****

**MSPM’S**

**Deogiri Institute of Engineering and Management Studies, Aurangabad**

Report on

**HP Pavilion laptop 15-cs 2000tx**

Submitted By

**Name: Aniket Anil Yeola**

**Division: SE-I**

**Batch: A3**

**Roll No: 26038**

Under the Guidance of

**Prof.P.H.Durole**

Department of Computer Science and Engineering

(Deogiri Institute of Engineering and Management Studies)

2019-2020



CERTIFICATE

This is to certify that Mr. Aniket Anil Yeola Seat No.26038 has completed a report writing on HP Pavilion laptop in subject Computer Architecture and Organization.

**Prof.P.H.Durole Dr.S.B.Kalyankar Dr.Ulhas Shiurkar**

**Subject Teacher H.O.D. Director**

**Basic** **Information**

Hardware

|  |  |
| --- | --- |
| **Product number** | 5FP53PA |
| **Product name** | HP Pavilion 15-cs2000tx |
| **Microprocessor** | Intel® Core™ i5-8265U (1.6 GHz base frequency, up to 3.9 GHz with Intel® Turbo Boost Technology, 6 MB cache, 4 cores) |
| **Memory, standard** | 8 GB DDR4-2400 SDRAM (1 x 8 GB) |
| **Hard drive** | 1 TB 5400 rpm SATA |
| **Display** | 15.6" diagonal FHD SVA anti-glare micro-edge WLED-backlit (1920 x 1080) |
| **Keyboard** | Full-size island-style backlit keyboard with numeric keypad |
| **Pointing device** | HP Imagepad with multi-touch gesture support |
| **Wireless connectivity** | Intel® Wireless-AC 9560 802.11a/b/g/n/ac (2x2) Wi-Fi® and Bluetooth® 5 Combo |
| **Network interface** | Integrated 10/100/1000 GbE LAN |
| **Expansion slots** | 1 multi-format SD media card reader |
| **External ports** | 1 HDMI 1.4; 1 headphone/microphone combo; 1 RJ-45; 2 USB 3.1 Gen 1 (Data transfer only); 1 USB 3.1 Type-C™ Gen 1 (Data Transfer Only) |
| **Minimum dimensions (W x D x H)** | 36.16 x 24.16 x 1.79 cm |
| **Weight** | Starting at 1.85 kg |
| **Power supply type** | 65 W EM AC power adapter |
| **Battery type** | 3-cell, 41 Wh Li-ion |
| **Webcam** | HP Wide Vision HD Camera with integrated dual array digital microphone |
| **Audio features** | B&O PLAY, dual speakers, HP Audio Boost |

Software

|  |  |
| --- | --- |
| **Operating system** | Windows 10 Home Single Language 64 |
| **HP apps** | HP Audio Switch; HP Cool Sense; HP Documentation; HP e  Print; HP JumpStart;  HP Support Assistant; HP Connection Optimizer |
| **Software included** | McAfee LiveSafe™ |
| **Software - Productivity & finance** | Microsoft Office Home & Student 2019 |

**Introduction to x64 Assembly**



**Introduction**

For years, PC programmers used x86 assembly to write performance-critical code. However, 32-bit PCs are being replaced with 64-bit ones, and the underlying assembly code has changed. This Gem is an introduction to x64 assembly. No prior knowledge of x86 code is needed, although it makes the transition easier.

x64 is a generic name for the 64-bit extensions to Intel‟s and AMD‟s 32-bit x86 instruction set architecture (ISA). AMD introduced the first version of x64, initially called x86-64 and later renamed AMD64. Intel named their implementation IA-32e and then EMT64. There are some slight incompatibilities between the two versions, but most code works fine on both versions; details can be found in the AMD .We call this intersection flavor x64. Neither is to be confused with the 64-bit Intel® Itanium® architecture, which is called IA-64.

This Gem won‟t cover hardware details such as caches, branch prediction, and other advanced topics. Several references will be given at the end of the article for further reading in these areas.

Assembly is often used for performance-critical parts of a program, although it is difficult to outperform a good C++ compiler for most programmers. Assembly knowledge is useful for debugging code – sometimes a compiler makes incorrect assembly code and stepping through the code in a debugger helps locate the cause. Code optimizers sometimes make mistakes. Another use for assembly is interfacing with or fixing code for which you have no source code. Disassembly lets you change/fix existing executables. Assembly is necessary if you want to know how your language of choice works under the hood – why some things are slow and others are fast. Finally, assembly code knowledge is indispensable when diagnosing malware.

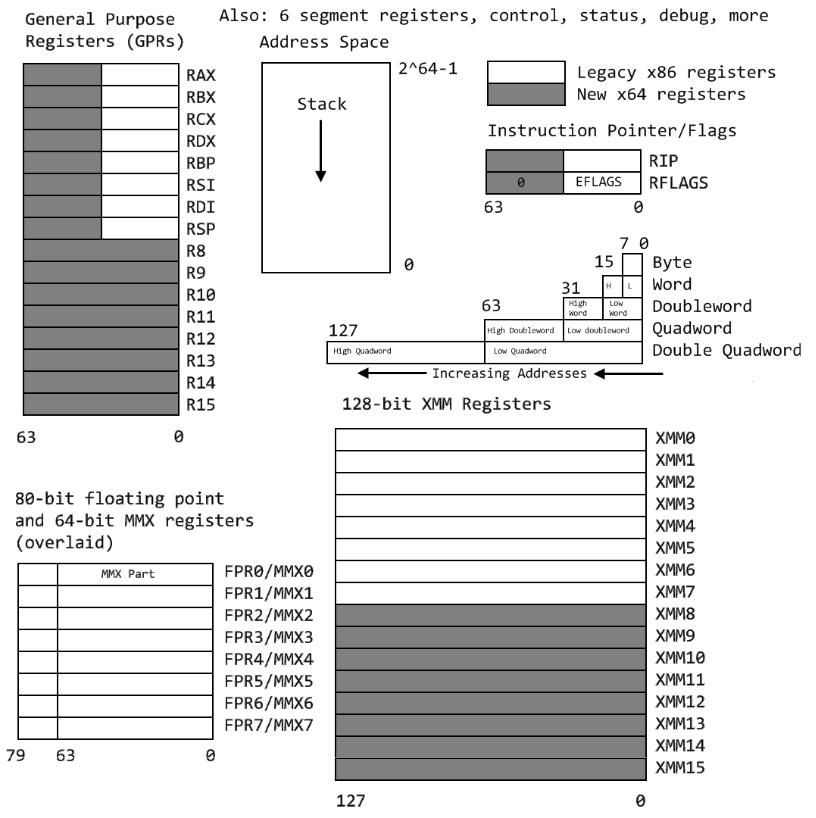
**Architecture**

When learning assembly for a given platform, the first place to start is to learn the register set.

**General Architecture**

Since the 64-bit registers allow access for many sizes and locations, we define a byte as 8 bits, a word as 16 bits, a double word as 32 bits, a quadword as 64 bits, and a double quadword as 128 bits. Intel stores bytes “little endian,” meaning lower significant bytes are stored in lower memory address.

[Figure 1](#page2) shows sixteen general purpose 64-bit registers, the first eight of which are labeled (for historical reasons) RAX, RBX, RCX, RDX, RBP, RSI, RDI, and RSP. The second eight are named R8-R15. By replacing the initial R with an E on the first eight registers, it is possible to access the lower 32 bits (EAX for RAX). Similarly, for RAX, RBX, RCX, and RDX, access to the lower 16 bits is possible by removing the initial R (AX for RAX), and the lower byte of the these by switching the X for L (AL for AX), and the higher byte of the low 16 bits using an H (AH for AX). The new registers R8 to R15 can be accessed in a similar manner like this: R8 (qword), R8D (lower dword), R8W (lowest word), R8B (lowest byte MASM style, Intel style R8L). Note there is no R8H.



**Figure 1 – General Architecture**

**SIMD Architecture**

Single Instruction Multiple Data (SIMD) instructions execute a single command on multiple pieces of data in parallel and are a common usage for assembly routines. MMX and SSE commands (using the MMX and XMM registers respectively) support SIMD operations, which perform an instruction on up to eight pieces of data in parallel. For example, eight bytes can be added to eight bytes in one instruction using MMX.

The eight 64-bit MMX registers MMX0-MMX7 are aliased on top of FPR0-7, which means any code mixing FP and MMX operations must be careful not to overwrite required values. The MMX instructions operate on integer types, allowing byte, word, and doubleword operations to be performed on values in the MMX registers in parallel. Most MMX instructions begin with „P‟ for “packed”. Arithmetic, shift/rotate, comparison, e.g.: PCMPGTB “Compare packed signed byte integers for greater than”.

The sixteen 128-bit XMM registers allow parallel operations on four single or two double precision values per instruction. Some instructions also work on packed byte, word, doubleword, and quadword integers. These instructions, called the Streaming SIMD Extensions (SSE), come in many flavors: SSE, SSE2, SSE3, SSSE3, SSE4, and perhaps more by the time this prints. Intel has announced more extensions along these lines called Intel® Advanced Vector Extensions (Intel® AVX), with a new 256-bit-wide datapath. SSE instructions contain move, arithmetic, comparison, shuffling and unpacking, and bitwise operations on both floating point and integer types. Instruction names include such beauties as PMULHUW and RSQRTPS. Finally, SSE introduced some instructions for memory pre-fetching (for performance) and memory fences (for multi-threaded safety).

[Table](#page5) lists some command sets, the register types operated on, the number of items manipulated in parallel, and the item type. For example, using SSE3 and the 128-bit XMM registers, you can operate on 2 (must be 64-bit) floating point values in parallel, or even 16 (must be byte sized) integer values in parallel.

To find which technologies a given chip supports, there is a CPUID instruction that returns processor-specific information.

**Table**

|  |  |  |  |
| --- | --- | --- | --- |
| Technology | Register size/type | Item type | Items in Parallel |
| MMX | 64 MMX | Integer | 8,4,2,1 |
| SSE | 64 MMX | Integer | 8,4,2,1 |
| SSE | 128 XMM | Float | 4 |
| SSE2/SSE3/SSSE3… | 64 MMX | Integer | 2,1 |
| SSE2/SSE3/SSSE3… | 128 XMM | Float | 2 |
| SSE2/SSE3/SSSE3… | 128 XMM | Integer | 16,8,4,2,1 |

**Instruction Basics**

**Addressing Modes**

Before covering some basic instructions, you need to understand addressing modes, which are ways an instruction can access registers or memory. The following are common addressing modes with examples:

* Immediate: the value is stored in the instruction.

**ADD EAX, 14 ; add 14 into 32-bit EAX**

* Register to register

**ADD R8L, AL** **; add 8 bit AL into R8L**

* Indirect: this allows using an 8, 16, or 32 bit displacement, any general purpose registers for base and index, and a scale of 1, 2, 4, or 8 to multiply the index. Technically, these

can also be prefixed with segment FS: or GS: but this is rarely required.

**MOV R8W, 1234[8\*RAX+RCX] ; move word at address 8\*RAX+RCX+1234 into R8W**

There are many legal ways to write this. The following are equivalent

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MOV** | **ECX, dword ptr table[RBX][RDI]** | | | |
| **MOV** | **ECX, dword ptr table[RDI][RBX]** | | | |
| **MOV** | **ECX,** | **dword** | **ptr** | **table[RBX+RDI]** |
| **MOV** | **ECX,** | **dword** | **ptr** | **[table+RBX+RDI]** |

The **dword ptr** tells the assembler how to encode the **MOV** instruction.

* RIP-relative addressing: this is new for x64 and allows accessing data tables and such in the code relative to the current instruction pointer, making position independent code

easier to implement.

**MOV AL, [RIP] ; RIP points to the next instruction aka NOP NOP**

Unfortunately, MASM does not allow this form of opcode, but other assemblers like

FASM and YASM do. Instead, MASM embeds RIP-relative addressing implicitly.

**MOV EAX, TABLE ; uses RIP- relative addressing to get table address**

* Specialized cases: some opcodes use registers in unique ways based on the opcode. For example, signed integer division **IDIV** on a 64 bit operand value divides the 128-bit value in **RDX:RAX** by the value, storing the result in **RAX** and the remainder in **RDX.**

**INSTRUCTIONS SET**

[Table](#page7) lists some common instructions. \* denotes this entry is multiple opcodes where the \* denotes a suffix.

**Table – Common Opcodes**

|  |  |  |  |
| --- | --- | --- | --- |
| Opcode | Meaning | Opcode | Meaning |
| MOV | Move to/from/between | AND/OR/XOR/NOT | Bitwise operations |
|  | memory and registers |  |  |
| CMOV\* | Various conditional moves | SHR/SAR | Shift right logical/arithmetic |
| XCHG | Exchange | SHL/SAL | Shift left logical/arithmetic |
| BSWAP | Byte swap | ROR/ROL | Rotate right/left |
| PUSH/POP | Stack usage | RCR/RCL | Rotate right/left through carry |
|  |  |  | bit |
| ADD/ADC | Add/with carry | BT/BTS/BTR | Bit test/and set/and reset |
| SUB/SBC | Subtract/with carry | JMP | Unconditional jump |
| MUL/IMUL | Multiply/unsigned | JE/JNE/JC/JNC/J\* | Jump if equal/not |
|  |  |  | equal/carry/not carry/ many |
|  |  |  | others |
| DIV/IDIV | Divide/unsigned | LOOP/LOOPE/LOOPNE | Loop with ECX |
| INC/DEC | Increment/Decrement | CALL/RET | Call subroutine/return |
| NEG | Negate | NOP | No operation |
| CMP | Compare | CPUID | CPU information |

A common instruction is the LOOP instruction, which decrements RCX, ECX, or CX depending on usage, and then jumps if the result is not 0. For example,

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **XOR** | **EAX, EAX** | **; zero out eax** | | | | |
| **MOV** | **ECX, 10** | **; loop** | | **10** | **times** | |
| **Label:** |  | **; this** | | **is** | **a** | **label in assembly** |
| **INX** | **EAX** | **;** | **increment** | | | **eax** |
| **LOOP** | **Label** | **;** | **decrement** | | | **ECX, loop if not 0** |

Less common op codes implement string operations, repeat instruction prefixes, port I/O instructions, flag set/clear/test, floating point operations (begin usually with a F, and support move, to/from integer, arithmetic, comparison, transcendental, algebraic, and control functions), cache and memory opcodes for multithreading and performance issues, and more. The Intel 64 Volume 2, in two parts, covers each opcode in detail.

**Random-access memory (RAM)**

RAM is a form of computer memory that can be read and changed in any order, typically used to store working data and machine code. A random-access memory device allows data items to be read or written in almost the same amount of time irrespective of the physical location of data inside the memory. In contrast, with other direct-access data storage media such as [hard disks](https://en.wikipedia.org/wiki/Hard_disk), [CD-RWs](https://en.wikipedia.org/wiki/CD-RW), DVD-RWs and the older [magnetic tapes](https://en.wikipedia.org/wiki/Magnetic_tape_data_storage) and [drum memory](https://en.wikipedia.org/wiki/Drum_memory), the time required to read and write data items varies significantly depending on their physical locations on the recording medium, due to mechanical limitations such as media rotation speeds and arm movement.

RAM contains [multiplexing](https://en.wikipedia.org/wiki/Multiplexer) and [demultiplexing](https://en.wikipedia.org/wiki/Demultiplexing) circuitry, to connect the data lines to the addressed storage for reading or writing the entry. Usually more than one bit of storage is accessed by the same address, and RAM devices often have multiple data lines and are said to be "8-bit" or "16-bit", etc. devices.

**Types**

The two widely used forms of modern RAM are [static RAM](https://en.wikipedia.org/wiki/Static_random_access_memory) (SRAM) and [dynamic RAM](https://en.wikipedia.org/wiki/Dynamic_random-access_memory) (DRAM). In SRAM, a [bit of data](https://en.wikipedia.org/wiki/Bit) is stored using the state of a six-transistor memory cell. This form of RAM is more expensive to produce, but is generally faster and requires less dynamic power than DRAM. In modern computers, SRAM is often used as cache memory for the CPU DRAM stores a bit of data using a transistor and capacitor pair, which together comprise a DRAM cell. The capacitor holds a high or low charge (1 or 0, respectively), and the transistor acts as a switch that lets the control circuitry on the chip read the capacitor's state of charge or change it. As this form of memory is less expensive to produce than static RAM, it is the predominant form of computer memory used in modern computers.

Both static and dynamic RAM are considered *volatile*, as their state is lost or reset when power is removed from the system. By contrast, read only memory (ROM) stores data by permanently enabling or disabling selected transistors, such that the cannot be altered. Writeable variants of ROM (such as EEPROM and flash memory) share properties of both ROM and RAM, enabling data to persist without power and to be updated without requiring special equipment. These persistent forms of semiconductor ROM include USB flash drives, memory cards for cameras and portable devices, and solid state drive. EEC memory (which can be either SRAM or DRAM) includes special circuitry to detect and/or correct random faults (memory errors) in the stored data, using partial bites or error correction codes.

In general, the term *RAM* refers solely to solid-state memory devices (either DRAM or SRAM), and more specifically the main memory in most computers.

**Read-only memory (ROM)**

ROM is a type of non-volatile memory used in computers and other electronic devices. Data stored in ROM cannot be electronically modified after the manufacture of the memory device. Read-only memory is useful for storing software that is rarely changed during the life of the system, sometimes known as [firmware](https://en.wikipedia.org/wiki/Firmware). Software applications for programmable devices can be distributed as [plug-in cartridges containing read-only memory](https://en.wikipedia.org/wiki/ROM_cartridge).

Erasable programmable read-only memory (EPROM) and electrically erasable programmable read-only memory (EEPROM) can be erased and re-programmed, but usually this can only be done at relatively slow speeds, may require special equipment to achieve, and is typically only possible a certain number of times.

****

**Solid-State Drive (SSD)**

A solid-state drive (SSD) is a solid-state storage device that uses integrated circuit assemblies as memory to store data persistently, typically using flash memory. It is also sometimes called a solid-state device or a solid-state disk, although SSDs lack the physical spinning disks and movable read-write heads used by conventional electromechanical storage such as hard drives. ("HDD") or floppy disks.

SSDs based on volatile memory such as DRAM are characterized by very fast data access, generally less than 10 microseconds, and are used primarily to accelerate applications that would otherwise be held back by the latency of flash SSDs or traditional HDDs.

The key components of an SSD are the controller and the memory to store the data. The primary memory component in an SSD was traditionally DRAM volatile memory, but since 2009 it is more commonly NAND flash non-volatile memory.

A flash-based SSD typically uses a small amount of DRAM as a volatile cache, similar to the buffers in hard disk drives. A directory of block placement and wear leveling data is also kept in the cache while the drive is operating. One SSD controller manufacturer, Sand Force , does not use an external DRAM cache on their designs but still achieves high performance. Such an elimination of the external DRAM reduces the power consumption and enables further size reduction of SSDs.

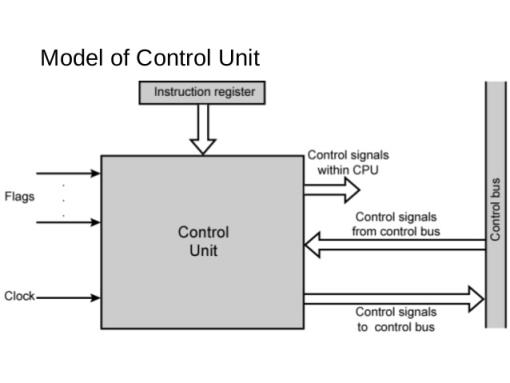
**Working of Control Unit**

The **control unit** (CU) is a component of a computer's central processing unit  (CPU) that directs the operation of the processor. It tells the computer's memory, arithmetic and logic unit and input and output devices how to respond to the instructions that have been sent to the processor.

The Control unit (CU) is digital circuitry contained within the processor that coordinates the sequence of data movements into, out of, and between a processor's many sub-units. The result of these routed data movements through various digital circuits (sub-units) within the processor produces the manipulated data expected by a software instruction (loaded earlier, likely from memory). It controls (conducts) data flow inside the processor and additionally provides several external control signals to the rest of the computer to further direct data and instructions to/from processor external destinations (i.e. memory).

The Control Unit (CU) is generally a sizable collection of complex digital circuitry interconnecting and directing the many execution units (i.e. ALU, data buffers, registers) contained within a CPU. The CU is normally the first CPU unit to accept from an externally stored computer program a single instruction (based on the CPU's instruction set). The CU then decodes this individual instruction into several sequential steps (fetching addresses/data from registers/memory, managing execution ([i.e. data sent to the ALU or I/O]), and storing the resulting data back into registers/memory) that controls and coordinates the CPU's inner works to properly manipulate the data. The design of these sequential steps is based on the needs of each instruction and can range in number of steps, the order of execution, and which units are enabled.

Thus by only using a program of set instructions in memory, the CU will configure all the CPU's data flows as needed to manipulate the data correctly between instructions. This result in a computer that could run a complete program and require no human intervention to make hardware changes between instructions (as had to be done when using only punch cards for computations before stored programmed computers with CUs were invented). These detailed steps from the CU dictate which of the CPU's interconnecting hardware control signals to enable/disable or which CPU units are selected/de-selected and the unit's proper order of execution as required by the instruction's operation to produce the desired manipulated data. Additionally, the CU's orderly hardware coordination properly sequences these control signals, then configures the many hardware units comprising the CPU, directing how data should also be moved, changed, and stored outside the CPU (i.e. memory) according to the instruction's objective.

****

**Input and Output Mechanism**

The process of giving input to computer and giving output from computer is called input/ output. The mechanism almost same for input and output. The operating system is mainly responsible for input output operating interrupt and error handling is important terms related to input/outputs.

I/O devices are divided into two categories:-

1.**Block devices**: -  A block devices is one that store information in fixed-sized blocks, each one, with its own address common blocked size ranges from 512 bytes to 32768 bytes. The essential property of a block device is that it is possible to read or write each block independently of all the other ones. In other word, at any instant, the program can read or write any of the blocks. The common examples of block device are disk. A disk is block addressable device because no matter where the arm currently is, it is always possible to seek to another cylinder and then wait for another block to rotate the head.

2. **Character devices**: -  A character device is one that delivers or accepts a stream of characters, without regards to any blocks structure. It is not accessible and does not have any such operation. The examples of character devices are printers, paper tapes, network interface card, mice and most other devices that are not disk like can be seen as.

**Device controller**

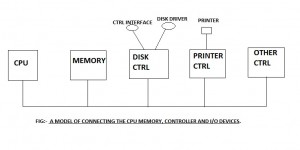
I/O units typically consist of mechanical part and the electronic part. The electronic part is also called the device controller or adapter. On pc, device controller takes the form of printed circuit card that can be inserted into an expansion slots. The controller card actually has a connected on it, into which a cable leading to the device itself can be plugged many controllers can handle more than one identical devices. The standard for interface between controller and device are ANSI, ICE, IDE, SCSI, ISO etc.

The interface between the controller and device is often a very low level interface. The controller job is to convert the serial bit stream into a block of bytes and perform any error. Correction if necessary the block of bytes is typically first assembled, bit by bit in a buffer inside the controller. After its checksum has been verified and block declared to be error free, it can then be copied to main memory.

Each controller has some registers for communicating with CPU and many devices have data buffer, which the CPU can read and write data. The issues that arise of how the CPU communicates with the controller registers and the device data buffer has two alternatives.

**I/o mupped I/o:-**  in this approach, each control register is assign an i/o port number and 8 bits or 16 bits integer. The scheme uses I/O instruction for I/O such as in OUT PORT, REG (CPU register).

**Memory mupped I/O:**- in this approach, all the control register are mapped into the memory space. Each control register is assigned a unique memory address to which no memory is assigned. Usually, the assigned address is at the top of the address space. In this approach memory instruction like mov, stor, load are used.



**Conclusion**

This has been a necessarily brief introduction to x64 assembly programming. The next step is to browse the [Intel® 64 and IA-32 Architectures Software Developer‟s Manuals.](http://developer.intel.com/products/processor/manuals/index.htm) Volume 1 contains the architecture details and is a good start if you know assembly. Other places are assembly books or online assembly tutorials. To get an understanding of how your code executes, it is instructive to step through code in debugger, looking at the disassembly, until you can read assembly code as well as your favorite language. For C/C++ compilers, debug builds are much easier to read than release builds so be sure to start there. Finally, read [the forums at](http://www.masm32.com/board/index.php) [masm32.com](http://www.masm32.com/board/index.php) for a lot of material.

**References**

“AMD64 Architecture Tech Docs,” available online at <http://www.amd.com/us-en/Processors/DevelopWithAMD/0,,30_2252_875_7044,00.html>

“Intel® 64 and IA-32 Architectures Software Developer's Manuals,” available online at <http://www.intel.com/products/processor/manuals/>