INTRODUCTION

This paper discusses the development of a C compiler for the Intel 80386 microprocessor. A brief discussion of the evolution of the 80386 is given, followed by a description of the 80386 register and instruction sets. From there, the C compiler's view of the 80386 is presented with emphasis on the features of the chip that make it attractive to a large-scale application such as the SAS system. In addition, comments on the future of the 80386, including operating system targets and processor evolution, are presented.

EVOLUTION OF THE 80386

The first member of the 80x86 family to gain wide acceptance was the 8088/8086. Presented as a 16-bit processor, the 8086 used a segmented memory architecture to maintain source level consistency with its 8-bit forerunner, the 8080A (for example, every 8080A instruction could be translated into an 8086 instruction). While this compatibility eased the transition from the 8-bit to the 16-bit PC, the segmentation concept proved to be a fundamental and limiting flaw in both the 8086 and its successor, the 80286.

Some major drawbacks of the 8086 chip were addressed in the 80286. While the 80286, a programmatic superset of the 8086, could be run in 8086 compatibility mode, it also introduced a privileged mode of operation that provided for memory protection and process control management. By using a virtual-to-physical memory mapping scheme, the 80286 running in privileged (or protected) mode could exceed the 1 megabyte memory limit of the 8086 and directly address up to 16 megabytes of memory.

However, the software overhead required by the segmented architecture of the 80286 proved to be a significant obstacle. The cost of tracking, manipulating, storing and loading segment registers was too high to perform a truly quality implementation of large-scale software systems (including operating systems).

This leads us to the 80386. Although Intel has once again attempted to maintain a level of compatibility with its predecessors, the power of the 80386 lies in its incompatibilities. In native 32-bit unsegmented flat model mode, the 80386 is a 32-bit processor capable of addressing up to 4 gigabytes of memory. It is also capable of efficiently addressing large areas of memory (up to 4 gigabytes). In addition, new addressing modes allow for increased flexibility in memory access.

It must be pointed out that the 80386 is capable of running in a 48-bit segmented model, allowing the direct access of up to 64 terabytes of virtual memory. This paper shall focus on an efficient implementation in the 32-bit flat model mode (no segment registers are used); the only currently viable 48-bit 80386 implementations are in the embedded systems area.

REGISTER AND INSTRUCTION SET

Consider the register set of the 80386 (see Figure 1). The shaded regions represent a direct mapping of the 80286 16-bit register set. It is worth noting that the high order 16 bits of the 32-bit registers are not available as distinct 16-bit entities.

<table>
<thead>
<tr>
<th>REGISTERS</th>
<th>AX</th>
<th>BX</th>
<th>CX</th>
<th>DX</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX</td>
<td>AX</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDX</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECX</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EBX</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESI</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EDI</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EBP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EIP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EFlags</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The instruction set for the 80386 will be very familiar to developers used to working with the 80286. Opcodes from the 80286 translate directly into 80386 opcodes. The 80286 8-bit manipulation instructions remain functionally unchanged and 16-bit manipulation instructions map to 32-bit manipulation instructions. The 80386 provides two new opcode prefix bytes, the operand-size prefix byte and the address-size prefix byte, to allow for the transformation of 32-bit instructions back to their 16-bit counterparts.

Instructions without immediate values use the same opcodes to map from 16-bit sequences to 32-bit sequences (6-bit instructions remain the same). Figure 2 demonstrates this, along with an example of a 16-bit instruction generated with an operand-size prefix byte.
instructions which make use of immediate values use similar op­
code sequences with 16-bit quantities changed to 32-bit quantities. Once again, 8-bit instructions remain the same (see Figure 3).

<table>
<thead>
<tr>
<th>OPCODES</th>
<th>80286</th>
<th>80386</th>
</tr>
</thead>
<tbody>
<tr>
<td>80 00</td>
<td>MOV AL, 1</td>
<td>MOV AX, 1</td>
</tr>
<tr>
<td>80 00 00</td>
<td>MOV AX, 1</td>
<td>MOV AX, 1</td>
</tr>
<tr>
<td>80 00 00 00</td>
<td>MOV AX, 1</td>
<td>MOV AX, 1</td>
</tr>
<tr>
<td>80 46 12</td>
<td>MOV AL[ES+12]</td>
<td>MOV AL[ES+12]</td>
</tr>
<tr>
<td>80 66 54 12</td>
<td>MOV AL[ES+1234]</td>
<td>MOV AL[ES+1234]</td>
</tr>
<tr>
<td>80 66 34 12 00</td>
<td>MOV AL[ES+1234]</td>
<td>MOV AL[ES+1234]</td>
</tr>
</tbody>
</table>

Figure 3: 80386 Instructions With Immediate Values

An important improvement of the 80386 is in the area of memory
addressing. All general purpose registers can now be used as
index registers. Also, a broad, new range of addressing modes has
been introduced. These changes are summarized in Figure 4.

| REG1 (optional) | REG1, EDX, ECX, ESP, EDI, ESP |
| REG2 (optional) | EAX, EBX, ECX, EDX, ESP, ESP |
| SCALE (optional) | 1, 2, 4, 8 |
| OFFSET (optional) | 8 or 32 bit quality |

Figure 4: 80386 Addressing Modes

Another significant improvement in the 80386 is addition of several
new instructions which are particularly useful for compiler writers.
These are summarized in Figure 5.

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVZX</td>
<td>Move of 8 and 16-bit quantities with sign extend</td>
</tr>
<tr>
<td>MOVZX</td>
<td>Move of 8 and 16-bit quantities with zero extend</td>
</tr>
<tr>
<td>SETc</td>
<td>Byte set on condition</td>
</tr>
<tr>
<td>Jcc</td>
<td>Jump on condition</td>
</tr>
<tr>
<td>SHRD</td>
<td>Double precision shift left or right (used in support routines)</td>
</tr>
<tr>
<td>BSR</td>
<td>Bit Scan reverse (used in support routines)</td>
</tr>
<tr>
<td>BIT(n)</td>
<td>Bit test (and modify) (used in support routines)</td>
</tr>
</tbody>
</table>

Figure 5: New 80386 Instructions

THE C COMPILER’S PERSPECTIVE

This section describes ways in which an 80386 compiler can take
advantage of the power of the processor. Remember when review­
ing the examples that 80386 instructions are typically longer than
their 80286 counterparts due to 16 vs. 32 bit offset differences. The
true efficiency of generated code must be measured by speed as
well as size. In each of the following examples, a significant speed
increase is achieved in generated 80386 code.

All 80286 examples are compiled using 32-bit pointer model, to
allow for a better functional comparison.

32-bit Linear Address Space

The greatest benefit of the 80386 is that it allows the compiler to
view memory as a 32-bit linear address space. Without the over­
head of segmentation, the compiler can generate faster and small­
er code in the areas of pointer arithmetic and memory access.
Example 1 demonstrates the improvements in generated code that
can be achieved in a linear address space.

In general, it is no longer necessary for the application program­
ter to assist the compiler in the code generation task by using extended keywords such as huge, near and far.

Example 1: Efficiencies of a Linear Address Space

32-bit default ints

Array indices are processed most effectively if default ints are the
same size as memory pointers. Thus, the natural size for default ints
on the 80386 is 32 bits. Note that the 80386 processes 32-bit
quantities more efficiently than 16-bit quantities, so there is no
performance penalty for this decision.

Another benefit of the 32-bit nature of the processor is that there
is less need for out-of-line code. Because the 80286 could not
efficiently perform operations on 32-bit quantities, frequent calls to
out-of-line support routines were necessary, as evidenced in Example 2.

A further consequence is that all integer and pointer return values can be returned in the single register EAX, freeing EBX for register variable use as outlined below.

\[ R16 \]

Example 2: Processing of 32-bit Quantities

Index registers

The addition of EAX, ECX and EDX as index registers allows for better register allocation algorithms. More index registers allow for keeping values in registers for longer periods of time during memory intensive operations. Example 3 demonstrates the efficiency of an increased number of index registers.

\[ R16 \]

Example 3: Using New Index Registers

In addition, the extra index registers allow the number of register variables to be increased from two to three. Previously, two register variables were the functional limit as one index register needed to be kept free at all times.

The natural choice for the third register variable is EBX (ESI and EDI are the first two, reflecting the choices of SI and DI for the 80286). Specialized instructions which make use of EAX, ECX and EDX discourage their use for register variables.

Example 4 shows why three register variables are not practical in the 80286 case and are of benefit in the 80386 case.

\[ R16 \]
New instructions

Another straightforward way to take advantage of the 80386 is to make use of the new instructions provided. Consider how the following new instructions are used:

MOVZX, MOVVSX — provide signed and unsigned conversion of 8 bit quantities to 16 and 32 bit quantities and signed and unsigned conversion of 16 bit quantities to 32 bit quantities. An important distinction from the 80286 instructions C万里和 CWO is their ability to convert in all registers and convert from memory.

Example 5: New MOVZX, MOVSX Instructions

Joe — allows conditional jumps to be taken within the entire 32-bit address space, eliminating the need for the tiresome jump around on condition construct.

Example 6: New Long Jump On Condition Instructions

SETcc — these instructions allow registers or memory locations to be set or reset depending on a condition code.

Example 7: New Set On Condition Instructions

New Addressing Modes

Taking advantage of the new addressing modes of the 80386 is a challenging problem for the compiler designer. The compiler must be able to plan ahead, keeping strategic values in registers for their subsequent use in complex addressing instructions.

New Long Jump On Condition Instructions

Example 8: New Long Jump On Condition Instructions

FUTURE OF THE INTEL 80x86 FAMILY

Intel has stated that the evolution of the 80x86 product line will occur in a truly upwardly compatible fashion with the 80386. Long range plans call for processors through the 80686 to be developed to improve performance only. Thus, programs written for the 80386 today will remain viable across the new Intel line for several years.

The 80486 remains faithful to this goal. Though it has added an onboard numeric coprocessor, there are no new processing modes requiring extensive code modifications to take full advantage of the chip. Programs written for the 80386 run with full power on the 80486.

At the present time, Microsoft Windows/386 provides the best example of a PC-type operating system running in 386 protected mode. The high performance Windows/386 implementation provides an appealing target for large scale applications which find themselves restrained by the memory and co-processing limitations of MS-DOS.
It seems inevitable that an 80386 native implementation of OS/2 will be introduced. It is only with the power of the 80386 that OS/2 will have its chance to demonstrate whether it can live up to its hype.

As far as UNIX-type operating systems, AIX 386, Sun386 UNIX and SCO UNIX System V/386 are examples of native 386 operating systems. These Workstation/PC environments seem positioned to challenge OS/2 for dominance of the 80386 operating system market.

CONCLUSION

When viewed as a 32-bit processor, able to directly address 4 gigabytes in a flat address space, the 80386 shows why it is well positioned to be the fundamental microprocessor design for a new generation of personal computers. Without the need for segmentation, large scale applications will move smoothly to these machines without the performance penalties associated with previous 80x86 generations.

The Institute's 80386 C compiler has been created with the SAS System in mind and promises exciting new implementations in the near future.

REFERENCES


SAS is a registered trademark of SAS Institute Inc, Cary, NC, USA.
Intel is a registered trademark of Intel Corporation, Santa Clara, CA, USA.
Microsoft is a registered trademark of Microsoft Corporation and Windows/386 is a trademark of Microsoft Corporation, Redmond, WA, USA.
AIX is a trademark of International Business Machines, Armonk, NY, USA.
Sun386 is a trademark of the Sun Corporation, Billerica, MA, USA.
SCO is a trademark of the Santa Cruz Operation, Santa Cruz, CA, USA.