Instruction set architecture ISA

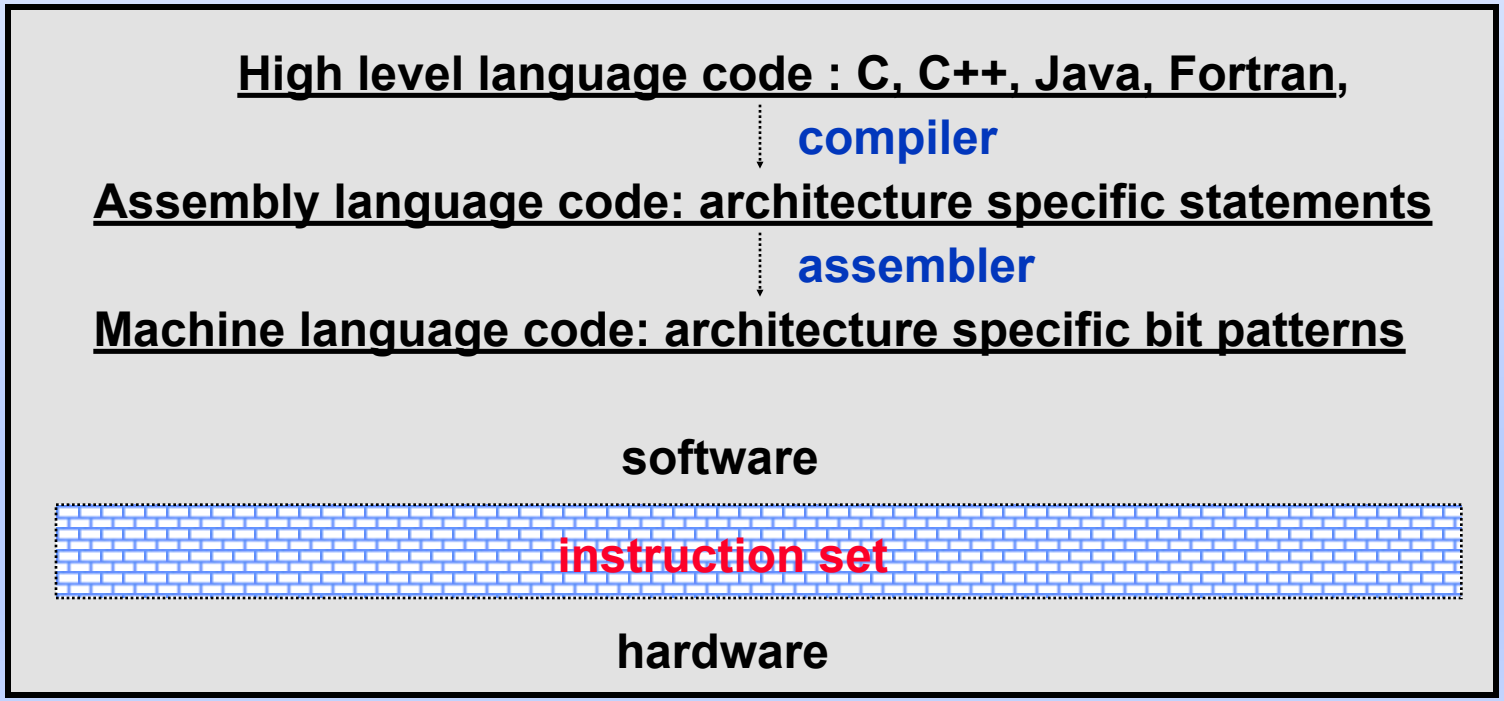
CISC VS RISC

What is an Instruction Set?

- The complete collection of instructions that are understood by a CPU
- Machine Code
- Binary
- Usually represented by assembly codes

Instruction Set Architecture (ISA)

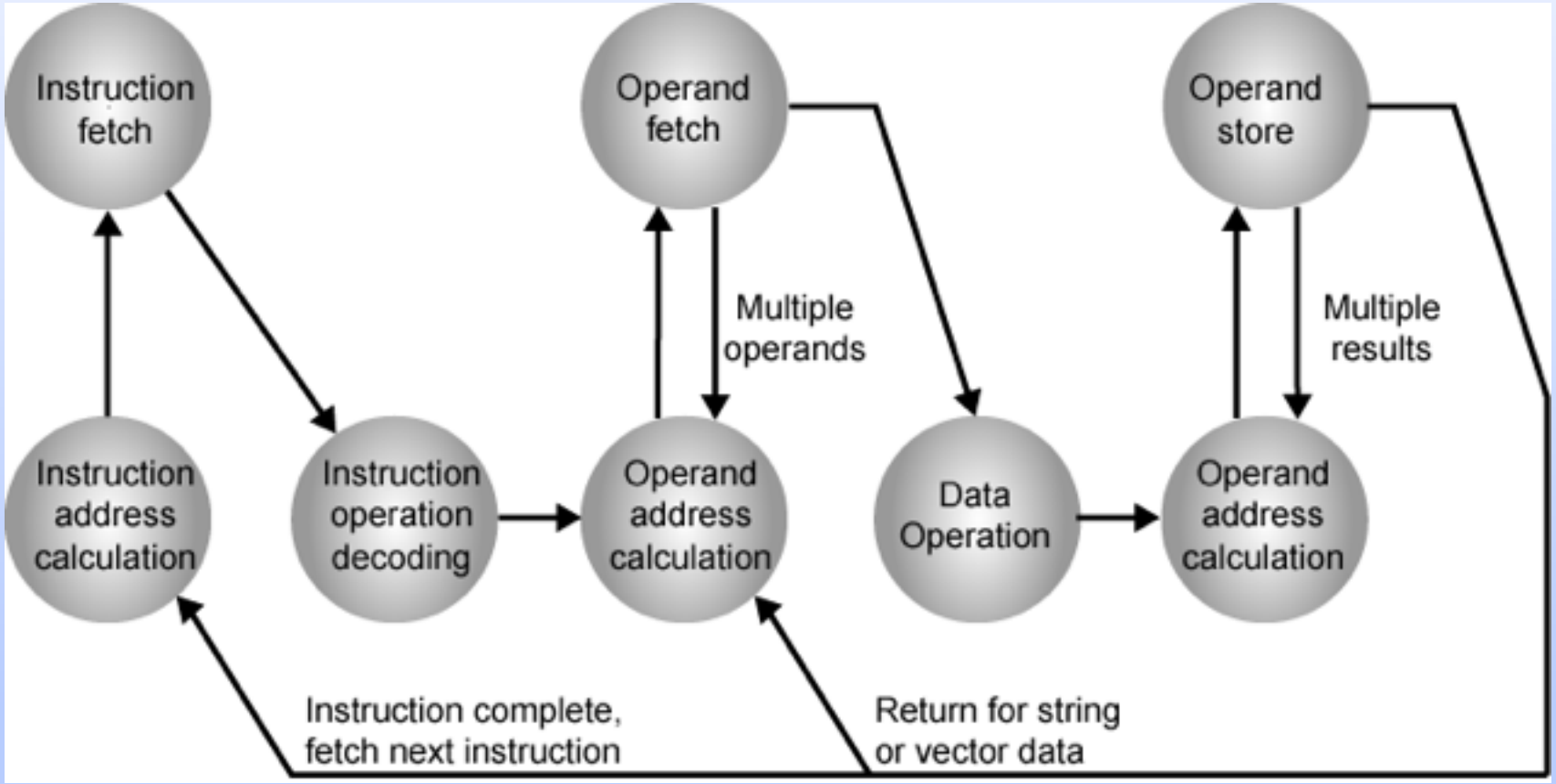
- Serves as an interface between software and hardware.
- Provides a mechanism by which the software tells the hardware what should be done.



Elements of an Instruction

- Operation code (Op code)
  - Do this
- Source Operand reference
  - To this
- Result Operand reference
  - Put the answer here
- Next Instruction Reference
  - When you have done that, do this...

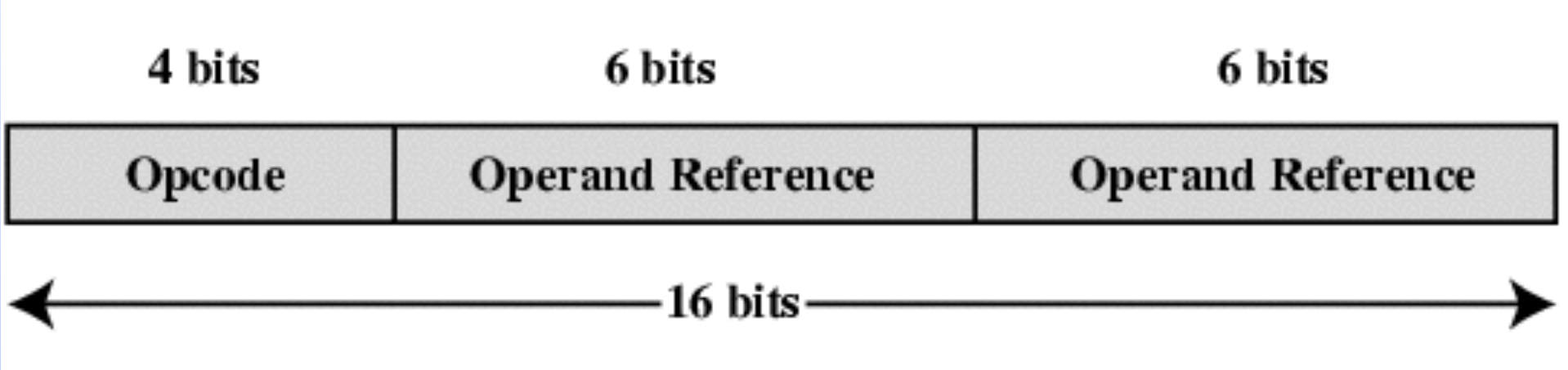
Instruction Cycle State Diagram



Instruction Representation

- In machine code each instruction has a unique bit pattern
- For human consumption (well, programmers anyway) a symbolic representation is used
  - e.g. ADD, SUB, LOAD
- Operands can also be represented in this way
  - ADD A,B

Simple Instruction Format



Instruction Types

- Data processing
- Data storage (main memory)
- Data movement (I/O)
- Program flow control

Number of Addresses (a)

- 3 addresses
  - Operand 1, Operand 2, Result
  - a = b + c;
  - May be a forth - next instruction (usually implicit)
  - Not common
  - Needs very long words to hold everything

Number of Addresses (b)

- 2 addresses
  - One address doubles as operand and result
  - a = a + b
  - Reduces length of instruction
  - Requires some extra work
    - » Temporary storage to hold some results

Number of Addresses (c)

- 1 address
  - Implicit second address
  - Usually a register (accumulator)
  - Common on early machines

Number of Addresses (d)

- 0 (zero) addresses
  - All addresses implicit
  - Uses a stack
  - e.g. push a
  - push b
  - add
  - pop c

  - c = a + b

How Many Addresses

- More addresses
  - More complex (powerful?) instructions
  - More registers
    - » Inter-register operations are quicker
  - Fewer instructions per program
- Fewer addresses
  - Less complex (powerful?) instructions
  - More instructions per program
  - Faster fetch/execution of instructions

Design Decisions (1)

- Operation repertoire
  - How many ops?
  - What can they do?
  - How complex are they?
- Data types
- Instruction formats
  - Length of op code field
  - Number of addresses

Design Decisions (2)

- **Registers**
  - **Number of CPU registers available**
  - **Which operations can be performed on which registers?**
- **Addressing modes**

- **RISC v CISC**

History of RISC/CISC

- 1950s IBM instituted a research program
- 1964 Release of System/360
- Mid-1970s improved measurement tools demonstrated on CISC
- 1975 801 project initiated at IBM's Watson Research Center
- 1979 32-bit RISC microprocessor (801) developed led by Joel Birnbaum
- 1984 MIPS developed at Stanford, as well as projects done at Berkeley
- 1988 RISC processors had taken over high-end of the workstation market
- Early 1990s IBM's POWER (*Performance Optimization With Enhanced RISC*) architecture introduced w/ the RISC System/6k
  - AIM (Apple, IBM, Motorola) alliance formed, resulting in PowerPC

What is CISC?

- CISC is *Complex Instruction Set Computer and are chips* that are easy to program and which make efficient use of memory. Since the earliest machines were programmed in assembly language and memory was slow and expensive, the CISC philosophy made sense, and was commonly implemented in such large computers as the PDP-11 and the DECsystem 10 and 20 machines.
- Most common microprocessor designs such as the Intel 80x86 and Motorola 68K series followed the CISC philosophy.
- But recent changes in software and hardware technology have forced a re-examination of CISC and many modern CISC processors are hybrids, implementing many RISC principles.
- CISC was developed to make compiler development simpler. It shifts most of the burden of generating machine instructions to the processor. For example, instead of having to make a compiler write long machine instructions to calculate a square-root, a CISC processor would have a built-in ability to do this.

CISC Attributes

The design constraints that led to the development of CISC (small amounts of slow memory and fact that most early machines were programmed in assembly language) give CISC instructions sets some common characteristics:

- A 2-operand format, where instructions have a source and a destination. Register to register, register to memory, and memory to register commands. Multiple addressing modes for memory, including specialised modes for indexing through arrays
- Variable length instructions where the length often varies according to the addressing mode
- Instructions which require multiple clock cycles to execute.

E.g. Pentium is considered a modern CISC processor

Most CISC hardware architectures have several characteristics in common:

- **Complex instruction-decoding logic, driven by the need for a single instruction to support multiple addressing modes.**
- **A small number of general purpose registers. This is the direct result of having instructions which can operate directly on memory and the limited amount of chip space not dedicated to instruction decoding, execution, and microcode storage.**
- **Several special purpose registers. Many CTSC designs set aside special registers for the stack pointer, interrupt handling, and so on. This can simplify the hardware design somewhat, at the expense of making the instruction set more complex.**
- **A 'Condition code" register which is set as a side-effect of most instructions. This register reflects whether the result of the last operation is less than, equal to, or greater than zero and records if certain error conditions occur.**

At the time of their initial development, CISC machines used available technologies to optimize computer performance.

- **Microprogramniing is as easy as assembly language to implement, and much less expensive than hardwiring a control unit.**
- **The ease of microcoding new instructions allowed designers to make CISC machines upwardly compatible: a new computer could run the same programs as earlier computers because the new computer would contain a superset of the instructions of the earlier computers.**
- **As each instruction became more capable, fewer instructions could be used to implement a given task. This made more efficient use of the relatively slow main memory.**
- **Because microprogram instruction sets can be written to match the constructs of high-level languages, the compiler does not have to be as complicated.**

CISC Disadvantages

Designers soon realised that the CISC philosophy had its own problems, including:

- **Earlier generations of a processor family generally were contained as a subset in every new version - so instruction set & chip hardware become more complex with each generation of computers.**
- **So that as many instructions as possible could be stored in memory with the least possible wasted space, individual instructions could be of almost any length - this means that different instructions will take different amounts of clock time to execute, slowing down the overall performance of the machine.**
- **Many specialized instructions aren't used frequently enough to justify their existence - approximately 20% of the available instructions are used in a typical program.**
- **CISC instructions typically set the condition codes as a side effect of the instruction. Not only does setting the condition codes take time, but programmers have to remember to examine the condition code bits before a subsequent instruction changes them.**

What is RISC?

- **RISC is *Reduced Instruction Set Computer.* is a type of microprocessor architecture that utilizes a small, highly-optimized set of instructions, rather than a more specialised set of instructions often found in other types of architectures.**
- **History**
  The first RISC projects came from IBM, Stanford, and UC-Berkeley in the late 70s and early 80s. The IBM 801, Stanford MIPS, and Berkeley RISC 1 and 2 were all designed with a similar philosophy which has become known as RISC. Certain design features have been characteristic of most RISC processors:
  - ***one cycle execution time*: RISC processors have a CPI (clock per instruction) of one cycle. This is due to the optimization of each instruction on the CPU and a technique called PIPELINING**
  - ***pipelining*: a techique that allows for simultaneous execution of parts, or stages, of instructions to more efficiently process instructions;**
  - ***large number of registers*: the RISC design philosophy generally incorporates a larger number of registers to prevent in large amounts of interactions with memory**

RISC Attributes

The main characteristics of CISC microprocessors are:

- **Extensive instructions.**
- **Complex and efficient machine instructions.**
- **Microencoding of the machine instructions.**
- **Extensive addressing capabilities for memory operations.**
- **Relatively few registers.**

In comparison, RISC processors are more or less the opposite of the above:

- **Reduced instruction set.**
- **Less complex, simple instructions.**
- **Hardwired control unit and machine instructions.**
- **Few addressing schemes for memory operands with only two basic instructions, LOAD and STORE**
- **Many symmetric registers which are organised into a register file.**

Pipelining

RISC Pipelines

A RISC processor pipeline operates in much the same way, although the stages in the pipeline are different. While different processors have different numbers of steps, they are basically variations of these five, used in the MIPS R3000 processor:

- fetch instructions from memory
- read registers and decode the instruction
- execute the instruction or calculate an address
- access an operand in data memory
- write the result into a register

RISC Disadvantages

- There is still considerable controversy among experts about the ultimate value of RISC architectures. Its proponents argue that RISC machines are both cheaper and faster, and are therefore the machines of the future.
- However, by making the hardware simpler, RISC architectures put a greater burden on the software. Is this worth the trouble because conventional microprocessors are becoming increasingly fast and cheap anyway?

CISC versus RISC

CISC	RISC
Emphasis on hardware	Emphasis on software
Includes multi-clock complex instructions	Single-clock, reduced instruction only
Memory-to-memory: "LOAD" and "STORE" incorporated in instructions	Register to register: "LOAD" and "STORE" are independent instructions
Small code sizes, high cycles per second	Low cycles per second, large code sizes
Transistors used for storing complex instructions	Spends more transistors on memory registers

Summation

- As memory speed increased, and high-level languages displaced assembly language, the major reasons for CISC began to disappear, and computer designers began to look at ways computer performance could be optimized beyond just making faster hardware.
- One of their key realizations was that a sequence of simple instructions produces the same results as a sequence of complex instructions, but can be implemented with a simpler (and faster) hardware design. (Assuming that memory can keep up.) RISC (Reduced Instruction Set Computers) processors were the result.
- CISC and RISC implementations are becoming more and more alike. Many of today’s RISC chips support as many instructions as yesterday's CISC chips. And today's CISC chips use many techniques formerly associated with RISC chips.
- To some extent, the argument is becoming moot because CISC and RISC implementations are becoming more and more alike. Many of today's RISC chips support as many instructions as yesterday's CISC chips. And today's CISC chips use many techniques formerly associated with RISC chips.