Address and Data Register Separation on the M68000 Family

Technical Report No. 102

Dr F L Williams
Dr G B Steven

May 1990

Introduction

On the M68000 registers are partitioned into eight data registers and eight address registers[1]. Many authors consider this division to be a breakdown in instruction set orthogonality and a major shortcoming of the M68000 architecture[2][3]. However, an investigation at Hatfield Polytechnic into the impact of instruction set orthogonality on compiler code generation found that, in practice, this register division was not a problem.

FLEC[4], a Hatfield Polytechnic research compiler, which compiles a subset of Modula-2[5] onto the M68000, was written to assist in this investigation. Our observations are derived from our experience while writing FLEC.

A Summary of the M68000 Architecture

The programmer model of the M68000 is shown in Fig 1. The eight data registers can be used for byte, word and long word (32-bit) data manipulation. The eight address registers support 16 and 32-bit values and include separate stack pointers (A7) for user and supervisor mode. A comprehensive set of instructions are provided to manipulate the data registers, but only a limited number of instructions manipulate the address registers (Table 1).

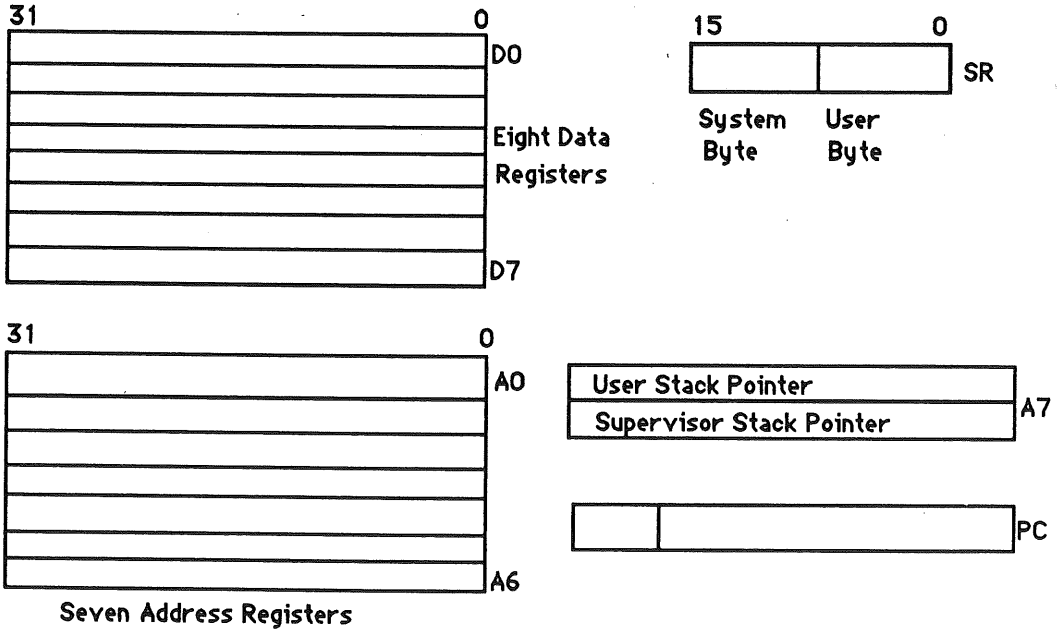

Fig 1 M68000 Programmer Model

The Case against Register Division

The arguments against the M68000 register division are based on the concept of orthogonality. An orthogonal instruction set allows every instruction to use every addressing mode or modifier in exactly the same manner[2]. Since data and address registers on the M68000 cannot be used interchangably, the division of registers into data and address registers represents a major departure from the principle of orthogonality.

As a result of this breakdown in instruction set regularity, the compiler writer might find it difficult to decide whether to return a result to a data register or an address register. The consequence would be either more compiler case analysis or redundant move instructions to transfer operands between the two register sets.

Dividing the registers into two sets will also inevitably lead to a non-optimum use of registers. It is unlikely that all programs will require eight data registers and eight address registers. Some programs, for example, will run better with 12 data and four address registers. This difficulty can only be resolved by providing 16 general-purpose registers.

The Case for Register Division

The case for register division is based on the concept of low-level typing. Address and data registers are provided to support different data types. Address registers are provided to support addresses or pointers while data registers are provided to support integers. Analogies can be drawn between this instance of low-level typing and the provision of separate floating-point registers.

If it is accepted that address registers are provided to support pointers, the instruction set should only provide sufficient address register operations to effectively manipulate pointers. Viewed in this light, multiply, divide and logical operations on pointers are meaningless and are quite rightly not provided. Similarly, byte operations on address registers are not required.

In spite of the above argument, dividing the register set is undesirable if there are no

compensating architectural gains elsewhere. However, on the M68000, major gains follow from the separation. It is possible to encode all the M68000 instructions into a single word, 16-bit format only because limited operations are provided on the address registers. In contrast, insisting on a single set of general-purpose registers would result in one of the following:

- some 32-bit instruction formats
- only eight general-purpose registers
- some other loss of instruction set functionality.

Finally, examining the implementation details of the M68000 family suggests that further performance gains follow from the register set division [6]. Each register set in the M68000 is associated with a separate arithmetic unit. The register division therefore permits parallel micro-operations to be performed on address and data registers.

Register Division Handling in the Hatfield Research Compiler

In the Hatfield compiler, the address registers and data registers were handled as separate groups and were reserved for different data types. Pointers, including all stack and display pointers, were assigned to address registers while all other types including integers were assigned to data registers. Array indices, for example, were computed exclusively in data registers. As a result of this rigid low-level typing there was never a requirement to move values between the address and data registers.

Conclusion

Ideally, 16 general-purpose registers would have been provided in the M68000. In practice, the use of separate address and data registers allows the instruction set to be tightly encoded in 16 bits, and thus avoids the performance degradation of multi-length instruction formats. Furthermore, since the division is based on a well-conceived concept of low-level typing, the division presents no significant problems for the compiler writer. In summary, the register division in the M68000 represents a reasonable design compromise.

Acknowledgements

Part of this work was supported by a SERC Postgraduate Research Studentship.

References

1. *M68000 16/32-bit Microprocessor Programmer's Reference Manual* Motorola Semiconductors, 4th Edition, 1984
2. Wulf W A *Compilers and Computer Architecture* IEEE Computer, Vol 14, No 7, July 1981, pp 41-47
3. Tanenbaum A S *Structured Computer Organisation* (3rd edition) Prentice-Hall, Englewood, N.J.,1990.
4. Williams F L *The Impact of Instruction Set Orthogonality and Complexity on Compiler Code Generation* PhD Thesis, Hatfield Polytechnic, UK. August 1989
5. Wirth N *Programming in Modula-2* 3rd Edition, Springer Verlag, 1985
6. Tredennick N *Implementation Decisions for the MC68000 Microprocessor* Proc. 3rd Rocky Mountain Symposium, Microcomputers: Systems, Software, Architecture, August 1979, pp 30-35

TABLE 1 Addressing Mode Restrictions on the M68000

	An	Dn	PC Relative	(An)+ -(An)	Imm
1) ALL					
ADD <ea>,Dn|MOVE <ea>,Dn	-	-	-	-	-
SUB <ea>,Dn|CMP <ea>,Dn					
ADDA <ea>,An|SUBA <ea>,An					
CMPA <ea>,An|MOVEA <ea>,An					
2) DATA					
AND <ea>,Dn|OR <ea>,Dn	x	-	-	-	-
CHK, DIVS, DIVU, MULS, MULU					
3) DATA ALTERABLE					
ADDI, SUBI, CMPI	R	-	x	-	M
ANDI, EORI, ORI	x	-	x	-	M
CLR, NEG, NOT, TST	x	-	x	-	M
NBCD, NEGX	x	-	x	-	M
BCHG, BCLR, BSET, BTST	x	-	x	-	M
MOVE Dn,<ea>	R	-	x	-	M
EOR Dn,<ea>	x	-	x	-	M
4) ALTERABLE					
ADDQ,SUBQ	-	-	x	-	M
5) MEMORY ALTERABLE					
ADD Dn,<ea>|SUB Dn,<ea>	R	R	x	-	M
AND Dn,<ea>|OR Dn,<ea>	x	R	x	-	M
SHIFTS	x	R	x	-	M
6) CONTROL					
JMP, JSR	M	M	-	M	M
PEA, LEA	M	M	-	M	M

Key

R : Removes redundant format

M : Meaningless operation excluded

- : Not excluded

x : Potentially useful operation lost