Chapter 19
Processor and Computer Architecture

19.01 The control unit

While a program is being executed, the CPU is receiving a sequence of machine-code instructions. It is the responsibility of the control unit within the CPU to ensure that each machine instruction is handled correctly. There are two ways that a control unit can be designed to allow it to perform its function.

One method is for the control unit to be constructed as a logic circuit. This is called the hard-wired solution. The machine-code instructions are handled directly by hardware.

The alternative is for the control unit to use microprogramming. In this approach, the control unit contains a ROM component in which is stored the microinstructions or microcode for microprogramming. This is often referred to as firmware. The choice of which method is used is largely dependent on the type of processor.

19.02 CISC and RISC processors

The 'architecture' of a processor can be defined in a number of ways. From the point of view of a sophisticated programmer, the architecture involves the following:

- the instruction set
- the instruction format
- the addressing modes
- the registers accessible by instructions.

The choice of the instruction set is the main factor in deciding on a suitable architecture. One view is that the instruction set should be chosen so that it can be clearly applied to important problems, that only simple equipment is required and that important problems are handled speedily. An opposing view is that it should be chosen to suit the needs of high-level languages.

Early developments in computing led to the latter view becoming dominant. Computer systems contained what would now be referred to as CISC (Complex Instruction Set Computers) processors with the complexity increasing with the advent of new systems. However, the philosophy began to be challenged in the late 1970s. It was argued that RISC (Reduced Instruction Set Computers) would be a better approach. Table 19.01 contains a number of features that distinguish RISC from CISC.

RISC	CISC
Fewer instructions	More instructions
Simpler instructions	More complex instructions
Small number of instruction formats	Many instruction formats
Single-cycle instructions whenever possible	Multi-cycle instructions
Fixed-length instructions	Variable-length instructions
Only load and store instructions to address memory	Many types of instructions to address memory
Fewer addressing modes	More addressing modes
Multiple register sets	Fewer registers
Hard-wired control unit	Microprogrammed control unit
Pipelining easier	Pipelining more difficult

Table 19.01 Comparison of RISC with CISC

It can be seen that 'reduced' affects more than just the number of instructions. The simplicity of the instructions allows data to be stored in registers and manipulated in them with no resource to memory access other than that necessary for initial loading and possible final storing. The simplicity also allows hard-wiring inside the control unit with limited complexity required.

In contrast, the specialised instructions that can be part of a CISC architecture often require repeated memory access. The complexity of some of the instructions makes hard-wiring extremely difficult so microprogramming is the norm. However, the increased complexity of instructions for CISC is often because they more closely match high-level language constructs. This means that compiler writing becomes much easier for a CISC processor.

Extension question 19.01

Can you find out whether the processors in any systems you are using are described as RISC or CISC?

One of the major driving forces for creating RISC processors was the opportunity they would provide for efficient **pipelining**. Pipelining is a form of parallelism applied specifically to instruction execution. Other forms of parallelism are discussed in Section 19.03.

KEY TERMS

Pipelining: instruction-level parallelism

The underlying principle of pipelining is that the fetch–decode–execute cycle described in Chapter 5 (Section 5.04) can be separated into a number of stages. One possibility is a five-stage model consisting of:

- instruction fetch (IF)
- instruction decode (ID)
- operand fetch (OF)
- instruction execute (IE)
- result write back (WB).

Processor units	Clock cycles 1	2	3	4	5	6	7
IF	1.1	2.1	3.1	4.1	5.1	6.1	7.1
ID		1.2	2.2	3.2	4.2	5.2	6.2
OF			1.3	2.3	3.3	4.3	5.3
IE				1.4	2.4	3.4	4.4
WB					1.5	2.5	3.5

Figure 19.01 Pipelining for five-stage instruction handling

Figure 19.01 shows how pipelining would work with this five-stage breakdown of instruction handling. For pipelining to be implemented, the construction of the processor must have five independent units, with each handling one of the five stages identified. This explains the need for a RISC processor to have many register sets; each processor unit must have access to its own set of registers. Figure 19.01 uses the representation 1.1, 1.2 and so on to define the instruction and the stage of the instruction. Initially only the first stage of the first instruction has entered the pipeline. At clock cycle 6 the first instruction has left the pipeline, the last stage of instruction 2 is being handled and instruction 6 has just entered.

It can be seen that once under way the pipeline is handling five stages of five individual instructions. In particular, at each clock cycle the complete processing of one instruction has finished. Without the pipeline the processing time would be five times longer.

One issue that has to be dealt with regarding a pipelined processor is interrupt handling. The discussion in Chapter 5 (Section 5.06) referred to a processor with instructions handled sequentially. In the pipelined system described above there will be five instructions in the pipeline when an interrupt occurs. One option for handling the interrupt is to erase the pipeline contents for the latest four instructions to have entered. Then the normal interrupt-handling routine can be applied to the remaining instruction. The other option is to construct the individual units in the processor with individual program counter registers. This allows current data to be stored for all of the instructions in the pipeline while the interrupt is handled.

Discussion Point:

Consider the two consecutive instructions:

```
ADD R1, R2, R3
ADD R5, R1, R4
```

These are typical three-register instructions favoured for RISC. The first adds the contents of registers R2 and R3 and stores the result in R1. The next instruction is similar but uses the value stored in R1. In a pipelined structure, the second instruction will be reading the contents of R1 before the previous instruction has placed the value there. How could this potential problem be overcome?

19.03 Parallel processing

Parallel processor systems

One computer can have multiple processors running in parallel.

In principle, there are four categories of system:

- SISD (Single Instruction Single Data stream)
- SIMD (Single Instruction Multiple Data stream)
- MISD (Multiple Instruction Single Data stream)
- MIMD (Multiple Instruction Multiple Data stream).

SISD (Single Instruction Single Data stream) is the typical arrangement found in early personal computers. There is a single processor so no processor parallelism. The single data stream just means one memory.

SIMD (Single Instruction Multiple Data stream) describes how an array or vector processor works. The multiple processors each have their own memory. One instruction is input and each processor executes this instruction using data available in its dedicated memory.

MISD (Multiple Instruction Single Data stream) isn't implemented in commercial products.

MIMD (Multiple Instruction Multiple Data stream) has examples in modern personal computers which are of the symmetric multiprocessor type using identical processors. In this case, each processor executes a different individual instruction. The multiple data stream can be provided by a single memory suitably partitioned. Each processor might have a dedicated cache memory.

Parallel computer systems

Examples of one type of multicomputer system are called massively parallel computers. These are the systems used by large organisations for computations involving highly complex mathematical processing. They are the latest in an evolution of what have traditionally been called 'supercomputers'. The major difference in architecture is that instead of having a bus structure to support multiple processors there is a network infrastructure to support multiple computer units. The programs running on the different computers can communicate by passing messages using the network.

An alternative type of multicomputer system is cluster computing, where a very large number of PCs are networked.

CHAPTER 19	
PAPER	**QUESTION**
w15/31	4
w15/32	4
w17/31	2
w17/32	2
w18/32	5
s19/32	8
w19/31	9