[Document subtitle]

Instruction Set Architecture (ISA)

The Instruction Set Architecture (ISA) is the part of the processor that is visible to the programmer or compiler writer. The ISA serves as the boundary between software and hardware.

Instruction Set Architecture (ISA)

The *Instruction Set Architecture* (ISA) is the part of the processor that is visible to the programmer or compiler writer. The **ISA** serves as the boundary between software and hardware.

An **instruction set**, or **instruction set architecture** (**ISA**), is the part of the computer architecture related to programming, including the native data types, instructions, registers, addressing modes, memory architecture, interrupt and exception handling, and external I/O. An ISA includes a specification of the set of opcodes (machine language), and the native commands implemented by a particular processor.

ISA is the parts of a processor's design that need to be understood in order to write assembly language, such as the machine language instructions and registers. Parts of the architecture that are left to the implementation, such as number of superscalar functional units, cache size and cycle speed, are not part of the ISA.

Instruction set architecture is distinguished from the microarchitecture, which is the set of processor design techniques used to implement the instruction set. Computers with different microarchitectures can share a common instruction set. For example, the Intel Pentium and the AMD Athlon implement nearly identical versions of the x86 instruction set, but have radically different internal designs.

We will briefly describe the instruction sets found in many of the microprocessors used today. The ISA of a processor can be described using 5 categories:

Operand Storage in the CPU: Where are the operands kept other than in memory?
Number of explicit named operands: How many operands are named in a typical instruction?
Operand location: Can any ALU instruction operand be located in memory? Or must all operands be kept internaly in the CPU?
Operations: What operations are provided in the ISA.?
Type and size of operands: What is the type and size of each operand and how is it specified?

Of all the above the most distinguishing factor is the first.
The 3 most common types of ISAs are:

1. ***Stack*** **-** The operands are implicitly on top of the stack.
2. ***Accumulator*** **-** One operand is implicitly the accumulator.
3. ***General Purpose Register (GPR)*** - All operands are explicitly mentioned, they are either registers or memory locations.

Classification of instruction sets
A complex instruction set computer (**CISC**) has many specialized instructions, which may only be rarely used in practical programs. A reduced instruction set computer (**RISC**) simplifies the processor by only implementing instructions that are frequently used in programs.

Number of operands
Instruction sets may be categorized by the maximum number of operands *explicitly* specified in instructions.

Complex instructions

Some computers include "complex" instructions in their instruction set. A single "complex" instruction does something that may take many instructions on other computers. Such instructions are typified by instructions that take multiple steps, control multiple functional units, or otherwise appear on a larger scale than the bulk of simple instructions implemented by the given processor. Some examples of "complex" instructions include:

- saving many registers on the stack at once
- moving large blocks of memory
- complex and/or floating-point arithmetic (sine, cosine, square root, etc.)
- performing an atomic test-and-set instruction
- instructions that combine ALU with an operand from memory rather than a register

A complex instruction type that has become particularly popular recently is the SIMD or Single-Instruction Stream Multiple-Data Stream operation or vector instruction, an operation that performs the same arithmetic operation on multiple pieces of data at the same time. SIMD have the ability of manipulating large vectors and matrices in minimal time. SIMD instructions allow easy parallelization of algorithms commonly involved in sound, image, and video processing. Various SIMD implementations have been brought to market under trade names such as MMX, 3DNow! and AltiVec.

Instruction length

The size or length of an instruction varies widely, from as little as four bits in some microcontrollers to many hundreds of bits in some VLIW systems. Processors used in personal computers, mainframes, and supercomputers have instruction sizes between 8 and 64 bits. The longest possible instruction on x86 is 15 bytes (120 bits). Within an instruction set, different instructions may have different lengths. In some architectures, notably most reduced instruction set computers (RISC), instructions are a fixed length, typically corresponding with that architecture's word size. In other architectures, instructions have variable length, typically integral multiples of a byte or a halfword.

Representation

The instructions constituting a program are rarely specified using their internal, numeric form; they may be specified by programmers using an assembly language or, more commonly, may be generated by compilers.

Design

The design of instruction sets is a complex issue. There were two stages in history for the microprocessor. The first was the CISC (Complex Instruction Set Computer) which had many different instructions (see lecture 11). In the 1970s, however, places like IBM did research and found that many instructions in the set could be eliminated. The result was the RISC (Reduced Instruction Set Computer), an architecture which uses a smaller set of instructions (see lecture 11). A simpler instruction set may offer the potential for higher speeds, reduced processor size, and reduced power consumption. However, a more complex set may optimize common operations, improve memory/cache efficiency, or simplify programming.

Reduced Instruction Set Computer (RISC)

As we mentioned before most modern CPUs are of the GPR (General Purpose Register) type. A few examples of such CPUs are the IBM 360, DEC VAX, Intel 80x86 and Motorola 68xxx. But while these CPUS were clearly better than previous stack and accumulator based CPUs they were still lacking in several areas:

1. Instructions were of varying length from 1 byte to 6-8 bytes. This causes problems with the pre-fetching and pipelining of instructions.

2. ALU (Arithmetic Logical Unit) instructions could have operands that were memory locations. Because the number of cycles it takes to access memory varies so does the whole instruction. This isn't good for compiler writers, pipelining and multiple issues.
3. Most ALU instruction had only 2 operands where one of the operands is also the destination. This means this operand is destroyed during the operation or it must be saved before somewhere.

Thus in the early 80's the idea of RISC was introduced. The SPARC project was started at Berkeley and the MIPS project at Stanford. RISC stands for Reduced Instruction Set Computer. The ISA is composed of instructions that all have exactly the same size, usually 32 bits. Thus they can be pre-fetched and pipelined successfully. All ALU instructions have 3 operands which are only registers. The only memory access is through explicit LOAD/STORE instructions.

Thus A = B + C will be assembled as:

```
LOAD   R1, A
LOAD   R2, B
ADD     R3, R1, R2
STORE  C, R3
```

Although it takes 4 instructions we can reuse the values in the registers.

Why this architecture is called RISC? What is reduced about it?

The answer is that to make all instructions the same length the number of bits that are used for the opcode is reduced. Thus fewer instructions are provided. The instructions that were thrown out are the less important string and BCD (binary-coded decimal) operations. In fact, now that memory access is restricted there aren't several kinds of MOV or ADD instructions. Thus the older architecture is called CISC (Complete Instruction Set Computer). RISC architectures are also called *LOAD/STORE* architectures.

The number of registers in RISC is usually 32 or more. The first RISC CPU the MIPS 2000 has 32 GPRs as opposed to 16 in the 68xxx architecture and 8 in the 80x86 architecture. The only disadvantage of RISC is its code size. Usually more instructions are needed and there is a waste in short instructions (POP, PUSH).

So why are there still CISC CPUs being developed? Why is Intel spending time and money to manufacture the Pentium II and the Pentium III?

The answer is simple, backward compatibility. The IBM compatible PC is the most common computer in the world. Intel wanted a CPU that would run all the applications that are in the hands of more than 100 million users. On the other hand Motorola which builds the 68xxx series which was used in the Macintosh made the transition and together with IBM and Apple built the Power PC (PPC) a RISC CPU which is installed in the new Power Macs. As of now Intel and the PC manufacturers are making more money but with Microsoft playing in the RISC field as well (Windows NT runs on Compaq's Alpha) and with the promise of Java the future of CISC isn't clear at all.

An important lesson that can be learnt here is that superior technology is a factor in the computer industry, but so are marketing and price as well (if not more).