About this Information 1

1.1 Notation Conventions

The following notations are used throughout this information set.

#	The pound symbol (#) appended to a signal name indicates that the signal is active low.
Variables	Variables are shown in italics. Variables must be replaced with correct values.
Instructions	Instruction mnemonics are shown in upper case. When you are programming, instructions are not case-sensitive. You may use either upper or lower case.
Numbers	Hexadecimal numbers are represented by a string of hexadecimal digits followed by the character H. A zero prefix is added to numbers that begin with A through F. (For example, FF is shown as 0FFH.) Decimal and binary numbers are represented by their customary notations. (That is, 255 is a decimal number and 1111 1111 is a binary number. In some cases, the letter B is added for clarity.)
Units of Measure	The following abbreviations are used to represent units of measure:

A	amps, amperes
mA	milliamps, milliamperes
µA	microamps, microamperes
Mbyte	megabytes
Kbyte	kilobytes
Gbyte	gigabyte
W	watts
KW	kilowatts
mW	milliwatts
µW	microwatts
MHz	megahertz
ms	milliseconds
ns	nanoseconds
µs	microseconds
µF	microfarads
pF	picofarads
V	volts

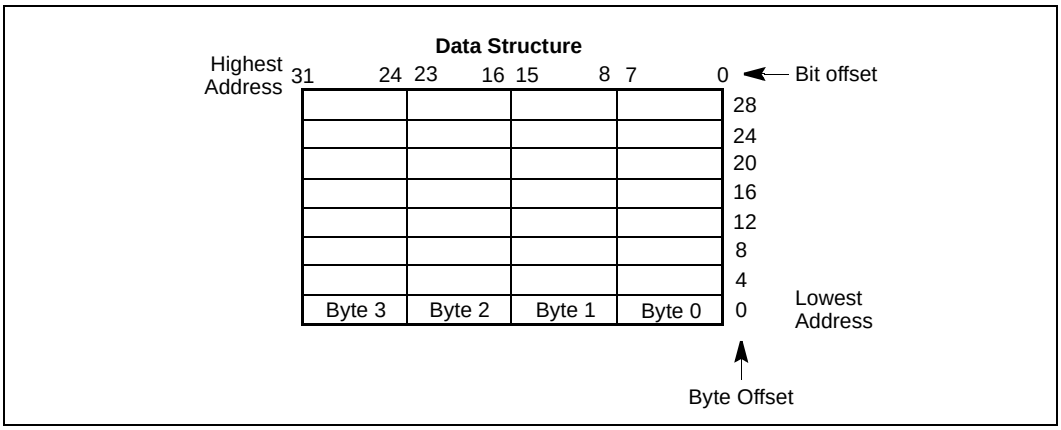
Bit and

Signal Ranges	When the text refers to a range of register bits or signals, the range is represented by the highest and lowest number, separated by a dash (example: A15–A8). For register bits, the first bit shown is the most-significant and the second bit shown is the least-significant.
Register Names	Register names are shown in upper case. When a register name contains a lower case, italic character, the name represents more than one register. For example, CR*n* represents these registers: CR0, CR1, CR2, etc.
Signal Names	Signal names are shown in upper case. A pound symbol (#) appended to a signal name identifies an active-low signal.

1.1.1 Bit and Byte Order

In illustrations of data structures in memory, smaller addresses appear toward the bottom of the figure; addresses increase toward the top. Bit positions are numbered from right to left. The numerical value of a set bit is equal to two raised to the power of the bit position. Intel Architecture processors is a “little endian” machines; this means the bytes of a word are numbered starting from the least significant byte. Figure 1-1 illustrates these conventions.

Figure 1-1. Bit and Byte Order



1.1.2 Reserved Bits and Software Compatibility

In many register and memory layout descriptions, certain bits are marked as **reserved**. When bits are marked as reserved, it is essential for compatibility with future processors that software treat these bits as having a future, though unknown, effect. The behavior of reserved bits should be regarded as not only undefined, but unpredictable. Software should follow these guidelines in dealing with reserved bits:

- Do not depend on the states of any reserved bits when testing the values of registers which contain such bits. Mask out the reserved bits before testing.
- Do not depend on the states of any reserved bits when storing to memory or to a register.
- Do not depend on the ability to retain information written into any reserved bits.
- When loading a register, always load the reserved bits with the values indicated in the documentation, if any, or reload them with values previously read from the same register.

Note: Avoid any software dependence upon the state of reserved bits in Intel Architecture registers. Depending upon the values of reserved register bits will make software dependent upon the unspecified manner in which the processor handles these bits. Depending upon reserved values risks incompatibility with future processors.

1.1.3 Instruction Operands

When instructions are represented symbolically, a subset of the Intel Architecture assembly language is used. In this subset, an instruction has the following format:

```
label: mnemonic argument1, argument2, argument3
```

where:

- A **label** is an identifier which is followed by a colon.
- A **mnemonic** is a reserved name for a class of instruction opcodes which have the same function.
- The operands *argument1*, *argument2*, and *argument3* are optional. There may be from zero to three operands, depending on the opcode. When present, they take the form of either literals or identifiers for data items. Operand identifiers are either reserved names of registers or are assumed to be assigned to data items declared in another part of the program (which may not be shown in the example).

When two operands are present in an arithmetic or logical instruction, the right operand is the source and the left operand is the destination.

For example:

```
LOADREG: MOV EAX, SUBTOTAL
```

In this example LOADREG is a label, MOV is the mnemonic identifier of an opcode, EAX is the destination operand, and SUBTOTAL is the source operand. Some assembly languages put the source and destination in reverse order.

1.1.4 Hexadecimal and Binary Numbers

Base 16 (hexadecimal) numbers are represented by a string of hexadecimal digits followed by the character H (for example, F82EH). A hexadecimal digit is a character from the following set: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

Base 2 (binary) numbers are represented by a string of 1s and 0s, sometimes followed by the character B (for example, 1010B). The “B” designation is only used in situations where confusion as to the type of number might arise.

1.1.5 Segmented Addressing

The processor uses byte addressing. This means memory is organized and accessed as a sequence of bytes. Whether one or more bytes are being accessed, a byte address is used to locate the byte or bytes memory. The range of memory that can be addressed is called an **address space**.

The processor also supports segmented addressing. This is a form of addressing where a program may have many independent address spaces, called **segments**. For example, a program can keep its code (instructions) and stack in separate segments. Code addresses would always refer to the code space, and stack addresses would always refer to the stack space. The following notation is used to specify a byte address within a segment:

Segment-register:Byte-address

For example, the following segment address identifies the byte at address FF79H in the segment pointed by the DS register:

```
DS:FF79H
```

The following segment address identifies an instruction address in the code segment. The CS register points to the code segment and the EIP register contains the address of the instruction.

```
CS:EIP
```

1.1.6 Exceptions

An exception is an event that typically occurs when an instruction causes an error. For example, an attempt to divide by zero generates an exception. However, some exceptions, such as breakpoints, occur under other conditions. Some types of exceptions may provide error codes. An error code reports additional information about the error. An example of the notation used to show an exception and error code is shown below.

```
#PF(fault code)
```

This example refers to a page-fault exception under conditions where an error code naming a type of fault is reported. Under some conditions, exceptions which produce error codes may not be able to report an accurate code. In this case, the error code is zero, as shown below for a general-protection exception.

```
#GP(0)
```

See Chapter 5, *Interrupt and Exception Handling*, in the *Intel Architecture Software Developer's Manual, Volume 3*, for a list of exception mnemonics and their descriptions.

1.2 Special Terminology

The general terms “processor,” “embedded Pentium processor,” and “embedded Pentium processor family” are used throughout this information set to refer to the embedded Pentium processor, the embedded Pentium processor with Voltage Reduction Technology, the embedded Pentium processor with MMX technology, and the low-power embedded Pentium processor with MMX technology together. Some of the features or functions described using these terms, however, may not be available on each processor type. Refer to the datasheet for each product to determine whether a specific feature is offered.

In some instances, the names “embedded Pentium processor,” “embedded Pentium processor with Voltage Reduction Technology,” “embedded Pentium processor with MMX technology,” and “low-power embedded Pentium processor with MMX technology” are used in this information set to distinguish between processors when specific differences exist.

See “Related Documents” on page 1-8 for a list of datasheets and other documents that describe the operation of Pentium processors.

The following terms have special meanings in this information set.

Assert and Deassert The terms assert and deassert refer to the acts of making a signal active and inactive, respectively. The active polarity (high/low) is defined by the signal name. Active-low signals are designated by the pound symbol (#) suffix; active-high signals have no suffix. To assert FLUSH# is to drive it low; to assert HOLD is to drive it high; to deassert FLUSH# is to drive it high; to deassert HOLD is to drive it low.

DOS I/O Address Peripherals that are compatible with PC/AT system architecture can be mapped into DOS (or PC/AT) addresses 0H–03FFH. In this information set, the terms *DOS address* and *PC/AT address* are synonymous.

Expanded I/O Address All peripheral registers reside at I/O addresses 0F000H–0FFFFH. PC/AT-compatible integrated peripherals can also be mapped into DOS (or PC/AT) address space (0H–03FFH).

PC/AT Address Integrated peripherals that are compatible with PC/AT system architecture can be mapped into PC/AT (or DOS) addresses 0H–03FFH. In this information set, the terms *DOS address* and *PC/AT address* are synonymous.

Set and Clear The terms set and clear refer to the value of a bit or the act of giving it a value. If a bit is set, its value is “1”; setting a bit gives it a “1” value. If a bit is clear, its value is “0”; clearing a bit gives it a “0” value.

1.3 Technical Support

1.3.0.1 Online Documents

Product documentation is provided online in a variety of web-friendly formats at:

http://developer.intel.com/design/litcentr/index.htm

1.3.0.2 Intel Product Forums

Intel provides technical expertise through electronic messaging. With publicly accessible forums, you have all of the benefits of email technical support, with the added benefit of the option of viewing previous messages written by other participants, and providing suggestions and tips that can help others.

Each of Intel’s technical support forums is based on a single product or product family. Questions and replies are limited to the topic of the particular forum. Intel also provides several non-technical support related forums.

Complete information on Intel forums is available at:

http://support.intel.com/newsgroups/index.htm.

1.3.1 Telephone Technical Support

In the U.S. and Canada, technical support representatives are available to answer your questions between 5 a.m. and 5 p.m. PST. You can also fax your questions to us. (Please include your voice telephone number and indicate whether you prefer a response by phone or by fax). Outside the U.S. and Canada, please contact your local distributor.

1-800-628-8686	U.S. and Canada
916-356-7599	U.S. and Canada
916-356-6100 (fax)	U.S. and Canada

1.4 Product Literature

You can order product literature from the following Intel literature centers.

1-800-548-4725	U.S. and Canada
708-296-9333	U.S. (from overseas)
44(0)1793-431155	Europe (U.K.)
44(0)1793-421333	Germany
44(0)1793-421777	France
81(0)120-47-88-32	Japan (fax only)