stichting
mathematisch
centrum

MC

AFDELING INFORMATICA IW 129/80 JANUARI
(DEPARTMENT OF COMPUTER SCIENCE)

J. HEERING

A COMPARITIVE ANALYSIS OF THE INTEL 8086, THE ZILOG Z8000 AND THE MOTOROLA MC68000 MICROPROCESSORS

Preprint

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O).

1980 Mathematics subject classification: 68A05

ACM Computing Reviews Category: 6.21, 6.35

A comparative analysis of the Intel 8086, the Zilog Z8000 and the Motorola MC68000 microprocessors*)

by

Jan Heering

ABSTRACT

The architectures of three of the latest 16-bit microprocressors, viz. the Intel 8086, the Zilog Z8000 and the Motorola MC68000, are discussed.

KEY WORDS & PHRASES: 16-bit microprocessors, Intel 8086, Zilog Z8000, Motorola MC68000, decentralized systems, computer architecture

*) This paper is not for review; it is intended for publication elsewhere.

1. INTRODUCTION

1.1. General

Recent advances in large scale integrated circuit technology have made possible the cheap production of high-performance 16-bit microprocessors. In combination with a small number of support chips and a suitable amount of memory the new devices are powerful enough to compete with existing low and high end minicomputers. At the low end their application range overlaps with that of their 8-bit predecessors.

The designer who decides to incorporate one or more of the new 16-bit microprocessors in his system has to take into account a large number of diverse factors, some of which have emerged only recently as a direct or indirect result of decreasing hardware prices.

In the following an attempt will be made to isolate the most important technical issues facing the designer. They will be used as a framework for the discussion of three of the latest 16-bit micros, viz. the Intel 8086 [1,2], the Motorola MC68000 [3,4] and the Zilog Z8000 [5,6,7].

1.2. Trend towards decentralization

Rapidly dropping hardware prices do not merely confront us with a quantitative change, but they imply both a number of new application areas for computers as well as a need to rethink existing computer organizations. No longer are CPU's and memories expensive resources which must be shared to the utmost in order to reach an acceptable price/performance level. As a result, one of the key concepts in the current development is the decentralization of computing power. A few examples may clarify in what ways decentralization can be achieved at different levels of the system hierarchy:

(1) File buffering, packing of logical records into physical records (blocks) and unpacking of blocks to logical records, tasks which have traditionally been performed by the operating system, can be off-loaded into the device controllers. This has a dual advantage. Modularity is improved and the device relieves its host from a non-trivial task. This is an example of a general tendency to off-load device driver and file system functions into (programmable) peripheral controllers and I/O processors.

(2) Screen editors can be split in two parts, a front-end which can be off-loaded into the terminal and a back-end which runs on the terminal host. The front-end keeps track of the cursor position, accepts edit commands from the user and modifies its display memory accordingly, while sending the modifications on to the back-end. The latter manages the workfile and supplies the front-end with new lines of text on request. A further step towards decentralization is to off-load the entire editor into the terminal. To this end the

terminal has to be equipped with a small mass memory for storage of editor text- and workfiles.

(3) Multiprogramming and time-sharing were in large part invented to get more productivity out of expensive mainframes. Technically, time-sharing is difficult and the operating systems required for it tend to be very complex and expensive pieces of software. An alternative system should preferably eliminate the need to share CPU and memory resources, but it should retain or improve upon existing facilities for interprocess communication and data sharing, because these are essential for many applications. A network of single user systems, coupled by high capacity communication links to a central computer, which handles file system calls and manages disks, magnetic tape units and fast line printers, is an example of a configuration which is rapidly becoming an attractive replacement for a classical, centralized time-sharing system. In such a system every process has its own CPU and memory, but data sharing and interprocess communication are possible. The central machine runs a message driven multi-user file system and acts as an intermediate station in the exchange of messages between the satellites. Because different processes run on different computers, memory allocation is easy and there is no memory protection problem. File protection is handled by the central machine. Not all satellite processors need have the same capabilities. Some of them might, for instance, be tailored to execution of a single language, while others might be oriented towards graphical interaction or text formatting. Many variations on this theme are possible, all of them belonging to the province of distributed systems. Nothing illustrates better the current drive towards decentralization than the fast pace at which distributed systems are evolving.

(4) On a larger scale distributed systems are envisioned for the decentralized control of plants, electrical power networks, etc. [8]. Centralized control of plants has turned out to be extremely difficult, essentially because the multiplexing of sharable resources (memory, CPU time, I/O channels) leads to an unacceptable degradation of system response. Even in relatively simple systems it is often difficult or impossible to obtain adequate resources for a high-priority process rapidly enough every time its execution is triggered. An obvious solution is to off-load high-priority processes into separate, dedicated processors and link these to the central machine. Depending upon the complexity of the environment to be controlled it can make sense to continue this off-loading strategy to a depth of several levels in order to keep resource sharing to a minimum.

1.3. Technical issues

Any user of a 16-bit micro will be confronted with a number of technical issues, some of which have system-wide implications. The most important of them are listed below:

(1) High-level languages and portability of software.

(2) Interprocessor communication and portability of hardware.

(3) (Decentralized) operating systems and functional off-loading.

Some issues, like high-level languages and portability of application programs, are not in any way new, but others, like decentralized operating systems, portability of hardware and functional off-loading, have come to the fore as a direct result of the trend towards decentralization.

Because they are oriented towards interprocessor communication, decentralized systems are inherently more open-ended than their centralized counterparts. As a result they are frequently heterogeneous (in the sense that they consist of interconnected processors of different types) and they almost always show a strong tendency to become more heterogeneous with time. It is therefore important to make everything, ranging from application programs to operating systems and from peripheral interfaces to data communication, as portable as possible.

The requirement for portability of both software and hardware has lead to various standardization efforts in the fields of data communication, peripheral buses, floating point representation and high-level languages [9]. The designer should be aware of the existence of these standards, because if he doesn't comply with them he is making things unnecessarily difficult for himself. He should also be aware of the fact that currently existing standards are no panacea. Making operating systems portable will still be a difficult job, even if they are written in PASCAL which is becoming a standard high-level language for microprocessors. The reason is, of course, that there aré large architectural differences between different computers. In the near future microprogrammable microprocessors may offer a cheap solution to this problem. Both the 8086 and the 68000 are microprogrammed, but their microprograms cannot be changed. Although they have not yet been announced, versions with writable control memory (either off- or on-chip) and writable opcode mapping tables can confidently be expected to be available soon [10]. These devices would have a variable macro architecture and (depending upon their micro architecture) could be used as general purpose emulators.

Together the points listed above constitute a suitable framework in which to discuss the merits of various 16-bit microprocessors and their associated families of support chips. Separate sections will be devoted to the details of why each issue is of importance and how it might influence the designer in his choice of specific implementations. The various existing standards will be discussed in the appropriate sections.

2. SYNOPSIS OF THE 8086, THE Z8000 AND THE MC68000

The main features of the three micros mentioned in the title of this section are listed in table 1. While the 8-bit micros developed so far are rather primitive in comparison with existing 16-bit minis, this is no longer true for their 16-bit successors. Table 1 will probably hold some surprises for readers accustomed to the Turing machine-like quality of 8-bit microprocessors.

First of all, address spaces have become huge. The 8086 supports a modest 4 segments of 64kbyte each per process as long as the latter does not modify its own relocation registers. If it does, 1Mbyte is available. The Z8000 is offered in two versions: the Z8001 and the Z8002. The Z8002 package has fewer pins and does not allow for off-chip memory management. The Z8001, however, in combination with the Z8010 memory management unit supports segmented virtual memory. A process which does not change its own memory map could have a total address space of 384 segments of 64kbyte each. As each Z8010 can hold 64 segment descriptors, 6 Z8010's would be needed to perform virtual to physical address translation in this extreme case. Details on the virtual memory organization of the 68000 were not available at the time of writing. Without memory management a process can address something like 16Mbyte code and 16Mbyte data on this machine.

A second important point is that the 16-bit micros examined here all have full 16-bit integer arithmetic. The Z8000 even has full 32-bit integer arithmetic (including multiply and divide). The 8086 and the Z8000 also have byte-string instructions, such as string move and compare. These are important in many applications such as text editing and parsing. They have hitherto rarely been present on 16-bit machines, however.

Last but not least all three processors have instructions with which interlocks (semaphores) can be implemented to synchronize access to sharable resources in a multiprocessor configuration. This is wholly in agreement with the trend towards decentralization mentioned in section 1.2.

A few remarkable implementational features deserve special mention. The 8086 uses instruction prefetch to speed up its operation. It has a 6-byte instruction queue, which it tries to keep filled at all times. The instruction set of the 68000 is interpreted by two levels of microcode to minimize the number of control memory bits (and thus the chip area) required. The first level has vertical type microinstructions. These consist of jump addresses and pointers to instructions at the second level. Instructions at this level (nanoinstructions) are highly horizontal (70 bits wide) and directly control the execution unit. There is a single nanoinstruction for every microinstruction in the sense that identical nanoinstructions are not duplicated in the nano control memory, but a single copy is shared between different microinstructions. Sequencing is done at the microinstruction level [10]. The 8086 is microprogrammed also, but the Z8000 is not. By sheer coincidence the Z8000 chip contains about as many transistors (17 500) as the ENIAC, the first general purpose electronic computer, had tubes [11]. If one compares the performance of both processors (in instructions/second/dollar) it becomes clear that CPUs do not constitute profitable investment. The inflation of CPUs is matched only by the inflation of the Deutsche Reichsmark between the First and the Second World War.

The 8086 came into the market in the middle of 1978, about one year before the Z8000. Volume production of the 68000 is expected to begin in early 1980. This difference in age is reflected in the number of support chips available for each processor. For the Intel machine there are a bus controller for large configurations (8288), a floating point processor (8087) and an I/O processor (8089) [12]. Furthermore, Intel has a complete single board computer (the iSBC 86/12) based on the 8086 and oriented towards multiprocessing [13]. Several other manufacturers offer single board systems based on the 8086 or the Z8000. These generally have S-100 bus interfaces (see section 4). Motorola has recently released the MEX68KDM design and evaluation module for the 68000. S-100 bus oriented boards for the 68000 are expected to be available soon from independent manufacturers.

In the following sections successive magnifications of table 1 will be presented corresponding to the various issues listed in paragraph 1.3. Together they should provide the reader with a fairly comprehensive picture of the architectural features of the micros under discussion.

3. HIGH-LEVEL LANGUAGES AND PORTABILITY OF SOFTWARE

The advantages of high-level language programming as opposed to assembly language programming are well known. Programming in high-level languages is easier and the resulting programs are easier to debug and to maintain. Furthermore, programs in a high-level language are to a large measure machine independent. With the advent of microprocessors these advantages have lost nothing of their importance. First, the ratio of software costs to hardware costs tends to be very high for microprocessor based systems because the hardware is so cheap. The use of high-level languages helps to keep software costs down. Secondly, as was pointed out in section 1.3, the decentralized and heterogeneous character of many microprocessor based systems calls for program portability. The most important method to achieve this is to use a widely available, standardized high-level language. A second method, which is becoming increasingly popular, is to base software systems on a machine independent intermediate language. If the intermediate language is of a low level, such software can simply be implemented on any computer by writing an

	8086	Z8000	MC68000
Manufacturer	Intel	Zilog	Motorola
Second source	Mostek	AMD	Rockwell
Clock frequency	5MHz (fast 8MHz version announced, but not yet available)	4MHz	8MHz
Bus cycle time	800nsec (500nsec for 8MHz version)	750nsec	500nsec read 750nsec write
Execution time of 16-bit add memory to register (absolute addressing mode)	3microsec (1.875microsec for 8MHz version) (prefetch not taken into account)	2.25microsec (non-segmented) 2.5microsec (8-bit segment offset) 3microsec (16-bit segment offset)	1.5microsec (16-bit address) 2microsec (24-bit address) (no timing available for segmented operation)
CPU organization	register oriented	register oriented	register oriented
I/O organization	I/O mapped or memory mapped	I/O mapped or memory mapped	memory mapped - no separate I/O instructions
Vectored priority interrupt	yes - with separate 8259A programmable interrupt controller(s)	vectored interrupt - no on-chip priority arbitration	yes - 7 levels
Byte addressing	yes - 16-bit words can be fetched from both even and odd addresses. Odd-address fetches take an extra memory cycle.	yes - 16-bit words can be fetched from even addresses only.	yes - 16-bit words can be fetched from even addresses only.

Table 1

	8086	Z8000	MC68000
Virtual memory and memory protection	no - on chip address relocation. No segment limit detect.	yes - with separate Z8010 memory manag-ement unit(s) (Z8010 announ-ced but not yet available)	yes - with separate memory management unit (announced - no details available)
Max. address space per process	64kbyte code 2*64kbyte data 64kbyte stack	with memory management 128*64kbyte code 128*64kbyte data 128*64kbyte stack without memory management 8Mbyte code 8Mbyte data 8Mbyte stack	with memory management unknown; without memory management 16Mbyte code and 16Mbyte data
Max. amount of physical memory	1Mbyte	with memory management 16Mbyte/Z8010; without memory management 24Mbyte user and 24Mbyte system	with memory management unknown; without memory management 32Mbyte user and 32Mbyte system
System/user mode	no	yes	yes
Interlock instructions for multiprocessor operation	yes	yes	yes
16-bit integer multiply/divide	yes	yes	yes
32-bit integer multiply/divide	no	yes	no
String operations	yes	yes	no
Floating point operations	no	no	no

Table 1 (cont'd)

interpreter for the intermediate language. Many PASCAL systems, for instance, use a low level intermediate language called P-code. These systems are highly portable.

The two most popular high-level languages for microprocessors are BASIC and PASCAL. Standards for both are in the making [14,15]. On the 16-bit micros BASIC and PASCAL will probably be joined by FORTRAN, which has never been very popular on the 8-bit machines. With features like programmer definable datatypes, recursive procedures, and ALGOL-like control structures PASCAL is by far the most powerful of the three and is gaining rapidly in popularity. The acceptance of a PASCAL standard will only further this trend. Other languages in use are mostly subsets of PL/I, like Intel's PL/M, Zilog's PLZ and Motorola's MPL. These languages have not found wide acceptance beyond the products of the companies that introduced them and they are not candidates for standardization at this time. PASCAL for the 8086 has just been released by Intel, while PASCAL compilers for the Z8000 and the 68000 have been announced. FORTRAN has also been announced for all three machines.

Table 2 lists a number of features which may be of interest to the compiler writer. All three machines are register oriented, i.e. generally at least one of the operands of a dyadic operation has to reside in a register. Stacks are used mainly to store procedure activation records. The Z8000 and 68000 have much to offer in this respect. Apart from standard procedure call and return instructions, they have operations to save/restore a selected set of registers on/from the stack on procedure entry/exit. The 68000 also has instructions to establish a new stack frame on procedure entry (link stack instruction), and to return to the previous stack frame on procedure exit (unlink stack instruction). In this way procedure call overhead can be significantly reduced. Local variables and paramaters can be addressed on all three processors using the register-indirect-with-offset addressing mode. On the 8086 and the Z8000 call-by-reference parameters can be implemented using the load-effective-address-into-register instruction followed by a push. The 68000 can do this in one instruction. The 68000 also has a memory-to-memory move which is equivalent to a simple assignment statement in high level languages.

The 8086 has a highly irregular architecture as far as the use of registers is concerned. Almost none of its 12 registers can be considered as truly general purpose. Many instructions as well as certain addressing modes implicitly use dedicated registers. The Z8000 and the 68000 are much better in this respect, although the Z8000 suffers from a lack of regularity in that memory reference instructions generally cannot be combined with all available addressing modes, but only with a subset which varies from instruction to instruction. Relative addressing, for instance, can be used only to a very limited extent, because most instructions do not permit it. Similarly, auto-increment and -decrement addressing modes apply only to string moves and string I/O. For the sake of coding efficiency the Z8000 in some cases has two identical instructions, which differ only with respect to admissible addressing modes. The 68000 has a reasonably consistent instruction set. It is expected to be the best architecture for the execution of compiler generated code and the most convenient one for the assembly language programmer.

Although the second most important method to achieve program portability is based on interpretation, interpreters do not seem to be foremost in the minds of 16-bit microprocessor designers, as evidenced by the number of 'no' entries in table 3. All interpreters consist of a main loop, which does virtual instruction decoding and mapping, and a number of interpretation routines corresponding to individual virtual instructions. If the code to be interpreted is low-level in nature (like PASCAL P-code), the main loop of the interpreter contributes decisively to total interpretation time. Even in case of high level code the overhead due to decoding is generally not negligible, because the most frequently occurring statements are simple assignments which take very little time to execute [16]. Decoding and mapping overhead can be reduced if the host processor has the right instructions and an adequate number of registers. As can be seen from table 3 interpreters are not very well supported by any of the three microprocessors, because they do not have bit field select instructions. The 8086 also lacks an indexed jump. With bit field instructions lacking the intermediate language designer does best to pack everything in bytes or multiples of bytes, although he may incur a considerable space overhead in doing so. In some cases bit operations (on the Z8000 and the 68000) may be useful.

4. INTERPROCESSOR COMMUNICATION AND PORTABILITY OF HARDWARE

A network consisting of more than two or three computers will survive any of its components. It is generally simply not practicable or necessary to replace the entire network. To facilitate replacement of subsystems and addition of new and different subsystems, the intercommunication network should conform to existing data communication standards, while individual processors should preferably use standardized buses. With larger computers this cannot yet be achieved, but with microprocessors it is just now becoming possible to build such highly standardized systems. The main reasons are that IEEE standards have been proposed or established for a number of microprocessor buses, viz. the S-100 bus [17], the General Purpose Interface Bus [18] and Intel's MULTIBUS [19], while the CCITT has almost finished work on its X.25 protocol for public (and private) data networks [20].

The S-100 bus in its present, non-standardized, form is used by some 200 000 personal computer systems. According to the latest standard proposal, the bus will be upgraded to 16 data lines and 24 address lines, while timing specifications will be rigorously defined. This extension will make the bus eminently suitable for use with 16-bit micros. Single board systems for the 8086 and the Z8000, which conform to the proposed S-100 standard, are already offered by several independent manufacturers.

	8086	Z8000	MC68000
Datatypes		bits,	bits,
	1-digit ASCII numbers,		
	2-digit BCD numbers,	2-digit BCD numbers,	2-digit BCD numbers,
	8- and 16-bit logicals,	8- and 16-bit logicals,	8-, 16- and 32-bit logicals,
	8- and 16-bit signed and un-signed integers,	8-, 16- and 32-bit signed integers,	8-, 16- and 32-bit signed and unsigned integers,
	32-bit addresses (20 bits used), byte strings, word strings	32-bit addresses (23 bits used), byte strings, word strings	32-bit addresses (24 bits used)
Registers	4 more or less general purpose, 1 procedure call stack pointer, 1 stack frame pointer, 2 data offset/ index, 4 segment relo-cation, 1 status	15 general pur-pose, 1 procedure call stack pointer, 1 status	8 data (32-bits), 7 address (32-bits), 1 procedure call stack pointer (32-bits), 1 status
Addressing modes	immediate, reg direct, reg indirect, reg indirect indexed, reg indirect with offset, reg indirect indexed with offset,	immediate, reg direct, reg indirect, reg indirect indexed, reg indirect with offset,	immediate, reg direct, reg indirect, reg indirect indexed, reg indirect with offset, reg indirect indexed with offset,
		reg indirect with postdecre-ment,	reg indirect with predecre-ment,

Table 2

	8086	Z8000	MC68000
Addressing modes (cont'd)	absolute,	reg indirect with postincrement, absolute, absolute indexed, relative,	reg indirect with postincrement, absolute, relative, relative indexed with offset
Procedure call support	procedure call stack; call, return, and load effective address operations; push and pop operations	procedure call stack; save/restore registers, call, return, and load effective address operations; push and pop operations	procedure call stack; save/restore registers, link/unlink stack, call, return, load effective address and push effective address operations; move instruction in combination with auto-increment and -decrement addressing modes
Other stacks besides procedure call stack	no	yes - 7 register pairs can be used as stack pointer	yes - any of the 7 address registers can be used as stack pointer
Hardware stack overflow/underflow detect	no	yes - causes segmentation trap from external memory management unit	not on-chip. Maybe with external memory management unit. No details available
Arithmetic error traps	divide by zero, overflow	none	divide by zero, overflow

Table 2 (cont'd)

	8086	Z8000	MC68000
Decrement and branch not zero for loop control	yes	yes	yes
Multiple precision arithmetic support (add with carry, sign extend, etc.)	yes	yes	yes

Table 2 (cont'd)

	8086	Z8000	MC68000
Bit field select for opcode isolation	no	no	no
Bit instructions	no	yes	yes
Indexed jump for opcode mapping	no	yes - absolute indexed only	yes - relative indexed only
Adequate number of registers to hold virtual PC, virtual stack pointer, etc.	marginal	yes	yes
Multiple stacks		see table 2	
Multiple precision arithmetic support		see table 2	
Word string moves for compactifying garbage collector (for LISP, etc.)	yes	yes	no

Table 3

	8086	Z8000	MC68000
Interrupt and I/O organization		see table 1	
Direct memory access	yes	yes	yes
Compatible with already existing 8-bit peripheral chips of the same manufacturer	yes	no	yes
High-speed synchronous/asynchronous serial interface	yes - 8251A	not yet, but expected soon	yes - MC6850, MC6852
HDLC interface	yes - 8273 (64kbits/sec max.)	not yet, but expected soon	yes - MC68B54 (2Mbits/sec max.)
GPIB (IEEE-488) interface	yes - 8291, 8292	no	yes - MC68488
S-100 bus oriented single board systems	yes - several available from independent manufacturers	yes - several available from independent manufacturers	not yet, but expected soon

Table 4

Cheap high capacity (tens of Mbytes) disk units (using the new 8-inch Winchester technology pioneered by IBM) will be available soon together with S-100 interface cards. As yet there are no S-100 bus interfaces available for industry standard magnetic tape.

The General Purpose Interface Bus (GPIB - IEEE-488 standard) is a small scale byte oriented bus. It is already in wide use to interface all kinds of devices to each other and to microprocessors and minicomputers. Due to its limited bandwidth (500kbyte/sec typical) and its limited addressing capability, the GPIB is not well suited to act as main system bus in microprocessor systems, but it can be used to advantage as auxiliary I/O bus. GPIB adapters for the S-100 bus are available from several manufacturers.

Intel's MULTIBUS has been upgraded to 16 data and 20 address lines for use with the 8086. In its original form this bus is also quite popular. An IEEE task force is currently working on a further standardization of the MULTIBUS. The final standard is expected very soon.

The CCITT X.25 protocol is already being used by several large scale data networks and most countries that are planning a public data network will adopt it. Small scale distributed systems will probably also increasingly be based on X.25 (except IBM's which use a different protocol). X.25 level 1 (the physical level) specifies synchronous serial communication conforming to the EIA RS-422/423 standard. The latter is an upgraded version of the well-known RS-232 specification. Depending on the network speeds up to several Mbits/sec are possible. X.25 level 2 (the data link level) specifies the basic frame format as well as address and control conventions and a frame check sequence for error detection purposes. Level 2 is compatible with the ISO High Level Data Link Control (HDLC) standard. Chips for both level 1 and level 2 are available from many manufacturers, although most of the HDLC chips (like Intel's 8273 and Motorola's 68B54) do not completely implement level 2. An exception is the WD2501/2511 "micro packet network interface" recently announced by Western Digital Co.

5. OPERATING SYSTEMS AND FUNCTIONAL OFF-LOADING

With two processor modes and off-chip virtual addressing and memory protection (announced, but not yet available), the Z8000 and the 68000 are well suited to classical time-sharing and multiprogramming applications. This may seem a bit surprising in view of the current trend away from multiuser systems, but the usefulness of both processors for personal computers and distributed systems is in no way diminished by their multiprogramming capabilities. Without memory management the Z8001 still has an 8Mbyte direct addressing range (23-bits address), extendable to a maximum of 48Mbyte if a distinction is made between code, data and stack access for each processor mode. (32Mbyte is a more realistic maximum, because the Z8000 cannot in all cases distinguish a stack access from a data access, so in practice it will not be feasible to implement separate stack and data spaces.) The bare 68000 offers the user a 16Mbyte direct addressing range (24-bits address) and four times as much if access type and processor mode are taken into account. See table 5 for further details.

As a result of the tendency to off-load functions into specialized slave processors (see paragraph 1.2), tightly coupled multi microprocessor systems emerge rather naturally. Processors in such a configuration require special interlock (semaphore) operations for synchronization purposes. Both the 8086 and the 68000 have such operations, while the Z8000 provides a rather complicated mechanism the details of which will not be given here [6,7]. Suffice it to say it is not at all clear that this mechanism is more powerful than the simpler operations offered by the other two. Multiprocessor configurations also require a special bus controller to resolve bus conflicts that would otherwise occur due to the uncoordinated operation of several CPUs. Such multimaster bus controllers will shortly be available for all three processors.

	8086	Z8000	MC68000
Virtual memory, memory protection, address space, etc.		see table 1	
System/user mode	no	yes	yes
Automatic memory map switch on processor mode switch	not applicable	yes	unknown
Automatic stack pointer switch on processor mode switch	not applicable	yes	yes
Interrupt and I/O organization		see table 4	
Multiple stacks and stack limit detect		see table 2	
Self-relative (position independent) code	no	to a very limited extent - most memory reference instructions do not permit relative addressing	yes
System call	yes	yes	yes
Reentrant procedures	yes	yes	yes

Table 5

The 8086 has an interesting instruction (escape) which is intended for high speed transfer of control to a slave processor. The only thing the escape instruction does is to put the effective address of its operand on the bus. It is to be used in combination with the wait instruction, which suspends operation of the processor until the latters test input is asserted by an outside source. By monitoring the bus the slave processor is able to detect any escape instruction fetched by the master and to subsequently intercept the corresponding address. An example may clarify this. Suppose one would like to let a slave processor perform a certain operation OP (for instance, a floating point add) with two operands A and B and result C. In that case one would write

```
esc OP
esc A
esc B
esc C
wait
```

The four escape instructions supply the slave processor with pointers to the opcode, the two operands and the location of the result. After that the 8086 executes the wait and goes to sleep. When the slave is finished, it wakes the 8086 by asserting its test input. If no slave processor is present, software interpretation of OP can be invoked simply by changing the first escape to a trap instruction. Table 6 summarizes the multiprocessing and functional off-loading facilities offered by each processor.

	8086	Z8000	MC68000
Trap to specified vector	yes	no - use sytem call with parameter or unimplemented instruction trap	yes
Escape to slave processor	yes - see text	no	no
Interlock instructions for multiprocessor operation	yes - special lock prefix in conjunction with exchange memory with register instruction	yes - see text	yes - indivisible test and set instruction

Table 6

6. CONLUSION

Of the three 16-bit microprocessors examined in this report, the Motorola MC68000 is the fastest and has the best architecture for the execution of compiler generated code and the most convenient one for the assembly language programmer. The Zilog Z8000 has a comparable, but less regular architecture. Both processors support essentially infinite amounts of memory and, with off-chip memory management, are well suited for multiprogramming and time-sharing applications. The Intel 8086 is less sophisticated than the other two, but, at least at the present time, it is the strongest as far as support chips and single board systems are concerned. All three processors are suitable for multiprocessor operation. Somewhat surprisingly, interpreters are not very well supported by any of the three machines.

REFERENCES

[1] MCS-86 User's Manual. Intel Co., Pub. No. 9800722A, July 1978.

[2] Morse, S.P., Pohlman, W.B. & Ravenel, B.W. 'The Intel 8086 microprocessor: A 16-bit evolution of the 8080.' Computer, June 1978, pp. 18-27.

[3] MC68000 Preliminary Product Description. Motorola Inc., 1978.

[4] Stritter, E. & Gunter, T. 'A microprocessor architecture for a changing world: The Motorola 68000.' Computer, February 1979, pp. 43-52.

[5] Z8001/Z8002 CPU Product Specification. Zilog Inc., Pub. No. 03-8002-01, Preliminary ed., March 1979.

[6] Z8000 CPU Instruction Set. Zilog Inc., Pub. No. 03-8020-01, Preliminary ed., February 1979.

[7] Peuto, B.L. 'Architecture of a new microprocessor.' Computer, February 1979, pp. 10-21. [The Zilog Z8000].

[8] Kahne, S., Lefkowitz, I. & Rose, C. 'Automatic control by distributed intelligence.' Scientific American, June 1979, pp. 54-66.

[9] Gustavson D.B. 'Standards Committee activities: An update.' Computer, July 1979, pp. 61-64.

[10] Stritter, S. & Tredennick, N. 'Microprogrammed implementation of a single chip microprocessor.' The 11th Annual Microprogramming Workshop, 1978, pp. 8-16.

[11] Mauchly, J.W. 'Mauchly on the trials of building the ENIAC.' IEEE Spectrum, April 1975, pp. 70-76.

[12] El-Ayat, K.A. 'The Intel 8089: An integrated I/O processor.' Computer, June 1979, pp. 67-78.

[13] iSBC 86/12 Single Board Computer. Intel Co., Pub. No. 9800770A, 1978.

[14] ANS Committee X3J2/77 (Proposed standard for MINIMAL BASIC), May 1977.

[15] Ravenel B.W. 'Toward a PASCAL standard.' Computer, April 1979, pp. 68-82.

[16] Klint P. 'How inefficient are stack oriented machines?' Report IW 123/79, Mathematical Centre, Amsterdam, 1979.

[17] Elmquist, K.A., Fullmer, H., Gustavson, D.B. & Morrow, G. 'Standard specification for S-100 bus interface devices.' Computer, July 1979, pp. 28-52.

[18] IEEE Standard Digital Interface for Programmable Instrumentation. IEEE Std 488-1978. IEEE, 1978. (Can be ordered from IEEE Service Center, 455 Hoes Lane, Piscataway, N.J. 08854, USA.)

[19] Barthmaier, J. Intel MULTIBUS Interfacing. Intel Application Note AP-28A, Intel Co., Pub. No. 9800587B, 1979.

[20] Folts H.C. 'Status report on new standards for DTE/DCE interface protocols.' Computer, September 1979, pp. 12-19.