SLAC PUB-1874
DECEMBER 1976

CURRENT ISSUES IN THE ARCHITECTURE OF MICROPROCESSORS*

Bernard L. Peuto+
Zilog, Inc.

and

Leonard J. Shustek
Stanford Linear Accelerator Center
and
Computer Science Department
Stanford University

* Work supported in part by the Energy Research and Development Administration under contract E(043)515.

+ Work done while a Visiting Scientist at the Stanford Linear Accelerator Center.

To Appear: COMPUTER (IEEE) February, 1977

INTRODUCTION

Despite the fact that microcomputers have existed commercially for only five years, microcomputer architecture is not an entirely new field. It is, rather, the application of the general principles of computer architecture to microcomputers. In "Planning a Computer System", Frederick P. Brooks Jr. [1] defines computer architecture as being, like other architecture, "the art of determining the needs of the user of a structure and then designing to meet those needs as effectively as possible within economic and technological constraints".

In many ways the design and use of microprocessors is identical to that of larger machines. In both cases there are very high initial engineering and development costs which have little influence on the final price of the product. In both cases the products which are most successful are those which are general purpose. Perhaps the most important similarity involves the issue of software; any other small differences between large and small machine architectures are outweighed by the similar high cost in program development.

Rather than expand on the similarities, this paper will concentrate on the peculiar constraints which are unique to current MOS microprocessors. The differences which confront

the architect are not sufficient to require a different approach to the design, but they certainly influence the details of the compromises made. We will first describe the technological constraints and the special plight of a new industry with a recently acquired knowledge of computers. Then, using examples from recent microprocessors, we will examine some of the architectural features which have resulted. Finally, we will illustrate our belief in the unity of computer architecture by showing that the results of instruction set measurements for a typical microprocessor are quite similar to those for other established computers.

TECHNOLOGICAL CONSTRAINTS

The overriding technical constaint on microprocessor design is the size of the silicon die. Using an N-channel MOS process, a die equivalent to a square 200 mils on a side produced on a 3-inch wafer can be manufactured with acceptable yield (fraction of operating devices per wafer), but under these conditions a 10% increase in die size can result in as much as a 50% decrease in the the yield. Most of the progress in manufacturing has come from better control of defects and increase in the wafer size, which results in higher total yield. Advances in manufacturing can therefore either be applied to decreasing manufacturing cost for a given size die, or to increasing complexity by increasing the die size. The architect, to be successful,

must develop ways to relate potential architectural features to their space requirements on a silicon die.

The limits imposed by the die size can also be circumvented by changes in the details of the process (P-channel vs N-channel, depletion load devices, etc) which affect the gate density as well as the logic speed. For example the ZILOG Z80, on an N-channel depletion load 185 mil square die, has more than three times as many components as the older INTEL 4004, on a P-channel 136 mil square die, and the Z80 has a gate delay which is three times less than the 4004 [2].

Unique to the microprocessor field is the requirement to produce the device in a dual-in-line package with a small number of pins. This seemingly trivial issue has an enormous effect on the entire structure of the microcomputer system. In very large quantities the manufacturing cost of the microprocessor is dominated by the cost of the package, and users as well as package manufacturers have investments in standard package sizes which make it difficult to freely choose an appropriate package for the microprocessor. The 18-pin package for the INTEL 8008, which obliged the designer to use time-multiplexed busses, was primarily due to the lack of a larger standard package. As the INTEL 8080 was being designed, the 40-pin package then available allowed more freedom. In addition to pin count, the heat dissipation of the device has an influence on the choice of

package material; the easily manufactured plastic packages cannot dissipate more than 700 milliwatts. It is only with the success of N-channel depletion load logic that microprocessors can use the plastic packages, which can be up to 5 times less expensive than ceramic.

The MOS microcomputers have so far been noticeably slower than their minicomputer counterparts. The current process allows clock frequencies of 2.5 to 3 Mhz for standard parts, although part selection allows sufficient yield for parts as fast as 4 Mhz. This corresponds to an average instruction execution time of approximately 2 to 3 microseconds. The slow speed is not due to fundamental limits of MOS technology, but to the limited amount of power which can be dissipated in commercially available packages. The design has to trade-off power dissipation for speed and complexity (number of gates). MOS technologies with improved speed-power products are being developed today and one can expect that MOS clock rates as high as 6 MHz will be reached in a few years .

THE KNOWLEDGE GAP

The first microcomputers were designed by the memory component designers, whose ability to do so was a result of breakthroughs in acheiving greater memory density. Their primary expertise was in circuit integration rather than computer architecture, but the limitations at the time were

so severe that only the simplest machines could have been designed. The current state of the art is such that most of the features of larger machines could be incorporated in microprocessors within the next few years, and the industry has recognized that it is necessary to use architects with previous experience in medium-to-large computers. In both hardware and software these designers must learn some of the aspects of the component industry, but probably far less than the component designers would have had to learn about computer architecture.

The architecture of a new computer is driven by the market chosen for it, and by the requirements of the users in that market. The dilemma for the microprocessor architect is that the applications are not yet defined; they are often created by the existence and price of the product. More microprocessor have been produced since 1972 than all other computers combined, but the growth rate is such that this number (currently 1 to 2 million/year) is small compared even to conservative estimates of future use. Most of those future applications are unknown. It is even hard to characterize the current use of microprocessors because of the insufficient feedback from users to manufacturers. This complicates the architect's task, since the processor must be well matched to the intended software environment, about which little is known for large systems and even less for microprocessors. One of the most pressing needs in

microcomputer architecture is the measurement of the characteristics of user software.

As for large computers, software development often represents the largest investment in the production of a computer system. Unlike the users of large systems, the microcomputer programmer often has no prior experience in programming or software system design. It is often the digital engineers who must write the programs for the application, and although their sophistication is growing, they still must be considered novices. As a consequence, the tools being used are close to the hardware that the users are familiar with and thus resemble the software tools of the 1950's [3]. In this context, where the users are introduced to computer concepts via the hardware interface, assembly language programming requires the least additional knowledge compared to that required for higher-level languages. Fortunately there is a large body of experience with larger machines that can be directly applied to microprocessors once the users have recognized the advantages. This has promoted a large influx of software engineers from large machine environments and it has, for example, provided a new boom in the compiler-writing industry.

CURRENT ARCHITECTURES

The focus of a discussion of current architectures

should be on the top-of-the-line processors like the INTEL 8080, MOTOROLA 6800, TI 9900, and ZILOG Z80. The user will find, as he did for TTL design, that it is better to use only part of a complex device than to build what is required from more primitive components [4]. A comparison with older processors, such as the INTEL 4004, would be unfair because many of the technological constraints have since been relaxed, and because the designers now have a better understanding of the market needs.

Many microprocessors have some of the desirable features which seem appropriate for the intended applications, but none of them has a complete and consistent set. This fuels battles between manufacturers based on architectural features which are hard to evaluate and compare, and indeed, the tables which often accompany comparisons of microprocessors are quite misleading because of the simplifications which they must make. We illustrate this point by considering the wordsize, number of registers, addressing modes, and I/O structure of microcomputers.

Following in the footsteps of the minicomputers of the middle sixties, the microcomputer users have attempted to group microcomputers based on wordsize. The wordsize of any computer, however, is hard to define. Does the 8080 have a 16-bit wordsize because there is an instruction which adds two 16-bit registers? Does the IBM 370 have a wordsize of 8 bits because the smallest addressable data element is a

byte? The width of the memory data path is probably a better measure, but almost all machines manipulate data elements which are both larger and smaller than the data path. Most users would agree that the 360/370 series has a 32-bit word, but some of the smaller implementations have only an 8-bit internal data path. The bias toward calling it a 32-bit computer comes from the large number and regularity of the instructions which manipulate 32-bit quantities. Because the microprocessors to date are very irregular in their instructions for data manipulation, it is even more difficult to characterize them by their wordsize.

It is even impossible to count the number of registers. The SIGNETICS 2650 has seven 8-bit registers, but only four can be addressed at a time. The 8080 has seven 8-bit registers, but only one is an 8-bit accumulator. The others can be grouped in pairs and represent 16-bit registers, but each of the pairs is distinguished in some way by operations which can be performed on it alone. Again this lack of regularity makes architectural classification very difficult.

Taken as a group, microprocessors implement probably most of the addressing modes invented in the last 20 years of computer architecture. Taken individually, many of them have particular omissions or quirks that diminish their power: only one index register (6800), certain instructions without full memory addresses (8080), or

difficulty in manipulating data large enough to contain a full memory address (many of them).

Regularity does not only affect comparative evaluation; it is also the key to programmability. Both human programmers and compilers are considerably hampered by an architecture with a myriad of special cases instead of a few general rules. It has been shown [5] that the fewest mistakes are made when programming with a language which can be described with a small set of rules.

In most existing computer systems and especially those with microprocessors, the CPU is a small part of the total cost because of the complexity of the I/O devices and their interfaces. To open new markets where the use of a microprocessor will result in more than a marginal reduction in system cost, there must be I/O and peripheral interfaces of similar integration and thus low cost. A wide variety of such intermediate peripheral chips are starting to become available, such as parallel I/O interfaces, USARTs for serial I/O, and DMA (Direct Memory Access) processors for higher-speed devices. These chips can be quite complex internally, but their communication with the CPU is through a simple bus with few control lines. The result is that families of chips are produced which share a common but primitive communications protocol with the CPU. As the market evolves, one such protocol may eventually be adopted as a de facto standard so that components of different

manufacturers could be used together. The existence of a standard will also allow individual components to be improved without requiring all members of the family to be redesigned.

MEASUREMENTS OF AN EXISTING MICROPROCESSOR

To better evaluate microprocessor architectures and implementations, it is useful to make measurements of current processors. The results of these measurements can be used to identify deficiencies and indicate where implementation efforts should be concentrated.

This section will describe such measurements made for the INTEL 8080, which represents the largest share of the 8-bit microprocessor market today. The measurements are those that display properties of the instruction set and its implementation. Static opcode frequencies are useful to examine how efficient the instruction encoding is with respect to program size. Dynamic opcode frequencies, combined with the execution times of the individual instructions, help determine which instructions account for most of the execution time. From this basic data, other useful information can be derived, such as memory utilization per instruction and average instruction length.

An 8080 simulator running on a large computer was modified to provide information about instruction execution.

The simulated 8080 was made to execute several different programs in an attempt to obtain results that are not biased by artifacts of a specific program. Enough information was collected to allow the instruction timing formulas to be evaluated. The methods used are a simplification of those that have been used to compare two high-performance computers in a previous study [6]. The static opcode distributions were obtained by directly processing the source files.

The programs analyzed included two basic interpreters (JBASIC and TINYBASIC), the software used in a sophisticated text and graphics computer terminal (VGT) [7], a realtime music generation program (MUSIC), and a text editor (EDITOR). The text editor was written in the high-level language PL/M [8]; all others were in assembly language. Excerpts from the results of these measurements appear in tables 1 to 7. The opcode mnemonics have been changed to a form which should be more understandable to readers not familiar with the 8080.

Opcode Distribution

It has been observed many times that very few opcodes account for most of a program's execution. In the VGT (see Table 1) only 7 instructions represents 53.7% of all instructions executed, and 39 represent 99.2%. The most common instruction (14.9%) is the conditional jump (JMP

CC,xxx), followed closely by the 8-bit register to register load (LOD R,R). In EDITOR, the same two instruction are important, but the rotate instruction (ROT) appears in second place because it is used in the inner loop of the multiply subroutine. For the VGT the rotate appears only as the 34th instruction, which indicates that some generally infrequent instructions are occasionally very important for specific program. The jump, load, push (PUSH RR), and pop (POP RR) instructions, however, appear to be universally important for all programs.

The opcodes which account for 50% of the instructions executed are not the same as those which account for 50% of the execution time. Instructions like the unconditional subroutine call (CALL U,xxx), which are lengthy because of the stack references to memory, are more significant in execution time than frequency (7.2% vs. 3.8% for VGT), but simple instructions like LOD take less time than their frequency would indicate (7.5% vs. 11.4%). The same effect exists even to a much larger degree in computers like the IBM 370 because for those machines [6], the ratio of the longest to shortest instruction execution time is much larger than it is for microcomputers (300 for the 370/168 versus 4.5 for the 8080).

Dynamic opcode pair frequencies often clearly reveal the dominant loop of an executing program. In TEASIC for example (table 2), a string search constructed from an index

register increment (INC HL), a character comparison (CMP (HL)), and a conditional jump (JMP CC,xxx) represents 9.3% of the program execution, and a slightly longer variation ending with the same sequence represents an additional 4.9%. This is valuable information for the microprocessor architect, who can thereby judge the importance of including single instructions for performing equivalent operations, such as the compare increment and repeat (CPIR) of the Zilog Z80.

The static opcode distribution (Table 3) is often quite different from the dynamic distribution. Although loads and jumps still predominate, lengthy but infrequently executed initialization code is represented by the presence of the load-immediate instructions (LODI R,n for 8-bit data, LODI RR,xxx for 16-bit data) in the top 50% group. The simple byte movement instructions are statically common, but the dynamically important stack push and pop instructions are not. The static opcode frequencies are important in choosing a space-efficient encoding for instructions.

The static opcode pair frequencies in table 4 reflect common code sequences. Some of these sequences, like a compare (CMPI n) or a test (IOR R) followed by a conditional jump, are reasonable and unsurprising. Many others are indications of common sequences that, from an architectural point of view, should be incorporated into a single instruction. Rotates which follow rotates show that

multiple-bit rotation is indeed a common operation; note that a rotate is followed by another rotate twelve times more frequently than would have been expected from a simple count of its occurrences. Pairs of 8-bit register to register loads are common because they are used to simulate the missing 16-bit register to register loads. The static pair distribution for the EDITOR reflects the simple code-generation schemes used by the PL/M compiler. Addressing is often done by loading an address into the HL register and referencing the variable in a subsequent register-indirect instruction since the available full-address instructions are limited.

Memory References and Instruction Speed

Statistics on instruction length, memory references per instruction and instruction speed are given in Table 5 for all of the programs examined. The 8080 has instructions ranging from a single byte to three bytes. The first byte is always the opcode, which includes register designators, and the following bytes are either immediate data or memory addresses. The average instruction length varies from 1.4 to 1.8 bytes.

The number of data references per instruction is useful because many of the microprocessors do not overlap memory accesses with instruction execution, so that operand accesses must be directly added to the instruction execution

time. Compared to the cost of instuction fetch, however, the operand accesses are much less important. The number of operand bytes read per instruction is typically 0.35, and the number of bytes written is typically 0.2. Using a cost of 4 cycles per byte of instruction fetched and 3 cycles per byte of operand referenced, this implies that a typical instruction of 1.6 bytes requires 6.4 cycles for instruction fetch and execution, but only 1.6 cycles for the operand references.

The table also gives the instructicn execution rate in millions of instructions per second (MIPS), which is often used to compare computers. Knowing that an IBM 370/168 can execute instructions at a rate of 2.5 MIPS makes the value of .25 MIPS for the 8080 seem quite good. But the MIPS rate is a poor indicator of the time required to execute a particular algorithm since it clearly depends cn the power of the individual instructions. A program written for the 370 will almost always be much more than 10 times faster than the equivalent program fcr the 8080.

Branch and Execution Distance Analysis

The 8080 suffers from the absence cf jump instructions which are relative to the program ccunter; as indicated in Table 7, about 80% of the successful tranches are made to locations within 127 bytes of the jump instruction. Considering the importance of jump instructions, a sizeable

savings in program size and speed (because the jump instruction can be smaller) results from the introduction of a relative jump.

Because of the execution pipelines in larger computers, the analysis of branch instructions has been an important source of information for their designers. No such sophistication exists in current microcomputers, but as tables 6 and 7 show, the fraction of branch instructions and fraction of successful versus unsuccessful branches are strikingly similar to the same data for the 370. Even more suprising is the fact that the average number of instructions between sucessful branches is just as low for the 8080 as for the 370 (typically 5 to 10 instructions) despite the difference in instruction set sophistication.

Measurement Summary

Despite large differences in scale and applications between the 370 and the 8080, we have found striking similarities. The measurements of their instruction set properties demonstrate characteristics independent of the size and type of the programs run. On reflection this seems understandable inasmuch as both machines are register oriented, use variable length instructions, and share basic instructions types (register to register, full address, and register indirect). This similarity illustrates how an architect can benefit from the study of existing computers

for the design of new microprocessors.

CONCLUSION

Current technological constraints are so stringent that the needs of the fabricator (IC manufacturer) have been at least as important a consideration as the needs of the user. In the future the user's needs will take precedence as the component density increases enough to offset die size limitations. Package limitations may still be a problem, but circuit sophistication will make the architectural effect less important.

Knowledge of computer architecture and the technological constraints encountered in the implementation are essential, but in a competitive industry the human and business factors cannot be neglected. Much of the success of an architecture group depends on such things as the assessment of user needs, corporate cooperation, and management understanding of computers and technical issues. In that corporate context, the results of computer architecture are judged primarily by market acceptance; a successful architecture is one that sells. This is not to say that an unsuccessful product is architecturally unsuccessful. Some of the microprocessors currently available would seem not to be afflicted with the idiosyncracies and lack of regularity discussed in this

paper. Often, however, they suffer from other disadvantages that make them commercially less successful, such as slow speed, unusual packaging, or late introduction.

Computer architecture is a comparative field where much of the knowledge comes from studying previous computers. For the design of microcomputers, this implies that a computer architect can benefit from the study of large and mini computers. The basic problems of implementing a computer within the constaints of technology may dictate different solutions, but many of the same techniques are applicable regardless of size.

ACKNOWLEDGEMENTS

We wish to thank Federico Faggin and John Banning for their help in reviewing an early version of this paper.

REFERENCES

[1] F.P. Brooks, Jr., "Architectural Philcsophy" in "Planning a Computer System - Project Stretch", Edited by W. Buchholz, McGraw-Hill, 1962.

[2] F. Faggin, "The Role of Technology in Microcomputer Design and Evolution", Circuits and Systems, Vol 7, No 5, Feb 1975, pps 4-13.

[3] C. Bass, D. Brown, "A Perspective on Microcomputer Software", Proc. IEEE, Vcl. 64, No. 6, June 1976, pps. 905-909.

[4] T. Blakeslee, "Digital Design with Standard MSI and LSI", Wiley, New York, 1975.

[5] J.D. Gannon, J.J. Horning, "The Impact cf Language Design cf the Production cf Reliable Software", Proc. Intl. Conf. on Reliable Software, April 1975., pps. 10-22

[6] B.L. Peutc, L.J. Shustek, "An Instructicn Timing Model of CPU Performance", Proc. 4th Annual Computer Architecture Symposium, March 1977 (tc appear).

[7] F. Baskett, L.J. Shustek, "The Design of a Low-Cost Video Graphics Terminal", Computer Graphics, Vol 10, No 2, July 1976, pps. 235-340

[8] G.A. Kildall, "High-Level Language Simplifies Microcomputer Programing", Electrcnics, June 27, 1974, pps 103-109

Table 1 - DYNAMIC OPCODE FREQUENCIES

SORTED BY OCCURENCE — SORTED BY EXECUTION TIME

Program: VGT

Opcode	% Instr	% Time	Opcode	% Instr	% Time
JMP CC,XXX	14.92	16.64	JMP CC,xxx	14.92	16.64
LOD R,R	13.54	7.55	LOD R,R	13.54	7.55
PUSH RR	6.13	7.52	PUSH RR	6.13	7.52
POP RR	5.92	6.60	CALL U,xxx	3.80	7.21
ANDI n	4.86	3.79	POP RR	5.92	6.60
RET U	4.18	4.66	LD A,xxx	4.18	6.06
LD A,XXX	4.18	6.06		---	---
	---	---		48.48	51.57
	53.72	52.82			

Program: EDITOR

Opcode	% Instr	% Time	Opcode	% Instr	% Time
LOD R,R	25.58	18.21	LOD R,R	25.58	18.21
BOT	10.04	5.72	JMP CC,xx	8.48	12.07
JMP CC,xxx	8.48	12.07	PUSH RR	5.37	8.41
POP RR	5.37	7.65	POP RR	5.37	7.65
PUSH RR	5.37	8.41	BOT	10.04	5.72
	---	---		---	---
	54.84	52.05		54.84	52.05

Table 2 - DYNAMIC OPCODE PAIR FREQUENCIES

Program: VGT

Opcode #1	Opcode #2	% Inst Measured	% Inst Expected	Ratio Meas./Exp.
PUSH RR	PUSH RR	3.72	0.38	9.89
ANDI n	JMP CC,xxx	3.02	0.72	4.16
JMP CC,xxx	LOD R,R	2.84	2.02	1.41
LOD R,R	ANDI n	2.84	0.66	4.31
POP RR	POP RR	2.40	0.35	6.85

Program: TBASIC

Opcode #1	Opcode #2	% Inst Measured	% Inst Expected	Ratio Meas./Exp.
JMP CC,xxx	INC RR	6.69	1.71	3.91
CMP (HL)	JMP CC,xxx	6.07	0.96	6.30
INC RR	CMP (HL)	4.76	0.82	5.83
CMPI n	RET CC	3.63	0.39	9.36
CMPI n	JMP CC,xxx	3.47	1.03	3.35

Table 3 - STATIC OPCODE FREQUENCIES

Program: VGT

Opcode	% Instr
LOD R,R	11.53
JMP CC,xxx	8.29
CALL U,xxx	6.07
LODI RR,xxx	5.74
LODI R,n	5.56
ST A,xxx	5.09
LD A,xxx	3.91
JMP U,xxx	3.91
	50.09

Program: EDITOR

Opcode	% Instr
LOD R,(HL)	12.92
LODI R,n	11.65
LODI RR,xxx	9.63
ST R,(HL)	7.66
LOD R,R	7.07
INC R	6.39
	55.33

Table 4 - STATIC OPCODE PAIR FREQUENCIES

Program: VGT

Opcode #1	Opcode #2	% Inst Measured	% Inst Expected	Ratio Meas./Exp.
LOD R,R	LOD R,R	2.59	1.33	1.95
CMPI n	JMP CC,xxx	1.74	0.18	9.71
ROT	ROT	1.69	0.14	12.26
IOR R	JMP CC	1.22	0.10	11.89
LODI R,n	ST A,xxx	1.08	0.28	3.83

Program: EDITOR

Opcode #1	Opcode #2	% Inst Measured	% Inst Expected	Ratio Meas./Exp.
ST R,(HL)	INC RR	3.29	0.43	7.73
INC RR	ST R,(HL)	3.17	0.43	7.41
LODI RR,xxx	LOD R,(HL)	3.10	1.24	2.49
INC R	LODI R,n	3.05	0.74	4.09
LOD R,(HL)	INC R	2.89	0.72	4.00

Table 5 - INSTRUCTION STATISTICS

Program	# Inst	Instruction Size % 1-byte	% 2-byte	% 3-byte	Avg. bytes per Inst
VGT	85,798	52.50	11.24	36.26	1.838
IBASIC	177,486	64.90	13.37	21.73	1.568
MUSIC	168,768	70.30	19.63	10.07	1.398
JBASIC	226,826	76.61	4.03	19.36	1.428
EDITOR	197,330	73.51	10.57	15.92	1.424

Program	Bytes Read per Inst	Bytes Written per Inst	# Cyles per Inst	MIPS at 2 Mhz
VGT	.350	.303	8.967	.223
IBASIC	.343	.204	7.851	.255
MUSIC	.178	.165	6.771	.295
JBASIC	.226	.169	7.002	.286
EDITOR	.209	.160	7.024	.285

Table 6 - EXECUTION DISTANCE BETWEEN SUCCESSFUL JUMPS

Program	Average (bytes)	Std. Dev.	Average (instr)
VGT	9.526	6.771	5.184
IBASIC	7.098	6.629	4.526
MUSIC	10.547	11.770	7.546
JBASIC	12.685	10.287	8.886
EDITOR	15.879	13.888	11.150

Table 7 - TYPES OF JUMPS

Prog.	Jumps as % Inst	Types of Jumps %Uncond	conditional % Succ	% Unsucc	Branch Distances --- All Branches -- 0 to 127	0 to -127
VGT	16.15	7.62	58.08	34.30	65.36	27.69
IBASIC	17.64	19.43	33.55	47.02	25.13	64.83
MUSIC	4.67	26.72	38.92	34.36	37.58	56.59
JBASIC	8.94	13.63	41.81	44.55	36.75	27.95
EDITOR	10.65	20.45	51.21	28.34	47.61	43.83