Analysis of the M6809 instruction set

by JOEL BONEY
Motorola, Inc.
Austin, Texas

ABSTRACT

The M6809 has now been in the marketplace for about 3 years and is one of the most popular midrange microcomputers. With 3 years of history, it is now possible to analyze many of the existing M6809 programs to see how the computer is actually used.

This paper includes data I took regarding instruction-set and addessing-mode usage on existing M6809 programs.[1] The specific information should be of interest to M6809 programmers and to future computer architects who wish to create similar machines. Beyond the specific M6809 information, however, there are some basic usage trends that are apparent in almost all Von Neuman architectures. Therefore, the information in this paper will be of interest to most users of microprocessors.

The data point out to programmers and system engineers what attributes of a computer's instruction set really affect the memory efficiency and throughput and what attributes don't matter. With this knowledge the programmer/system engineer should be better able to evaluate a microcomputer before he selects one for his project.

From the collection of the Computer History Museum (www.computerhistory.org)

From the collection of the Computer History Museum (www.computerhistory.org)

INTRODUCTION

In the spring of 1977 several of us at Motorola felt it was time to plan the follow-on part to the successful M6800 microprocessor. We were not the first to envision such a part, but we were the first to actually have the time and resources to proceed with the design. The new part was labeled the M6809. Terry Ritter and I were assigned the task of defining the new architecture.

As part of the preliminary design of the M6809 we did an analysis of the then existing programs written for the M6800. Much of the data we gathered from this analysis was very helpful in the design of the M6809.

Several years have now passed since the introduction of the M6809, and I felt it was time to analyze how the M6809 is actually being used. I hope these data will be as useful to the computer architects that follow us as the M6800 data were to us. Further, I hope these data will enable programmers and system engineers to be more intelligent in their selection of microcomputers for their applications.

GOALS AND CONSTRAINTS OF THE M6809 PROJECT

Every design project in industry begins with some goals that are shared by the designers, the marketers and, hopefully, by the customer. Every project also has some design constraints that must be adhered to in order to design a product that is producible. To understand the analysis of the M6809 that follows, it is necessary to have some understanding of the goals and constraints of the M6809 design project.

A personal goal held by both my coarchitect Terry Ritter and me for the M6809 project was that we wanted to prove that it was possible to produce an inexpensive microprocessor that was also easy to program. We felt that too many of the existing microprocessors were needlessly difficult to program. We suspected that the reason was not that it was impossible to make a microcomputer that was easy to program, but, rather, that the architects of the early microprocessors were generally more hardware oriented than software oriented.

Our experience told us that the consistency (regularity or orthogonality) of the instruction set was one of the features of a computer that made it easy to program. We wanted all the instructions, addressing modes, and system resources, such as registers, to be treated consistently. Analysis of the M6800 showed that instructions such as add *B* to *A* were rarely used despite the fact that they provided a useful function with better than average performance. The reason they were not used was that these instructions were unusual; they behaved differently than other instructions. It was our observation that programmers will not use instructions that are hard to use or that require the programmer to remember peculiarities about their execution.

The second goal of the design team was to support the improvements we saw rapidly taking place in the design of microprocessor software. We wanted the architecture to efficiently support modern block-structured high-level languages. Features such as stack addressing were included for this purpose. We also wanted to better support assembly language with the ability to write recursive, reentrant programs and position-independent programs.

Another goal was to improve significantly the performance of the M6809 as compared to that of the M6800.

Along with goals must come some constraints. We felt that the M6809 must be compatible with the M6800/M6801 at either the assembler-source or machine-code level. The machine-code level was preferable. Because it was impossible, however, to get the necessary throughput improvements and remain machine-code compatible, we selected assembler source-code compatibility.

BRIEF OVERVIEW OF THE M6809 ARCHITECTURE

To understand the data analysis that follows, it will be helpful to have a working knowledge of the M6809 architecture. The following sections describe the programmer's model, the instruction classes, and the addressing modes of the M6809.

Programmer's Model

The M6809 is a $1\frac{1}{2}$-address, 8-bit Von Neuman architecture microcomputer. Figure 1 is the programmer's model of the M6809.

The A and B accumulators are general purpose 8-bit accumulators that can be considered as one 16-bit accumulator for 16-bit operations. When used as one 16-bit accumulator they are called the D accumulator.

The X and Y index registers are general-purpose index registers used in the various forms of indexed addressing. The U and S registers are also index registers, but they have the additional quality that they can be used as stack pointers. The U register is called the user stack pointer. The S register is the hardware stack pointer and is also used by the hardware to store machine state during subroutine calls and interrupts.

The program counter on the M6809 is 16 bits wide, thus supporting an address space of 65, 536 bytes. All addresses on the M6809 are 16 bits wide.

The address field of an instruction with direct addressing on the M6809 is only 8 bits wide. The direct page register is a base

From the collection of the Computer History Museum (www.computerhistory.org)

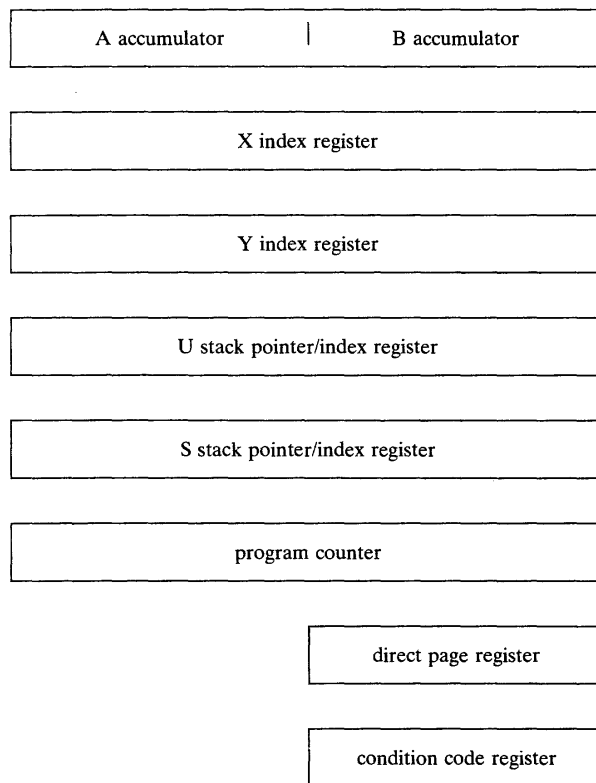

Figure 1—M6809 programmer's model

register that provides the most significant 8 bits of address for direct addressing. The condition code register contains the results from the last arithmetic or logical operation as well as interrupt masks and other control bits.

Instruction Classes

The 6809 has the following seven major classes of instructions:

1. Arithmetic, logical, load and store
2. Read / modify / write
3. Conditional branch
4. Load effective address
5. Push / pull
6. Control transfer
7. Miscellaneous

The arithmetic, logical, and load and store instructions make up the largest set of instructions. They are $1\frac{1}{2}$ address instructions that get one of their operands from memory and the other from an accumulator, and they store the result, if any, in the accumulator.

The read/modify/write instructions read a memory location or accumulator, perform some operation on its contents (e.g., clear, shift, increment), and store the result back to the same memory location or accumulator.

The conditional branch instructions are used for conditional program control transfer.

The load-effective-address instructions evaluate the effective address of an indexed addressing-mode instruction and return the effective address to an index register. This makes the powerful address-calculation hardware already present for indexed addressing available for address manipulation.

The push/pull instructions allow one or several of the registers to be pushed or pulled on the stacks pointed to by the U or S stack pointers. A single push or pull instruction can push or pull from 1 to 8 registers.

The control transfer instructions include the subroutine calls as well as the unconditional jumps and branches. The miscellaneous category includes instructions such as sign extend, no-operation, and transfer register to register. Their addressing mode, if any, is inherent.

Addressing Modes

The M6809 supports a variety of addressing modes. There are seven major types with several subtypes:

1. Inherent
2. Accumulator
3. Register
4. Immediate
5. Absolute
   - Extended
   - Direct
6. Relative
   - Long
   - Short
7. Indexed
   - Constant offset
   - Constant offset from the PC
   - Accumulator offset
   - Auto increment / decrement

Inherent addressing includes those instructions that have no addressing options. Accumulator addressing is similar to inherent except that an accumulator is specified (e.g., *CLRA*, *CLRB*). Some M6809 instructions specify one or several of the registers as the operands (e.g., TFR D,X—transfer D to X). This is called register addressing. In immediate addressing the source operand is assumed to be in the memory location immediately following the current opcode. The M6809 supports both 8-bit and 16-bit immediate values.

In absolute addressing all or part of the absolute memory address is included in the instruction. In extended addressing the full 16-bit address is included in the instruction. In direct addressing only the lower 8 bits of the address are included in the instruction. The upper 8 bits of the address are supplied by the direct page register.

Relative addressing is used for branches. There are both 8-bit and 16-bit relative offsets.

Many of the new features supported by the M6809 lie in its greatly expanded indexed addressing modes. In the constant offset indexed addressing modes a constant value of length 0,

From the collection of the Computer History Museum (www.computerhistory.org)

5, 8, or 16 bits is summed with an index register to obtain the effective address used to fetch the operand.

Constant offset from the program counter (program counter relative) works in much the same way except the program counter is used as the index register. This addressing mode is used most often by the load-effective-address instruction to find the starting address of tables in a position-independent program.

In accumulator offset mode the effective address is the sum of the signed accumulator and the specified index register. The original contents of the index register and the accumulator are unchanged by this addressing mode.

In auto increment mode the contents of the specified index register are used as the effective address; then they are incremented by 1 or 2 (postincrement). In auto decrement the contents of the index register specified are first decremented by 1 or 2 and then used as the effective address (predecrement). In both cases the contents of the index register are permanently changed.

All the indexed addressing modes and the extended addressing mode of the M6809 provide for an additional level of indirection. That is, the original effective address calculated by the addressing mode can be used as the address of another 16-bit value that specifies the final effective address.

STATIC VERSUS DYNAMIC ANALYSIS OF ARCHITECTURES

There are essentially two types of analyses that can be performed on an instruction set—static and dynamic. In static analysis, either the source code or the object code of a program or programs is analyzed to determine the frequency of *appearance* of various instructions, addressing modes, registers, and so on. In dynamic analysis, data are taken during the actual execution of a program and are used to determine the frequency of *execution* of an instruction, addressing mode, and so on.

Both types of data are useful for specific purposes. The static data can lead to improvements in future architectures that will reduce the size of the average program. The dynamic data can lead to a reduction in the execution time of the average program. These two improvements are also somewhat related. If a program is smaller, it generally has to fetch fewer bytes of opcode and, hence, runs faster.

STATIC ANALYSIS OF THE M6809

Instruction Classes

To make the data as useful as possible, I analyzed static code from several different classes of programs. I tried to balance the amount of code in each class so that one class of program would not bias the data. I classified the programs into the following classes:

Program Class	*Number of Bytes*
Compiler-generated code	14549
Compiler code	7695
Application code	26305
Monitor code	6293
Numeric code	7135
	61977

Average Instruction Size

One parameter of interest is the average size of an M6809 instruction. The data can be useful when estimating the memory needed for an application. The size of the average instruction for the various program classes and for all classes combined is given in the list that follows.

Class	*Average Size*
Numeric	2.16 bytes
Monitor	2.27 bytes
Compiler	2.30 bytes
Application	2.40 bytes
Compiled	2.43 bytes
All	2.35 bytes

If an M6809 programmer can estimate the approximate number of source lines he will write, he can use 2.35 to estimate his total memory requirements.

Most Frequently Appearing Single Opcodes

There are two ways of looking at the static data. One is to count the percentage of times an instruction (opcode plus additional addressing bytes) appears versus the total number of instructions. I call this the percentage by count. The other is to count the percent of the total bytes actually taken by an instruction. I call this the percentage by bytes.

Table I is the data from the 10 most frequently appearing single opcodes in the concatenation of all of the static data. It is interesting to note that the top 10 opcodes represent 37.4% of all instructions. Since there are 266 possible opcodes in the M6809, these 10 opcodes are only 3.76% of all possible op-

TABLE I—Top 10 most frequently appearing M6809 opcodes

Opcode	Instr.	By Count	%	By Bytes	%
17	lbsr	2307	8.76	6921	11.17
30	leax	922	3.50	2653	4.28
34	pshs	910	3.46	1820	2.94
86	lda immed.	906	3.44	1812	2.92
20	bra	877	3.33	1754	2.83
8e	ldx immed.	862	3.27	2586	4.17
26	bne	804	3.05	1608	2.59
27	beq	800	3.04	1600	2.58
ed	std indexed	739	2.81	1584	2.56
cc	ldd immed.	722	2.74	2166	3.49
	Total		37.40		39.53

From the collection of the Computer History Museum (www.computerhistory.org)

codes. Although this may be a surprise to those readers who have never seen instruction-usage data before, it is consistent with most modern architectures.

The top three opcodes are new M6809 instructions that were not available on the M6800. They account for 15.72% of all opcodes. Clearly there was a need for these new instructions.

Most Frequently Appearing Opcodes by Class

Although it is useful to know which individual opcodes occur most frequently, it is more useful to have the data broken down into slightly larger classes. Table II contains the top 10 classes sorted by count for the concatenation of all the static data.

The first 6 classes account for over 52% of all instructions and the top 10 for 66.69%. We can conclude that the M6809 behaves like most computers in that a very few instruction types account for most of the instructions.[2] Furthermore, most of the instructions are in the load-store category. (Pushes and pulls are also classified as loads and stores in some literature.)

Most Frequently Appearing Instructions by Large Class

We can take an even larger view of the instruction classes. These data are useful for comparing the usage of the M6809 to other computers. Table III contains these data along with static data gathered by Leonard Shustek for the IBM 370 and PDP-11.[3]

From the data in Table III we can deduce that the M6809 is not much different from other Von Neuman machines. All three machines have a high percentage of loads and stores, subroutine calls, conditional branches, and compares/tests. Furthermore, the amount of arithmetic and logical instructions is low.

What this tells the system designer who is trying to evaluate the memory efficiency of a microcomputer (remember we are dealing with static data here) is that the overall size of the program will be determined by a few instruction classes. Also, since the loads and stores are heavy users of the addressing modes, the byte efficiency of the addressing modes will greatly effect the efficiency of the architecture.

TABLE II—Top 10 classes of M6809 instructions

Class	Count	%	Bytes	%
16-bit loads	4114	15.62	11291	18.22
8-bit loads	2868	10.89	6144	9.91
Long branch subr.	2307	8.76	6921	11.17
Load eff. addr.	1708	6.49	4629	7.47
Push	1426	5.42	2852	4.60
Store 16-bits	1376	5.23	3155	5.09
Store 8-bits	1219	4.63	2991	4.83
Branch always	877	3.33	1754	2.83
Compare	860	3.27	1792	2.89
Branch not equal	804	3.05	1608	2.59
Total		66.69		69.60

TABLE III—Comparison of static data

Class	M6809	IBM 370	PDP-11
Load/store	50.83	48.00	32.80
Call	12.49	5.50	6.30
Cond branch	10.07	15.30	20.10
Control transfer	5.26	?	†
Cmp/tst	5.63	8.80	6.50
Arith/logical	4.04	3.50*	3.00‡
Other	11.68	18.90	26.30

*Subtract only.
†Control transfer included in conditional branch.
‡Add only.

Static Appearance of Addressing Modes

Another major area of interest is the use of the addressing modes of a computer. Table IV shows the addressing mode statistics for the concatenation of all the static data.

Indexed addressing is by far the most frequently appearing addressing mode because many of the unique features of the M6809 addressing modes are hidden under the umbrella of indexed addressing. For this reason indexed addressing is discussed in more detail in a later paragraph.

The relative addressing modes (short and long) account for 25.01% of all addressing modes. This indicates that M6809 programmers are using the relative rather than the absolute control transfers and subroutine calls. In short, programmers are writing a lot of position-independent codes for the M6809. The relatively small amounts of extended and direct absolute addressing also back up this conclusion.

Indexed addressing static statistics

Since indexed addressing represents about 72% of all the addressing modes that reference memory (direct, extended, and indexed), we now spend some time looking at the indexed addressing data. Table V breaks the indexed addressing down into its subgroups. The basic subgroups are the no offset, the

TABLE IV—M6809 static addressing mode usage

Addressing Mode	Count	%
Indexed	7371	27.99
Immediate	5132	19.49
Short relative	3532	13.41
Inherent	3466	13.16
Long relative	3054	11.60
Extended	1937	7.36
Direct	958	3.64
Accumulator b	456	1.73
Accumulator a	424	1.61
Indirect	175	0.66

From the collection of the Computer History Museum (www.computerhistory.org)

Table V—Static indexed addressing data

Addressing Mode	Number	% of total
No offset (offset = 0)	961	13.04
5-bit offset	3940	53.45
8-bit offset	631	8.56
16-bit offset	572	7.76
8-bit offset from PC	92	1.25
16-bit offset from PC	139	1.89
a accumulator offset	85	1.15
b accumulator offset	99	1.34
d accumulator offset	113	1.53
Auto increment by 1	286	3.88
Auto increment by 2	120	1.63
Auto decrement by 1	43	0.58
Auto decrement by 2	273	3.70
Extended indirect	17	0.23
Average additional bytes for indexed =		1.17

constant offset, the register offset, the auto increment/decrement, and extended indirect.

The constant offset varieties account for 72.91% of the total. If no offset is included with the constant offset subgroup, we find that 85.95% of the indexed instructions are of a simple type. The program that took the data also calculated the average number of bytes that are added for each indexed addressing mode above the base opcode. The average is 1.17 bytes. Since the minimum possible is 1.0 bytes, this is a very encouraging statistic. The code-size penalty for providing all the new M6809 indexed addressing modes is minimal. This is good news. As stated previously, the memory efficiency of the loads and stores and the addressing modes has the greatest influence on the memory efficiency of the whole architecture.

DYNAMIC ANALYSIS OF THE M6809

Although the static data used in the previous sections are useful in predicting the *size* of a program, data taken while a program is actually executing are more useful in determining the *throughput* of a computer. Unfortunately, reliable dynamic data are much harder to obtain than are static data; hence, there are few dynamic data for microprocessors.

There are two basic ways of collecting dynamic data. One is to build special high-speed hardware to monitor the instruction execution of a computer. This method has the advantage of being real time but has the disadvantage of being very expensive. To reduce hardware costs, it is usually necessary for the hardware to take only snapshots of a fixed number of cycles. It is hoped that these snapshots will faithfully represent the execution characteristics of the whole program.

The second method is to run programs using a simulator. The simulator can be instrumented to collect the data cycle by cycle. The problem with a simulator is that it slows down the program's execution so much that the statistics may become warped. This is especially true for real-time programs. In fact, many real-time programs won't run on a simulator at all. Furthermore, the simulator may have problems simulating the I/O and interrupt portions of the programs.

I used the simulator method for collecting the data presented in the next sections.

Program Mix

Because of the limitations just mentioned, I was able to analyze only five programs dynamically. The are shown in the list that follows.

Program	*Description*	*Source Language*
Chess	Chess playing program	Assembler
Ed	Line editor	Assembler
Mon	Small monitor	C
Mopet	Automatic test generator	Pascal, Assembler
M6839	Floating point package	Structured Assembler

Although I would like to have analyzed more programs, I feel these programs are representative. However, I did not feel justified in concatenating all the data into one data set as I did with the static data. In the following sections the dynamic data are presented independently for each simulated program.

The number of instructions and cycles in the simulation for each program are the following:

Program	*Instructions*	*Cycles*
Mopet	451,131	2,015,244
Chess	385,698	1,797,514
M6839	163,370	719,976
Ed	149,521	702,876
Mon	86,406	477,887
Total	1,236,126	5,713,497

Cycles Per Instruction and MIPS

A metric of interest for a processor is the number of cycles taken by the average instruction and the millions of instructions per second (MIPS) for each program. These data are contained in the list that follows.

Program	*Avg. Cycles/Instr.*	*MIPS at 2 MHz*
M6839	4.41	.454
Mopet	4.47	.447
Chess	4.66	.429
Ed	4.70	.425
Mon	5.53	.362
Average	4.75	.423

The data are fairly consistent, and a MIPS rate of approximately .4 can be used by a programmer to successfully estimate the execution speed of most M6809 programs.

A note on MIPS is in order here. The actual amount of work (throughput) done by a computer is a function of both

From the collection of the Computer History Museum (www.computerhistory.org)

the MIPS and the size or amount of data actually operated on during each instruction. For example, a 1-MIP 32-bit machine will have about four times the throughput of a 1-MIP 8-bit machine. The M6809 is somewhere between an 8-bit machine and a 16-bit machine.

Most Frequently Executed Opcodes by Class

Using the same classes of opcodes as described in the static data, we can determine what classes of instructions are most frequently executed. Table VI is the union of the top 10 classes for each program. Note that it takes 24 separate classes to get the union of the top 10 classes. This indicates that the dynamic data are not as consistent as the static data, where the union of the top 10 would only include 14 separate classes.

Most Frequently Executed By Large Class

Table VII contains the union of the three largest classes for each program. In both static and dynamic frequency, loads and stores make up the largest class of instructions by far. Next in frequency in the dynamic data are the conditional branches. Compares and tests also have a high dynamic frequency. Calls have a high frequency, but not as high as in static. Probably the most surprising result is that in programs that have a lot of shifts to begin with (statically), the dynamic frequency of the shifts is even higher.

Table VI—Union of the top ten classes (dynamic) showing % of executed instructions

Class	Chess	Editor	Monitor	Mopet	M6839
Load 8-bit	19.20	11.87	10.10	8.04	8.76
Branch if equal	9.93	6.56	1.44	12.07	.59
Load 16-bit	7.81	8.80	10.41	7.35	2.05
Load eff. addr.	5.06	4.82	3.72	10.95	2.92
Store 8-bit	4.85	5.01	1.49	2.19	4.22
Store 16-bit	4.23	3.58	3.29	4.26	1.88
Branch if not =	4.21	9.32	2.68	6.99	3.80
Bit test	3.49	0	.59	.32	.01
Branch if minus	3.22	.06	.06	.04	.03
Increment	2.89	.46	1.48	.32	.92
Compare 8-bit	2.74	13.18	6.56	8.68	9.35
Jump to subr.	2.64	6.45	3.29	2.60	.05
Push	2.19	3.37	4.20	1.45	1.55
Return from subr.	1.97	3.04	4.90	2.48	2.30
Pull	1.84	2.49	3.90	.77	.95
Compare 16-bit	1.40	5.62	1.39	11.63	1.25
Decrement	1.21	0	.08	.44	4.97
Branch always	.95	6.47	.64	4.64	6.20
Branch less than	.19	.09	0	.01	5.62
Branch higher/same	.11	.27	.30	3.10	1.68
Branch lower	.07	.41	3.56	1.68	.27
Long branch subr.	.04	.73	5.15	1.78	1.15
Rotate left	.02	.08	.06	.09	10.96
Rotate right	0	0	3.77	0	10.63
Other	19.74	7.32	26.94	8.12	16.01

Table VII—Union of the top three largest classes (dynamic)

Class	Chess	Editor	Monitor	Mopet	M6839
Load	27.01	20.67	20.51	15.40	10.81
Cond. branch	21.81	21.12	9.24	24.76	15.86
Store	9.08	8.59	4.78	6.46	6.10
Cmp/test	6.07	18.97	8.32	22.14	12.63
Call	3.18	7.18	10.66	5.54	2.97
Shifts	0.33	0.16	9.50	0.45	23.33

To a user trying to select a microprocessor, these data indicate that the speed of the loads, stores, and branches will have a very large impact on the throughput. The speed of the addressing modes will directly affect the speed of the loads and stores. Furthermore, the speed of the arithmetic (except compare) and logical instructions is almost irrelevant to throughput.

Dynamic Execution of Addressing Modes

This section presents the dynamic addressing-mode data collected by the simulator. Table VIII contains the frequency of execution of the various addressing modes for the five programs.

In all five programs indexed addressing is by far the most executed addressing mode. In fact, its dynamic frequency is about 10% higher than its static frequency for the same programs. Short relative addressing is a strong second with immediate addressing third. If a future architect were looking to improve the performance of the M6809, it would be advantageous to look at speeding up relative and indexed addressing. Indirect addressing is rarely used.

Indexed Addressing Dynamic Statistics

Since the frequency of indexed addressing is so high, it is worthwhile to see how indexed addressing is being used dynamically. Table IX contains the indexed addressing breakdown for the five programs analyzed. The 5-bit and no-offset indexed addressing are by far the most frequently executed. If it were possible to make these faster, it would certainly im-

Table VIII—Dynamic addressing mode usage

Mode	Chess	Editor	Monitor	Mopet	M6839
Indexed	40.79	33.74	29.76	31.05	41.46
Short relative	22.23	27.48	12.09	30.46	23.78
Immediate	14.23	18.47	15.46	12.12	11.49
Inherent	7.21	9.77	23.19	5.77	6.73
Extended	3.90	8.93	3.29	2.90	0.24
Direct	8.33	0.00	0.00	14.29	0.00
Long relative	1.08	0.88	5.18	1.98	1.27
Accumulator a	1.51	0.09	10.33	0.84	9.07
Accumulator b	0.72	0.64	0.70	0.59	5.95
Indirect	0.69	0.00	0.00	0.04	0.15

From the collection of the Computer History Museum (www.computerhistory.org)

Table IX—Dynamic indexed addressing statistics

Addr Mode	Chess	Editor	Monitor	Mopet	M6839
No offset	17.03	23.34	7.47	45.95	10.96
5-bit offset	62.00	39.80	37.82	37.35	62.32
8-bit offset	2.30	0.01	0.00	0.22	2.55
16-bit offset	6.77	0.01	0.00	1.21	0.00
8-bit off. on PC	0.00	0.06	11.21	0.02	0.14
16-bit off. on PC	0.00	0.02	0.00	0.23	1.53
Auto incr. by 1	2.70	34.76	21.26	6.78	1.68
Auto incr. by 2	1.57	1.46	0.00	3.06	0.06
Auto decr. by 1	1.15	0.02	0.05	4.26	0.19
Auto decr. by 2	1.60	0.00	0.00	0.00	0.38
a acc. offset	3.34	0.08	11.06	0.37	3.91
b acc. offset	0.35	0.00	0.00	0.26	15.93
d acc. offset	0.42	0.43	11.13	0.27	0.35
Extended indirect	0.77	0.00	0.00	0.00	0.00

prove the M6809's performance. Auto increment by 1 is used fairly often in an executing program. This is expected since almost all auto increments and decrements are in loops.

SUMMARY

As is the case with most Von Neuman architectures, only a few single opcodes make up a large percentage of all the instructions that appear statically. For the M6809, the top 20 opcodes accounted for over 58% of all the instructions. Three new M6809 instructions headed up the list of the most frequently appearing single opcodes. They were long branch to subroutine, load effective address, and push on the S stack. The rest of the top 20 was composed of loads, stores, branches, compares, subroutine calls, and subroutine returns.

In larger classes of instructions, the following statistics were the approximate static values for the top five classes:

1. Loads and stores = 36%
2. Subroutine calls = 12%
3. Conditional branches = 10%
4. Pushes and pulls = 8%
5. Load effective address = 6%

The arithmetic and logical instructions occurred infrequently.

Thus, the loads, stores, subroutine calls and conditional branches are a better metric of memory efficiency than are the arithmetic and logical instructions.

The static addressing-mode data indicate that the most common addressing mode is indexed (30%), followed by relative (24%), and by immediate (20%). The number of direct and extended instructions combined was only 10%. Indirect was practically never used.

In the static indexed addressing data we find that 5-bit and no-offset indexed account for 66% of all indexed instructions. Including the 8-bit, 16-bit, 8-bit program counter relative mode and the 16-bit program counter relative mode, we find 86% of the indexed instructions are constant offset or have no offset.

This indicates to future architects that having several efficient, simple offset indexed forms is beneficial. The M6809 has six such forms as compared to the M6800's one.

The average number of bytes added for each indexed instruction is 1.17 bytes.

There was a larger variation in the dynamic data. There are two reasons for this. One is that I had fewer dynamic data points. The other is that dynamic data seem to be more dependent on the application and programmer style.

The average number of cycles for an M6809 instruction is approximately 4.75. This gives a throughput of .423 MIPS with a 2-MHz M6809.

By classes, the conditional branches all combined to form the second most-executed group, second only to the loads and stores. The other large classes were compare and test, the calls, and the shifts.

In dynamic execution the indexed addressing mode accounts for approximately 35% of all addressing modes. Short relative is about 25%, and immediate is 15%. Long relative addressing usage is fairly low. Indirect is the big loser again.

In indexed addressing, the offset varieties accounted for 72%. This is down from the static data, but is still impressive. Auto increment and the accumulator offsets make up most of the rest of the indexed data.

The dynamic data reinforce the conclusions from the static analysis that a good measure of the overall efficiency of an architecture is best found in the loads, stores, conditional branches, subroutine calls, and addressing modes. This coincides with the modern view that computers spend most of their time in moving data around and making decisions based on the data rather than in number crunching.

REFERENCES

1. Boney, Joel. "Analysis of the M6809 Instruction Set," Report, University of Texas Computer Science Department.
2. Stone, Harold S. (ed.). *Introduction to Computer Architecture.* Science Research Associates, 1975, pp. 525–528.
3. Shustek, Leonard J. "Analysis and Performance of Computer Instruction Sets," Stanford Linear Accelerator Center Report No. 205, STAN-CS-78-658, January 1978.

From the collection of the Computer History Museum (www.computerhistory.org)

From the collection of the Computer History Museum (www.computerhistory.org)