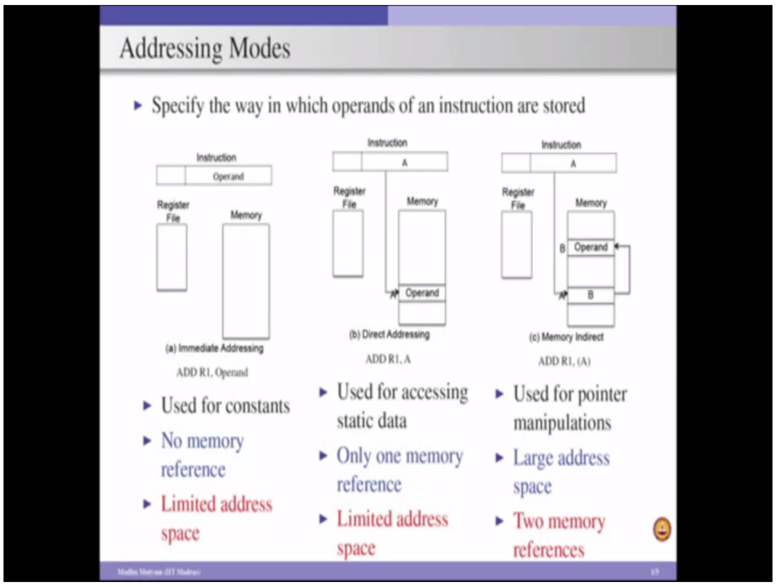
Computer Architecture

Prof. Madhu Mutyam
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 04
Instruction Set Principles (Part 2)

So, as part of the last module we discussed the role of instruction set architecture, the ISA classification and memory addressing. So, in this module we are going to discuss the addressing modes. So, why do we need addressing modes?

(Refer Slide Time: 00:28)



Addressing Modes

- Specify the way in which operands of an instruction are stored

- Used for constants
- No memory reference
- Limited address space

- Used for accessing static data
- Only one memory reference
- Limited address space

- Used for pointer manipulations
- Large address space
- Two memory references

So, a typical instruction consists of an opcode and a couple of operands. And opcode will specify the type of operation the instruction's going to perform and the operands are going to specify the data item value or the address of the data item that they are referring. Typically an instruction consists of few bits and so as a result operand cannot, because the number of bits allocated in the operand field is very limited.

So, as a result it is not possible to reference large address space. And sometimes the operands will directly provide the constraint values and some other cases the operands can refer an address of a register or a memory location. So, effectively the ISAs make use of these addressing modes to specify the way in which operands of an instruction are stored. So, in this module we are going to discuss six addressing modes.

Of course, there are several addressing modes are there in several ISAs and so on. So, we start with immediate addressing. So, I am going to explain these addressing modes with a pictorial representation. Consider an instruction which consists of two parts. One is the opcode field the other one R1 is an operand field. And in the case of immediate addressing an operand field actually contains the actual value the instruction is referring to.

So, a typical example of an immediate addressing is add R1, operand, where the operand is a constant value the address the add instruction is using. So, this addressing mode is typically used for accessing constants. And the main advantage with this addressing mode is there is no need of a memory reference, because the value required by the instruction is directly available in the address field or operand field. The disadvantage with this is the limited address space.

The reason is, as we know a typical instruction can consist of few bits and in that some bits will be already allocated for the opcode. So, as a result the number of bits allocated per operand field is very limited so, using that we can reference only a limited address space. The second addressing mode we considered is a direct addressing. So, difference between the direct addressing mode and the immediate addressing mode is, in the immediate addressing mode operand field contains the value, but where as in the case of direct addressing mode, the operand field contains the address of a memory location, where the actual operand value is stored.

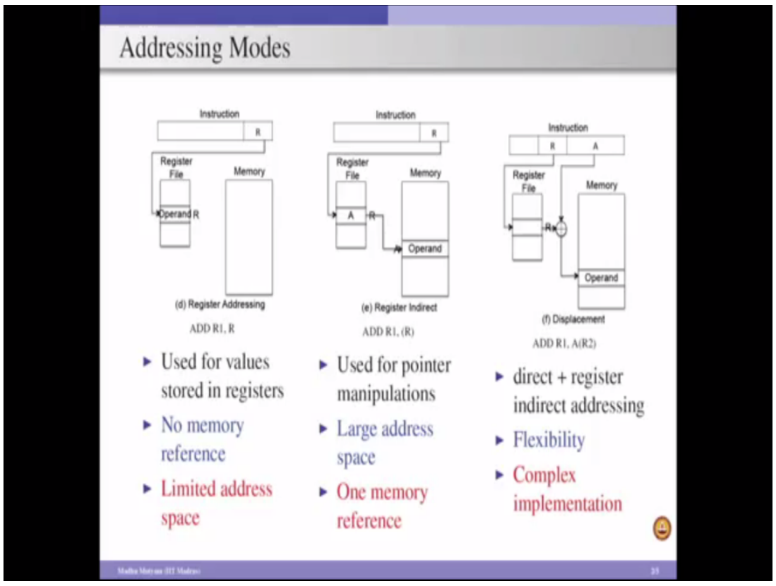
So, effectively once we decode the instruction, we get this address from the operand field and we use that address and access the memory to get the actual operand value. A typical example of this direct addressing mode is 'add R1, A', where 'a' specifies the address of a memory location. So, typically this addressing mode is used for accessing static data because typically static data is stored in the memory rather than the registers.

So, it requires only one memory reference compared to the other addressing modes that we discuss in a short while. The disadvantage with this addressing mode is still the limited address space. The number of addresses we refer using the operand field is limited because our operand field width is very limited here. So, similar to the immediate addressing mode it has a disadvantage of having limited address space. So, inorder to increase the address space we can use the next addressing mode which is called as memory indirect. So, here our operand field specifies the address of a location, once we go to that location we actually get the address of the actual operand.

So, it is like an indirection, from the operand field value we go to a memory location, this memory location in turn contains the address of the actual operand, we go to that location and get the actual value. So, here the typical example of this addressing mode is add R1, (A) . This (A) indicates the indirection. So, typically this addressing mode is used for pointer manipulations and the main advantage compared to the previous two addressing modes is the large address space, but the main disadvantage with this memory indirect addressing mode is, it requires two memory accesses.

First memory access is to get the actual address of the operand and the second memory access is to get the actual operand. So, similar to this memory addressing modes, because as I said previously operands can be specified as address in a memory location or address in a register file. So, we come up with two other addressing modes which use the address in a register file rather than in a memory.

(Refer Slide Time: 06:29)



Addressing Modes

ADD R1, R

- Used for values stored in registers
- No memory reference
- Limited address space

ADD R1, (R)

- Used for pointer manipulations
- Large address space
- One memory reference

ADD R1, A(R2)

- direct + register indirect addressing
- Flexibility
- Complex implementation

So, the first one is register addressing, this is analogous to direct addressing mode, rather than accessing the memory we access an address location in the register file specified by the operand field. A typical example of this addressing mode is ‘add R1, R’ where R specifies the address of a register in the register file. It is typically used for accessing values stored in registers and we know that register access is much faster compared to memory access and so on.

So, a register addressing mode can be used for performance of the system. And the main advantage with this register addressing mode is, it will not require any memory reference because of values are stored in the registers and we always access the register files to get these values, but the disadvantage is again because the operand field width is a limited. So, as a result we cannot access large memory address space. The next addressing mode is register indirect where the operand field specifies the address of a register in the register file and we go to that register file access that particular address location.

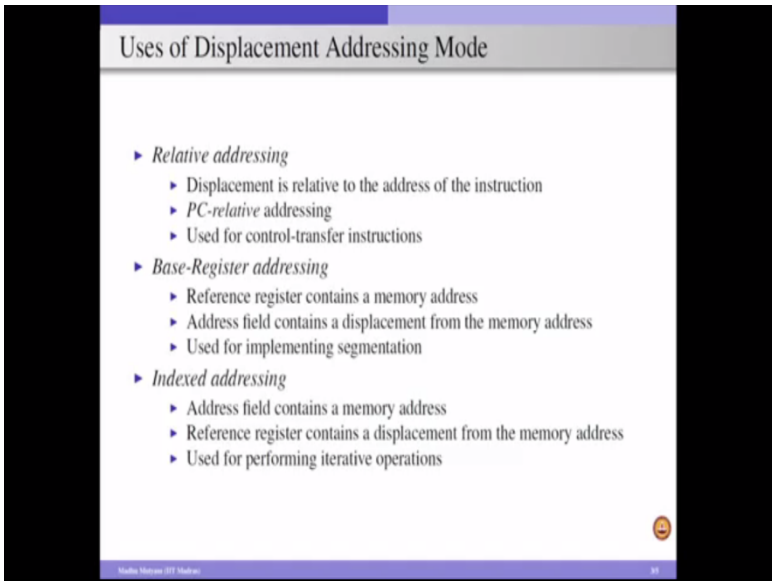
And that in turn contains the address of a memory location in memory. So, go to the memory and access that location and get the actual operand value. Typical example here, is add R1, (R) again as I mentioned earlier (R) indicates the indirection. And this type of addressing mode is typically used for pointer manipulation similar to memory indirect addressing mode and because we finally, access the memory to get the actual operand value.

So, we can provide large address space, but the compared to register addressing mode the register indirect addressing mode has a drawback, which is it incurs one memory access. Remember memory accesses are time consuming compared to register accesses. And finally, we consider very important addressing mode which is called as displacement addressing mode. In this displacement addressing mode we have an operand field which specifies the offset or the displacement.

And we may have another operand field which specifies the register address, but the second operand is optional here. So, we may specify the register explicitly in the instruction or this register can be represented implicitly. A typical example of this addressing mode is add R1, (R). So, here A specifies the displacement from the address specified by the contents of register R. And this addressing mode actually combines the benefits of both direct addressing mode and the register indirect addressing mode.

And it provides more flexibility in using this addressing mode in various scenarios, but the drawback with this addressing mode is it is complex to implement, compared to the other addressing modes. Now we are going to look at the uses of displacement addressing mode.

(Refer Slide Time: 10:08)



Uses of Displacement Addressing Mode

- *Relative addressing*
  - Displacement is relative to the address of the instruction
  - *PC-relative* addressing
  - Used for control-transfer instructions
- *Base-Register addressing*
  - Reference register contains a memory address
  - Address field contains a displacement from the memory address
  - Used for implementing segmentation
- *Indexed addressing*
  - Address field contains a memory address
  - Reference register contains a displacement from the memory address
  - Used for performing iterative operations

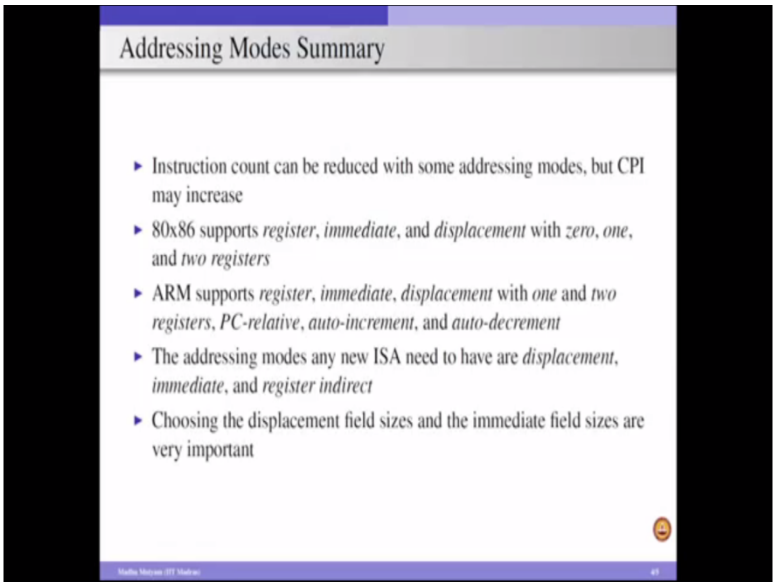
As I mentioned the displacement addressing mode actually specifies the displacement or an offset from a particular address. This address can be specified explicitly by a register operand or by an implicit register location. So, we can use this in relative addressing. So, what is relative addressing? So, relative addressing specifies that the displacement is relative to the address of the current instruction. We know that program counter typically stores the address of the next instruction to be executed. So, whenever we have a jump instruction or control transfer instructions, we know the next instruction address.

And all we have to give is, we specify the displacement with respect to the next instruction address in the operand field of the displacement addressing mode so that, we can directly go to the specified target address. So, as a result this relative addressing mode is also called as ‘PC Relative’ and this is mainly used in control transfer instructions. This displacement addressing mode can also be used in base register addressing. So, here again we can specify the base address in a specific register and the displacement can be specified in the operand field. So, that we displace the next instruction can be executed at an address which is a displacement distance from the contents in the base register.

So, the reference register contains a memory location. Typically we use the segment registers. So, which contains the address of a segment and the displacement can be provided in the operand field. So, that this base register addressing is typically used to implement segmentation in the systems. And finally, this displacement addressing mode can also be used for indexed addressing. This indexed addressing actually performs a reverse to the base register addressing.

So, here our address field contains a memory address and the reference register as I mentioned earlier, it can be explicitly mentioned or an implicitly mentioned which contains a displacement from the memory address. So, typically this type of addressing mode is used in performing operations on arrays or iterative computations.

(Refer Slide Time: 13:04)



Addressing Modes Summary

- Instruction count can be reduced with some addressing modes, but CPI may increase
- 80x86 supports *register*, *immediate*, and *displacement* with *zero*, *one*, and *two registers*
- ARM supports *register*, *immediate*, *displacement* with *one* and *two registers*, *PC-relative*, *auto-increment*, and *auto-decrement*
- The addressing modes any new ISA need to have are *displacement*, *immediate*, and *register indirect*
- Choosing the displacement field sizes and the immediate field sizes are very important

So, in summary, there are several addressing modes proposed in several ISAs and these addressing modes will provide design trade offs. Some addressing modes will reduce the instruction count that will be good for reducing our overall CPU time as we discussed in one of the previous modules, but when this addressing modes reducing the instruction count they may incur the high CPI clocks per instruction. So, as a result we need to be careful in selecting our addressing modes such that it should not complicate the overall design, it should not increase the CPI, average CPI of the system. And also it should not increase the overall computation time.

And 80x86 ISA supports the register addressing mode, immediate addressing mode and displacement addressing mode with variations which include zero registers, one register and two registers. When I say displacement addressing mode with zero registers. So, there is no the reference register our displacement is absolute value. And in the case of displacement with one register, we can implement our relative addressing mode as we discussed earlier or we can implement the base register addressing mode.

So, and when we consider displacement addressing with the two registers. Typically this is used for implementing multidimensional arrays and so on where we can use base index displacement addressing mode or base scaled index displacement addressing modes. And in the case of ARM ISA, we have support for register addressing mode, immediate addressing mode, displacement with one and two registers, it also supports a pc relative addressing mode and in addition to that it supports auto-increment and auto-decrement. So, this auto-decrement and auto-increment addressing modes we have not discussed earlier, but actually this auto-increment and decrement.

Typically, it does a specific thing when we access a memory location, the automatically the address of the location either incremented or decremented respectively. So, typically when we want to design a new ISA, we need to ensure that at least these three addressing modes are supported. Those are displacement, immediate and register indirect addressing modes and also because when we are dealing with displacement addressing modes. So, the displacement field or the operand field size, we have to be careful about that. If we keep too much displacement field width.

So, we may not support many operations because the number of bits in an instruction is a limited. And similarly, when we are implementing the immediate addressing mode we have to ensure that the bit field for this immediate value also should be considered reasonable value. So, otherwise again it has an impact on the overall instruction length.

So, in summary, choose the best addressing modes required for your ISA depending on the target applications you are looking for the computer. And when we are considering the displacement addressing mode and immediate addressing modes, choose reasonable size operand bit fields for these two addressing modes.

Thank you.