit processor 86 which was introduced in 78 there has been a reasonable amount of compatibility which has been maintained. So x86 was announced in 78 which was essentially 16-bit architecture then few years later its companion floating point processor coprocessor was added, in 1982 80286 was introduced which increased the address space to 24 bits, earlier it was 16 bits.

With 16 bits you have addressability of only 64 kilobytes, as the demand increased the space had to be increased. With 24 bits you get sixteen mega bytes and also some instructions are added. With 80386 this series has got 32-bit architecture and more newer addressing modes were defined and also paging in terms of virtual memory was introduced here. Then, after that we have 80486 Pentium, Pentium pro, Pentium 2, 3, 4 so the whole series and the emphasis has been on adding more instructions; all the compatibility is maintained but more instructions have been added for higher performance. MMX, for example, multimedia extension then there are SIMD instructions for again performing operations on arrays so although the additions have taken place but the core instruction set has remained. So let us have a look at that part but before that I would like you to read some interesting comments. These are from our textbook.

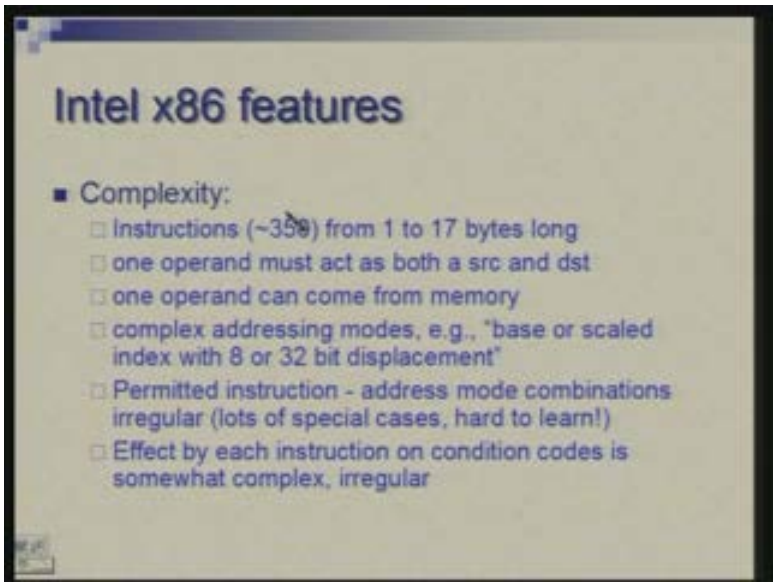
(Refer Slide Time: 39:48)



The history illustrates the impact of the golden handcuffs of compatibility. What it means is that to maintain compatibility the architecture has got tied up and you are sort of constrained; you can keep on adding features but you need to maintain what is there so that baggage has to be carried and it is like adding new features is something like adding clothing to a packed bag. It is an architecture that is difficult to explain and impossible to love. It is a very complex architecture; it is very difficult for a programmer to be sure that, yes, out of these instructions I am using the right ones; I am using them in the best possible manner. Even for a compiler it may be very difficult to make best use of all possible instructions and all possible variations which are there with the instruction.

But on the other hand, what this lacks in style is made up in quantity. So, because of compatibility it has been a commercial success which means it is widely used and available in large quantities.

(Refer Slide Time: 41:14)

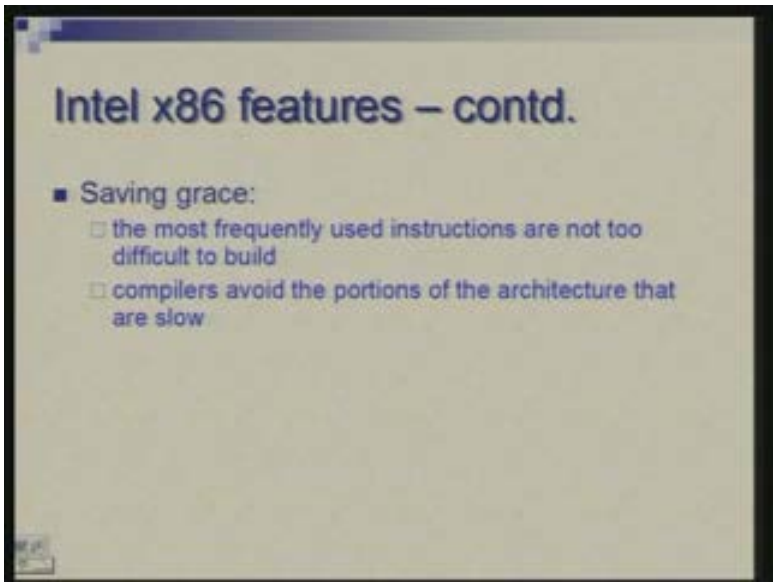


So let us look at some of the features; it has large number of instructions, about 350 and they also vary in size like VAX but not as much; from 1 to 17 which is still enough to give you headache when you are designing hardware for it. Now this is primarily a two address machine whereas VAX can be three address two address so a whole variety is there this is primarily a two address machine so one of the sources also doubles up as a destination.

Out of two operands one can come from memory and one has to be a register so both could be a register but one can come from register so it has RM type of architecture. You remember, I talked about RR, RM, MM; MM is one when both operands come from memory so VAX is a combination of all, VAX supports all of them and often one would call R plus M but this is RM type of architecture.

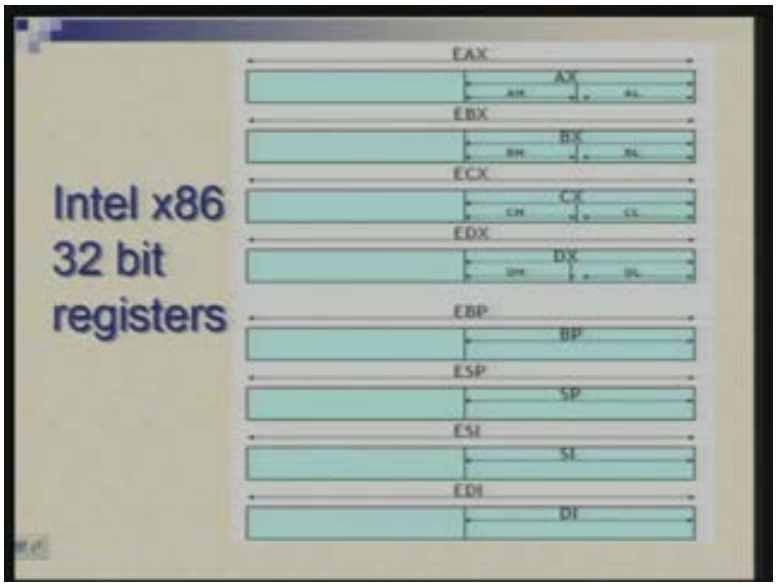
There are complex addressing modes almost a similar set as you have in VAX. There is a base addressing mode or scaled index with the displacement can be 8-bit or 32 bit and index can be scaled with 1, 2 or 4 depending upon whether you access in series of bytes or series of words. Now the difficulty is that the permitted combination of instruction and addressing modes are very irregular so the two extremes of MIPS is that there is a one-to-one sort of one-to-one correspondence; VAX says that you can use with anything and anything but here you have very peculiar combinations so there are lots of exceptions which are difficult to remember so lots of special cases which are hard to learn and the effect of each instruction on condition codes is again somewhat complex and irregular.

(Refer Slide Time 43:21)



On the other hand, there is a saving grace that the most frequently used instructions are not too difficult to build and compilers have learned to avoid the portions of architecture that is not very efficient so the compilers also focus on a small subset of all the instructions and those instructions can be built efficiently.

(Refer Slide Time: 43:48)



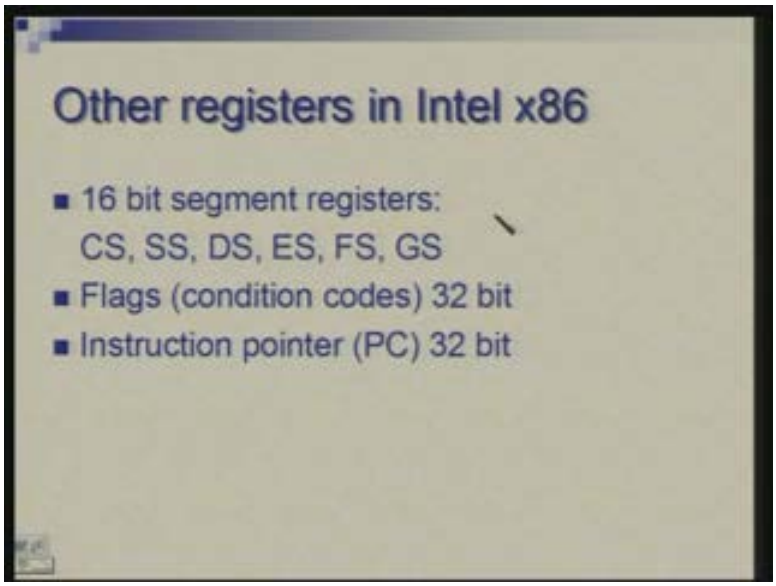
So here is a glimpse of register structure not all registers but I am showing all the major 32-bit registers of 8086. So they have a name like EAX, EBX, ECX, EDX this is one set where E and X are both there. Actually E stands for extension extended A register so actually extension has taken place in two stages so there were 8-bit architectures with

registers called A B C D and so on and then migrated to 16-bit architecture where you had AH AL BH BL where H stands for high and L stands for low and then they were extended further to get 32-bit registers. So still old names can be used so although EAX, EBX, ECX, EDX when you write in assembly they refer to 32-bit registers. You can also use AX BX CX DX that means you can work with half of the registers that means take data from half of it put the result in half of it that is possible and you can also work with quarters and refer to them as AH AL BH BL and so on.

Then there is another set of registers which in earlier versions are called BP SP SI DI or here called as EBP ESP EDI where E again stands for extended. BP is base pointer, SP is Stack Pointer, SI is Source Index DI is Destination Index. So, base pointer, stack pointer have their usual meaning; SI and DI are referred to operations which work on string so there is a source string and destination string and SI and DI would typically be used for indexing individual element in those strings.

The other registers particularly they are 16-bit segment registers which are labeled as CS SS DS and others. CS stands for code segment, stack segment, data segment and so on. So D E F G is all actually for data.

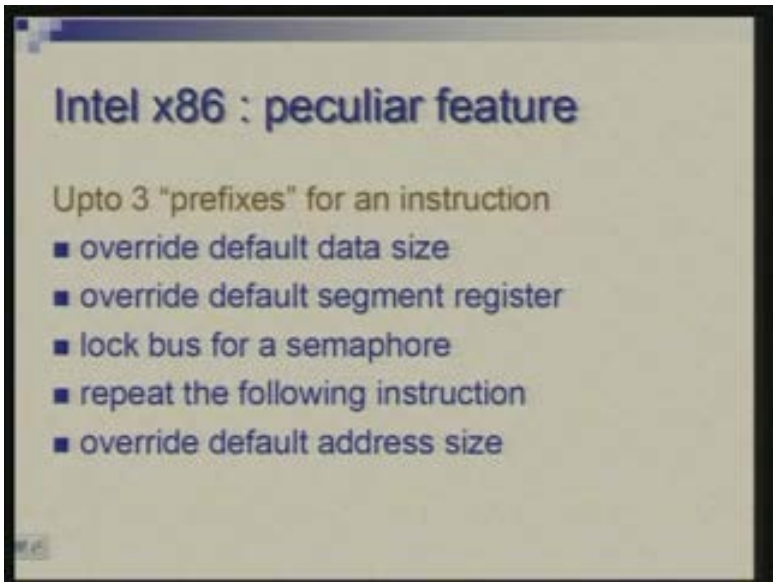
(Refer Slide Time: 46:04)



Therefore, segmentation was a technique which is brought into earlier versions of this architecture to address larger memory than what can be done with a 16-bit address register. So 16-bit register is limited to 64 k bytes but how do you access larger memory is with the help of a segment register. In a segment register you set up a base and starting from that base you can access 64 register sorry 64 k bytes of memory. You want to address another area you can move your base and then address 64 k bytes there. So although these segment registers are also 16 bits but they are used with some multiplication factor.

So, for example, to get 20-bit 24-bit addressing you imagine that segment register is shifted by 8 bits so you take 16-bit value with eight zeros and then add another address register that is how you will get a full-fledged 24-bit address. Of course they are flags and program counters which is called instruction pointer here.

(Refer Slide Time: 48:01)

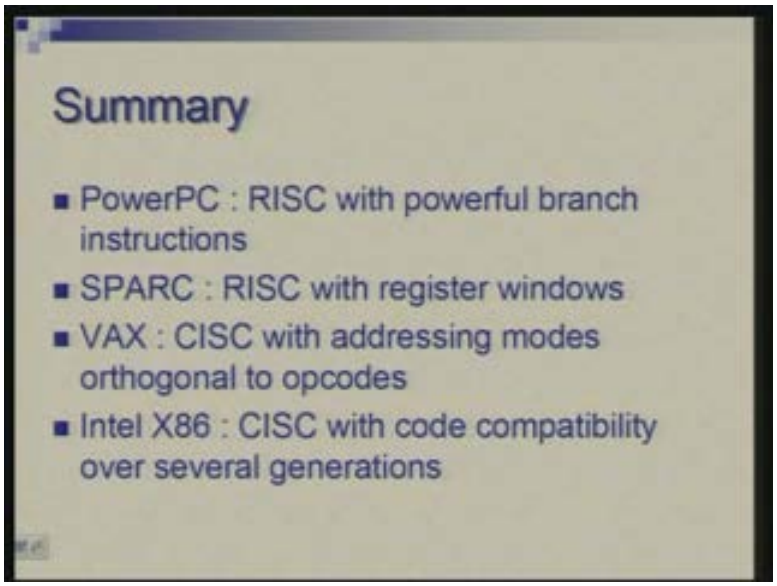


Lastly I will mention another peculiar feature of this architecture which is the concept of prefix. That means you have an instruction you can put a byte before the instruction. In the sequence of bytes in your code an instruction could be preceded by a byte which is called a prefix which in some sense modifies the meaning of the instruction and there can be up to three prefixes for an instruction. The kind of modifications these prefixes do are shown here.

For example, instructions work with some default data size unlike VAX case where the data size was carried as part of the instruction. For example, addl3, l was referred to the data size but here default size is set somewhere but these prefixes can override that. There is always a default segment register. You might, for example, put ES as a default segment register in some part of the code but for some instruction if you want to override this could do so. They could also help in implementing some special instruction like semaphore; they require exclusive as to bus and bus has to be locked. Prefixes can be used to repeat the following instruction certain number of times. The default address size can also be overridden. So in these ways and some others these prefixes can be used to modify effective instruction and effectively it gives an impression of in fact a large number of instructions with all these prefixes.

Finally let me conclude by summarizing one or two important features of each of the four architectures we have looked at.

(Refer Slide Time: 49:30)



Beginning with power PC we found that it is a RISC architecture but with very powerful set of branch instructions. SPARC is another RISC and its very significant feature is the concept of register windows. VAX, we took as an example of CISC machine which has wide variety of addressing modes and the strength of this is that all these addressing modes are orthogonal to opcodes. Finally we had some talk of x86 of Intel; it is a CISC where the strongest feature the code compatibility and that is the reason for its commercial success. I will stop with this.