Architectural considerations for a M88000 superscalar RISC processor

Steve Heath*

Abstract

With current RISC performances being matched or even surpassed by CISC processors like the MC68040, the future direction for RISC lies in the concept of multiple execution machines where multiple instructions are executed per clock. Although, the basic tenets of RISC design of fixed instruction length, regular decoding, and non-destructive data operations offer a big advantage over CISC architectures in implementing such designs, there are still several problems to overcome. The ease of solution is frequently dependent on architectural decisions made early in its implementation. This paper examines the Motorola MC88100 processor architecture and explains why certain key features were implemented and their benefit to later generations like the forthcoming MC88110 in breaking the one instruction per clock barrier. Topics discussed include data dependency between instructions and some suggested optimising techniques, pipeline control and its importance, condition evaluation, and a brief overview of the MC88110.

Why move to multiple instructions per clock?

With the arrival of CISC devices like the MC68040, the performance advantage of RISC has been largely unchallenged. However, with a CISC processor offering the benefit of 20 + MIPS with the advantage of man years of applications, compilers and operating systems, the gains offered by RISC appear to be small. The flaw with this argument is the assumption that RISC architectures have stood still and not progressed further. This is not so.

Examination of the three standard techniques used to increase processor performance show that two of the three techniques, increasing the processor clock and expanding the instruction set are equally applicable to CISC processors as well as RISC architectures. The third technique of increasing the internal parallelism and executing multiple instructions per clock is better suited to RISC architectures than CISC. Unless careful thought has been given to the architecture, certain features can create barriers that are difficult to overcome in any multiple execution design. The main difficulties arise in removing data dependencies and controlling instruction execution.

Data Dependency

Data dependency arises when adjacent instructions require data that is not yet available because of external memory bus delays or because the preceding instructions have not yet computed the data or similar problems. An example of this difficulty is also encountered with single instruction processors that have multiple execution units with different pipeline lengths such as a 5 stage floating point unit and a 3 stage

*Motorola Semiconductors, 69 Buckingham St, Aylesbury, Bucks.

integer unit. The problem arises when a floating point instruction is executed followed by an integer instruction which moves the result. The first instruction may take 5 clocks to complete but the second instruction only 3. Unless there is some intervention by hardware or software, the second instruction will execute using wrong data(figure 1).

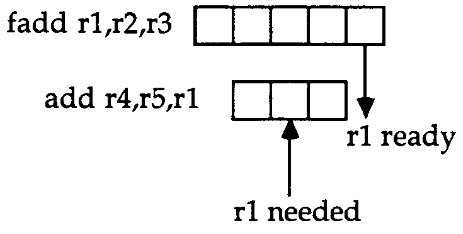

Figure 1: Pipeline synchronisation.

This situation is frequently solved by software techniques that optimise the code by separating the conflicting instructions either by NOP instructions or by instructions from elsewhere in the code sequence(figure 2).

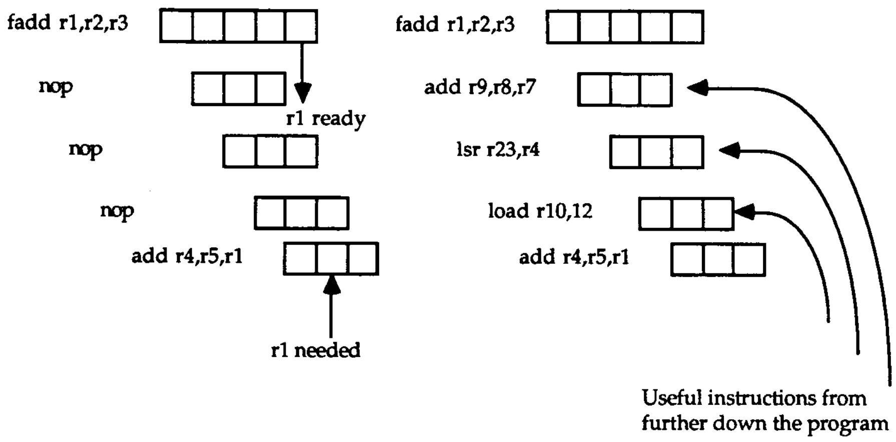

Figure 2: A software method of removing dependency

This software based solution can cause potential problems for binary compatible. If the required instruction separation is increased, binary code with a smaller separation will not execute correctly and binary compatibility is lost.

A variation of this problem can also arise if a floating point unit is not pipelined. It may not be able to accept sequential floating point instructions and therefore either prevent any further execution or rely on software to schedule any further floating point instructions until after the current instruction has completed. This can again cause potential problems for binary compatible. If two floating point instructions are separated by two other instructions to allow correct operation, the binary software will not work on another generation that may require a larger separation(figure 3).

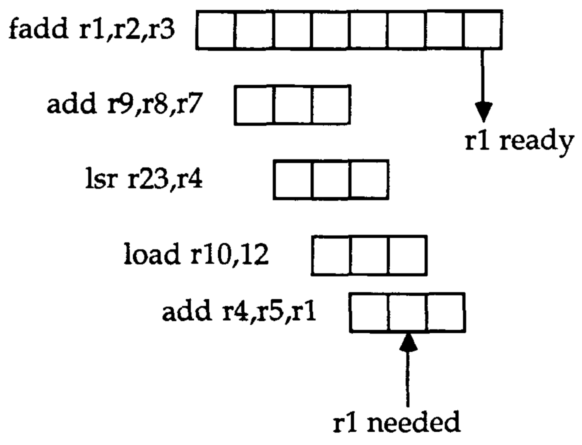

Figure 3: Processor corruption.

Further problems can be encountered if there are any dedicated resources such as registers or condition code flags. The M88000 architecture does not have any dedicated registers and any comparison results are allocated by the compiler to any one of its registers.

If multiple instructions are executed, the scope of these problems dramatically increase. With a single instruction design, the extent of the problem is effectively defined as the number of instructions that could be executed during the delay period. If a maximum dependency is 5 clocks, then the scope of the dependency would be 5 instructions. With a processor design executing 8 instructions per clock, 40 instructions would need to be checked. This increases the load on the compiler technology. Further complications can arise with branching where the compiler might have to check multiple paths. It is for these reasons that the M88000 family have used additional hardware to control the execution unit pipelines and data dependency.

Scoreboarding

The MC88100 maintains pipeline synchronisation by an internal scoreboard. Each register has a scoreboard bit which is set as a result of a decoding an instruction that modifies the register contents. As each instruction is decoded in the first stage of the pipeline, the registers are checked against the scoreboard. If a source register has a scoreboard bit set, the data is not yet available from a previous instruction and the execution will be delayed(figure 4). The destination register scoreboard bit is set and the instruction allowed to continue execution. The scoreboard is checked on a cycle by cycle basis. This mechanism provides a complete interlock that prevents correct execution. As a result, compiler optimisers can concentrate on optimising and not maintaining system synchronicity.

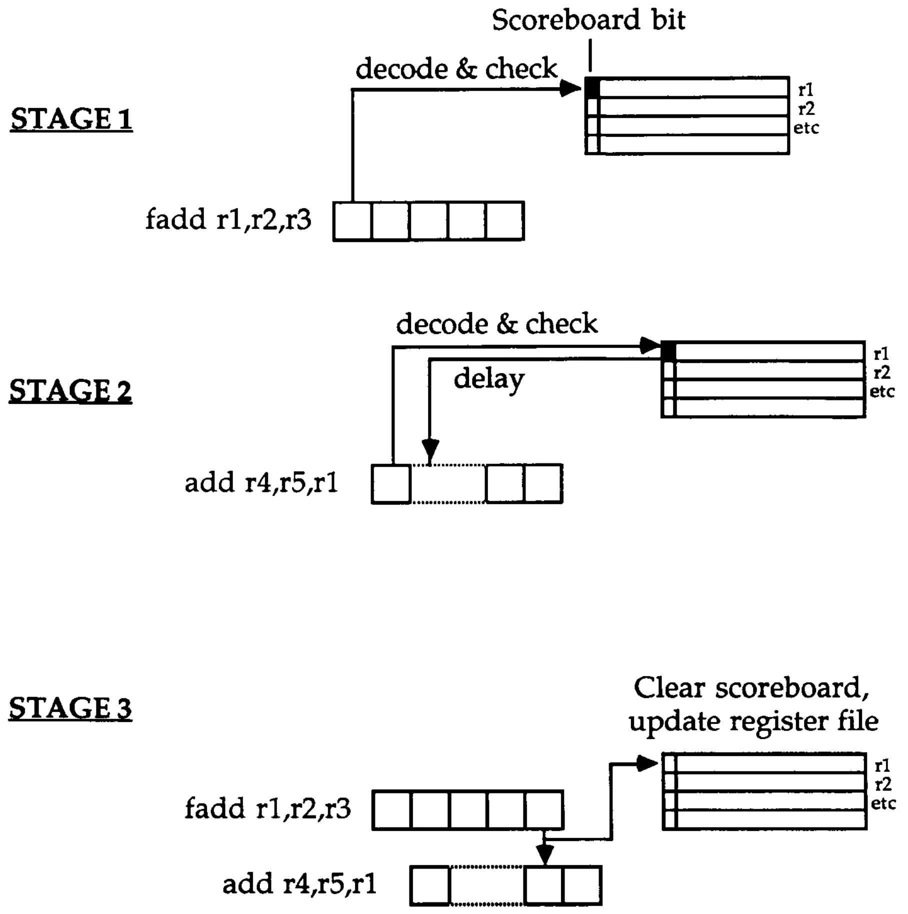

Figure 4: Scoreboarding.

Besides these techniques, the MC88100 processor uses feed forwarding to reduce pipeline delays by a single cycle. When the data is written to its destination register, it is automatically passed to the next instruction that is stalled and dependent on that data. By doing this, the waiting instruction does not need to interrogate the register file for the missing data and is not further delayed. This has other benefits in reducing the amount of bus traffic to the register file and thus allowing other instructions to update the register file that have had to wait.

Both these techniques can easily be adapted to a processor executing multiple instructions. Instead of checking a single instruction against the scoreboard, multiple instructions can be validated. If there is no dependency, execution can continue. If there is, the dependent instructions can continue as far as possible.

The MC88110 - a Symmetric Superscalar™ processor

The MC88110 is the next generation RISC processor in the M88000 family. It is a Symmetric Superscalar™ processor with multiple integer, floating point and graphics execution units, with integrated data and instruction caches and memory management units. Fabricated in 0.8 micron HCMOS and using 1.4 million transistors, the device offer 3-5 times performance over the current MC88100 and MC88200 CPU and cache MMU. The key to its performance is its Symmetric Superscalar™ internal architecture.

A Symmetric Superscalar™ processor should not be confused with multiprocessor or parallel processing designs, although they are very similar in that they have the capability of executing multiple instructions every clock cycle. The main difference is the level at which the allocation of resources is performed. With most parallel or multiple processor computers, the execution units virtually act independently and can be likened to several computer systems residing in a single chassis with a single controlling program. The MC88110 design performs this role at the VLSI hardware level so that the resource allocation is transparent to the user, thus enabling binary compatibility across a range of machines, irrespective of the number of instructions that can be executed per clock.

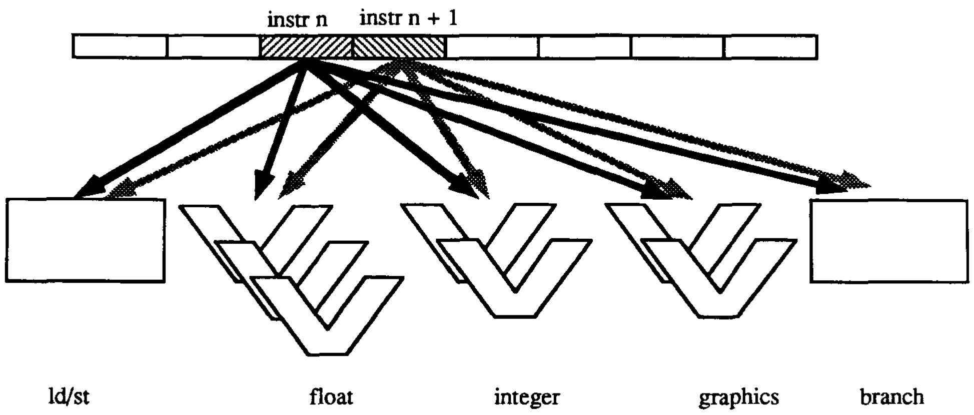

Figure 5: A Symmetric Superscalar™ Processor

The general principles behind Symmetric Superscalar™ are simple(figure 5). The execution units are replicated, with each unit capable of executing an instruction per clock cycle. These units are fed via an extremely wide bus so that a very large word comprising of multiple instructions, can be presented. A processor that executes 2 instructions per clock requires 2 op codes in every clock cycle. Each individual op code within it is allocated to a particular execution unit on every clock cycle. With CISC processors like the M68000 family whose op codes vary from 2 to 22 bytes, this creates an immediate problem of knowing exactly much data to bring in. In addition, the size of the op code is not dependent on the status of a couple of bits but is determined through multi-level decoding where individual bits may have several interpretations. Compare this with the simple regular decoding used within the M88000 processor instruction set. The op codes are all 32 bits in size and use a triadic register model capable of a+b=c data operations. This is far more efficient and removes the need for hardware to detect when an op code would overspill into the next allocated slot.

The next problem to overcome is the validity of the results. Although many instructions may have been executed, not all the results may be valid. If a branch instruction is encountered and taken, the instructions executed with it but located after it would not normally have been executed and their results should be discarded. Similarly, instructions that use results from previous instructions may not be valid because the data was not ready. The scoreboard technique inherent in the M88000 architecture provides the basic internal hardware control: instruction pairs are decoded, checked for data dependency and if necessary, one or both instructions may be delayed as necessary. The beauty of this system is that this is controlled by hardware and is transparent to the programmer who only sees a standard register model. As with its predecessor, this allows the compiler to concentrate on optimisation and not pipeline synchronisation.