COM1006 Devices and Networks (Autumn)
COM1090 Computer Architectures

Lecture #9

Instruction set architecture

Dr Dirk Sudholt
Department of Computer Science
University of Sheffield

d.sudholt@sheffield.ac.uk
http://staffwww.dcs.shef.ac.uk/~dirk/campus_only/com1006/

Based on Chapter 5 in Clements, Principles of Computer Hardware

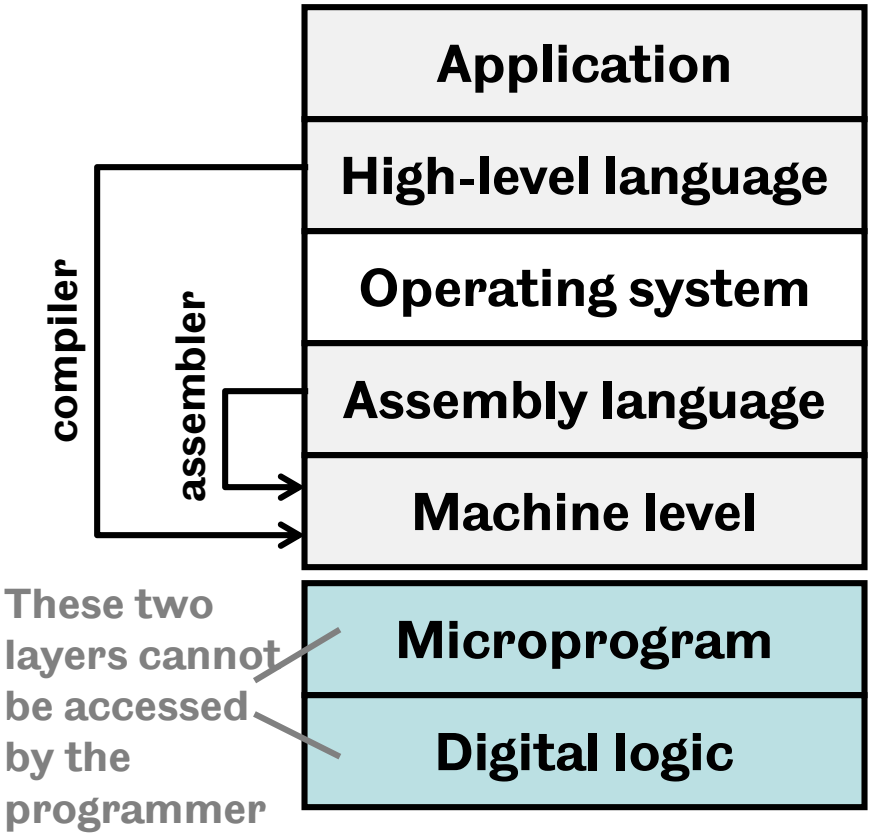
Aims of this lecture

- To discuss how instructions can be implemented in computers.
- To briefly explain the big picture regarding the central processing unit (CPU) and memory organisation.
- To discuss the use and advantages of registers.
- To briefly compare RISC and CISC architectures.
- To show how architectures can be classified according to the number of operands in their instruction set.
- To introduce addressing modes used in instructions.
- To introduce the Motorola 68K as a typical CISC architecture.

Instruction set architecture

- An **instruction set architecture** (ISA) is an abstract model of a computer that describes **what** it does, rather than **how** it does it.
- The ISA forms a boundary between the hardware and software.
- Design of ISA is determined by many factors, including
  - code density (combined size of instructions required to perform a particular task),
  - cost,
  - legacy support,
  - complexity,
  - expandability and
  - performance (pipelining).

Machine levels and architectural layers



Word processor, game, database

Java, C++, Python, etc

Windows, UNIX, MacOS

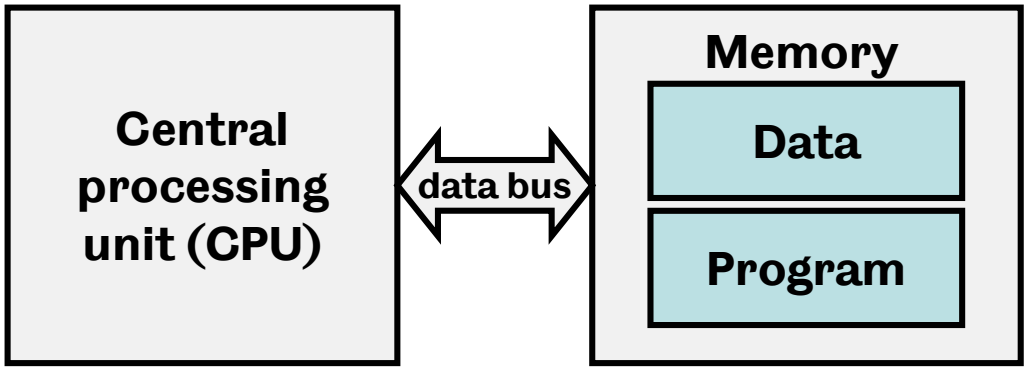
Note that operating system is rather different to the other layers because it provides facilities needed by higher levels (e.g. memory management)

Microprogram (in firmware) determines how machine code instructions are performed by digital logic

Introduction to the CPU

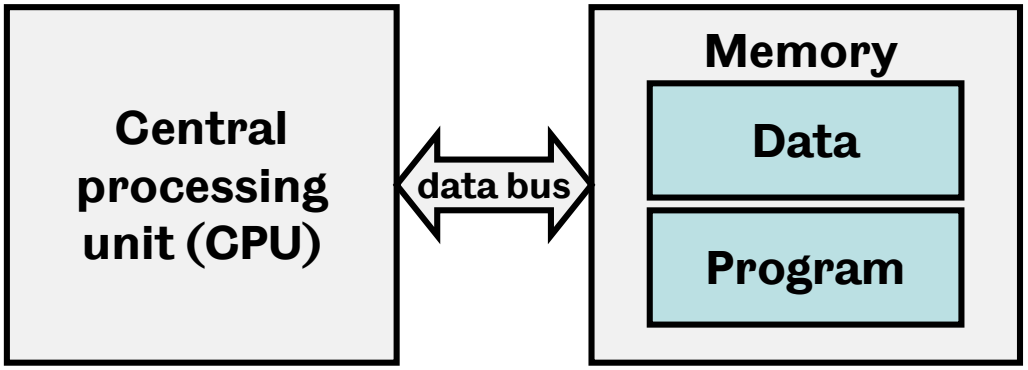
- The **central processing unit** (CPU) is the 'brain' of a computer, responsible for instruction execution.
- Usually computers store **programs** and **data** in a single memory system.
- Such computers are called **von Neumann machines** (in honour of John von Neumann (1903-1957).
- The CPU reads the sequence of instructions that make up a program one-by-one from memory.
- Instructions consist of **opcodes** and **operands**.
- In order to execute an instruction, the CPU may retrieve data from memory (**read cycle**) or write data generated by the instruction to memory for storage (**write cycle**).

Instruction execution (briefly)



- Consider ADD X, Y, **Z** (corresponding to Z=X+Y). The CPU:
  1. fetches the instruction via data bus and decodes it
  2. fetches values in memory locations X and Y via data bus
  3. adds the two values to give a result
  4. writes the result to memory location Z via data bus

The von Neumann bottleneck



- Most computers have a single bidirectional data bus for transfer of data between the CPU and memory.
- CPU needs to read both instructions and data via the data bus.
- This can lead to congestion that slows the computer down - this is the **von Neumann bottleneck**.

Memory and registers

- A **register** is a storage device that holds a word like a memory location, but it is located **within** the CPU.
- Registers can be accessed **faster** than memory locations.
- **Memory traffic is reduced** as the data bus is not being used to access registers (alleviates von Neumann bottleneck).
- Registers can be used effectively by compilers.
- There are a few registers, as opposed to millions of memory locations; fewer bits are therefore required to specify a register, **keeping instructions short**.
- Registers are used as temporary storage locations to store frequently used data, and also to record state of CPU.

RISC and CISC processors

- **Complex instruction set computer** (CISC) e.g., Intel Pentium, Motorola 68K.
- **Reduced instruction set computer** (RISC) e.g., ARM, PowerPC
- CISC have large, irregular instruction sets and can perform operations directly on data in memory.
- RISC have many on-chip registers and do not allow operations directly on data in memory ("load/store" architecture).
- RISC philosophy – doing more simple instructions may be faster than doing fewer complex instructions (due to pipelining).
- Note: 'reduced' in RISC means that the work done by a single instruction is less, not that the size of the instruction set is less.

The Motorola 68K family

- In much of what follows we'll use the Motorola 68K processor family as an example (typical CISC processor).
- Architecture of a processor family is defined by its **register set**, **instruction set** and **addressing modes**.
- The 68K register set consists of:
  - 8 **data registers** D0-D7 for scratchpad information
  - 8 **address registers** A0-A7 (pointers) used for data access
  - A **program counter** which contains the address of the next instruction to be executed
  - A **status register** that contains information about the state (operating mode) of the computer.

Instruction set

- Instructions are classified by what they do, and the number of operands that they take:
  - **data movement** (copy from one location to another)
  - **data processing** (operate on data)
  - **flow control** (alter order in which instructions are executed)
- The 68K has a two-address instruction format; other machines have different instruction formats.
- Before going further, introduce a shorthand notation for explaining what instructions do.

Register transfer language

- To explain how the CPU operates we'll use a shorthand called **register transfer language** (RTL).
- This is an algebraic notation, **not** a programming language.
- Square brackets indicate contents of a memory location:
  [6] = 3 means "memory location 6 contains the value 3"
- Left arrow indicates transfer of data between source (right side) and destination (left side)
- Registers are referred to by name (e.g., D4)

  [D4] ← 5 means "put value 5 in register D4"
  [D4] ← [5] means "put value in memory location 5 into D4"

Quick quiz on RTL

- **Explain the meaning of the following expressions written in RTL:**

  [20] = 6

  [20] ← 6

  [20] ← [6]

  [20] ← [6] + 3

  [20] ← [[6]]

Addressing modes

- Computers perform operations on data, so you need to specify **where** the data comes from.
- This is done by a number of **addressing modes**:
  - immediate addressing (operand is data)
  - absolute addressing (operand is an address)
  - indirect addressing (operand is a pointer)

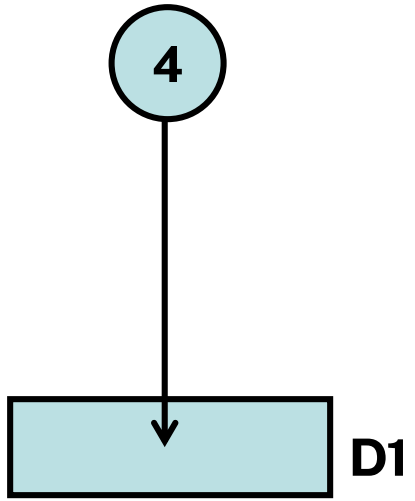
Immediate (literal) addressing

- In immediate addressing the **operand is the data itself** (not a reference to a memory location).
- In 68K assembler immediate addressing is indicated by prefixing the operand with '#'.

  Examples:

  ADD #4, **D0**
  [D0] ← [D0]+4

  MOVE #4, **D1**
  [D1] ← 4



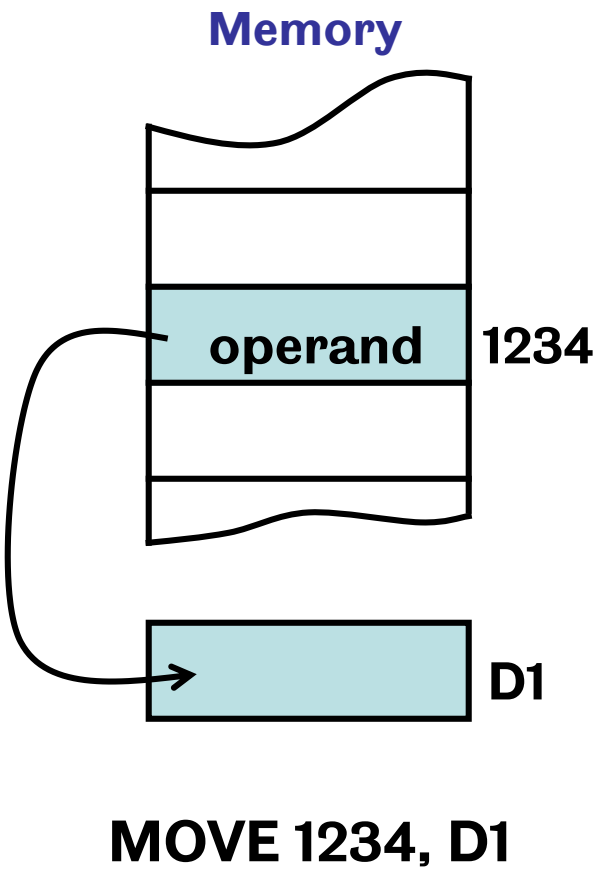
MOVE #4, D1

Absolute addressing

- In absolute addressing, the operand specifies the **address** of the data.
- The operand is a location in memory or a register.

  Examples:

- ADD P, **D1**
  P is a memory address
  [D1] ← [D1]+[P]
- CLR 1234
  Set address 1234 to zero
  [1234] ← 0



Indirect addressing

- Indirect addressing specifies a **pointer** to the actual operand.

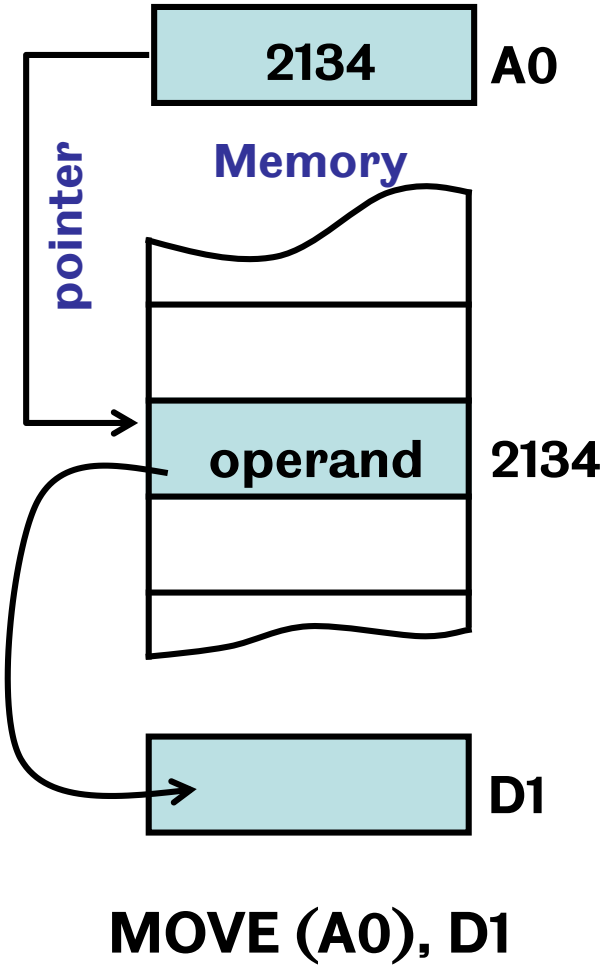
  Example:

  MOVE (A0), **D1**

  - First read the contents of register A0 to find the address of the operand
  - Read the operand at that address
  - Copy operand to register D1

  [D1] ← [[A0]]

- Useful for dealing with **tables** of data (point to the start and then increment the pointer to get successive elements).



MOVE (A0), D1

Three-address machines

- Typical three-address instruction:

  ADD P, Q, **R** $[R] \leftarrow [P]+[Q]$

- Add contents of P to contents of Q and put the result in R (we show destination in bold type). ADD is the **opcode**.
- In theory P, Q and R could all be memory addresses but in practice instructions would be very **long** (e.g., using 32-bit addresses requires 96 bits to specify the operands).

Why is this a problem?

- RISC machines such as the ARM use three-address instructions but addresses refer to a small number (e.g., 32) of fast on-chip registers.

Two-address machines

- The 68K uses a two-address instruction format:

  ADD P, **Q** $[Q] \leftarrow [P] + [Q]$

- This requires **less storage and memory traffic** at the price of **overwriting** one operand.
- Most computer instructions don't access two memory locations: one operand needs to be a **register**.
- Example: the 68K ADD instruction can be written

ADD D0, **D1**	$[D1] \leftarrow [D1] + [D0]$	Register-to-register
ADD P, **D2**	$[D2] \leftarrow [D2] + [P]$	Memory-to-register
ADD D7, **P**	$[P] \leftarrow [P] + [D7]$	Register-to-memory

One-address machines

- One operand is specified , the second (implicit) operand is a fixed register called the **accumulator**.

  ADD P [A] ← [A] + [P]

- Typical of 8-bit machines such as the Intel 8080 and Motorola 6800.
- Example: implement R = P + Q on the 6800 processor:

  LDA P [A] ← [P]
  ADD Q [A] ← [A] + [Q]
  STA R [R] ← [A]

- Code is **verbose**, but instructions are **short**.
- One-address machines are still used in low-cost, low-performance applications such as toys.

Zero-address (stack) machines

- A zero-address machine uses a stack for all computation.
- There are no registers, so data items must be pushed onto the stack. Results are left on the stack, so must be popped off in order to store them in memory.
- Example: implement R = P + Q on a stack machine

```
PUSH P      push contents of P onto stack
PUSH Q      push contents of Q onto stack
ADD         add top two stack elements, push result
POP R       pop top stack element, store in R
```

- Stack machines are largely experimental with one exception - the **Java Virtual Machine (JVM)**.

Summary

- An instruction set architecture (ISA) is an abstract model of a computer that describes what it does, rather than how it does it.
- Registers are faster than memory locations, reduce data bus traffic, and require fewer bits to be addressed.
- RISC and CISC represent different design approaches.
- ISAs can be characterised by the number of operands in instructions, use of registers and addressing modes.
- Instructions can have different addressing modes: immediate, absolute, and indirect.
- The Motorola 68K family is a typical CISC architecture.