# Introduction to Microprocessors and Assembly Language

Fairchild Semiconductor, founded in 1957, invented the first IC in 1959. In 1968, Robert Noyce, Gordan Moore, and Andrew Grove resigned from Fairchild Semiconductor. They launched their own company, Intel (Integrated Electronics). Intel grew from a 3-man start-up in 1968.

## Microprocessors

Microprocessors are program-controlled semiconductor devices (IC) that fetch (from memory), decode, and execute instructions. It is used as a CPU (Central Processing Unit) in computers. A microprocessor consists of the processor, the ALU (Arithmetic Logic Unit), the Instruction Decoder, and a few registers that store data for mathematical and logical operations. E.g., 8085, 8086.

At the heart of the microprocessor-based computer system is the microprocessor integrated circuit. The microprocessor sometimes called the CPU (central processing unit), is the controlling element in a computer system. The microprocessor controls memory and I/O through a series of connections called buses. The buses select an I/O or memory device, transfer data between an I/O device or memory and the microprocessor, and control the I/O and memory system. Memory and I/O are controlled through instructions stored in the memory and executed by the microprocessor.

The microprocessor performs three main tasks for the computer system: (1) data transfer between itself and the memory or I/O systems, (2) simple arithmetic and logic operations, and (3) program flow via simple decisions. Although these are simple tasks, it is through them that the microprocessor performs virtually any series of operations or tasks. The microprocessor's power is in its capability to execute billions of millions of instructions per second from a program or software (group of instructions) stored in the memory system. This stored program concept has made the microprocessor and computer system very powerful. Babbage wanted to use this stored program concept in his Analytical Engine.

## Assembly Language

The operations of the computer's hardware are controlled by its software. When the computer is on, it is always in the process of executing instructions. To fully understand the computer's operations, we must also study its instructions.

A CPU can only execute machine language instructions. As we've seen, they are bit strings. The following is a short machine language program for the IBM PC:

| Machine instruction | Operation |
| --- | --- |
| 10100001 00000000 00000000 | Fetch the contents of memory word 0 and put it in register AX. |
| 00000101 00000100 00000000 | Add 4 to AX. |
| 10100011 00000000 00000000 | Store the contents of AX in memory word 0. |

A more convenient language to use is assembly language. In assembly language, we use symbolic names to represent operations, registers, and memory locations. If location 0 is symbolized by A, the preceding program expressed in IBM PC assembly language would look like this:

| Assembly language instruction | Comment |
| --- | --- |
| MOV AX, A | Fetch the contents of location A and put it in register AX. |
| ADD AX, 4 | Add 4 to AX |
| MOV A, AX | Move the contents of AX into location A. |

A program written in assembly language must be converted to machine language before the CPU can execute it. A program called the assembler translates each assembly language statement into a single machine language instruction.

# Number Systems and Data Representation

*Key points*

* The mechanical computer age began with the advent of the abacus in 500 B.C. This first mechanical calculator remained unchanged until 1642, when Blaise Pascal improved it. An early mechanical computer system was the Analytical Engine developed by Charles Babbage in 1823. Unfortunately, this machine never functioned because of the inability to create the necessary machine parts.
* The first electronic calculating machine was developed during World War II by Konrad Zuse, an early pioneer of digital electronics. His computer, the Z3, was used in aircraft and missile design for the German war effort.
* The first electronic computer, which used vacuum tubes, was placed into operation in 1943 to break secret German military codes. This first electronic computer system, the Colossus, was invented by Alan Turing. Its only problem was that the program was fixed and could not be changed.
* The first general-purpose, programmable electronic computer system was developed in 1946 at the University of Pennsylvania. This first modern computer was called the ENIAC (Electronics Numerical Integrator and Calculator).
* The first high-level programming language, called FLOWMATIC, was developed for the UNIVAC I computer by Grace Hopper in the early 1950s. This led to FORTRAN and other early programming languages such as COBOL.
* The world’s first microprocessor, the Intel 4004, was a 4-bit microprocessor—a programmable controller on a chip—that was meager by today’s standards. It addressed a mere 4096 4-bit memory locations. Its instruction set contained only 45 different instructions.
* Microprocessors that are common today include the 8086/8088, which were the first 16-bit microprocessors. Following these early 16-bit machines were the 80286, 80386, 80486, Pentium, Pentium Pro, Pentium II, Pentium III, Pentium 4, and Core2 processors. The architecture has changed from 16 bits to 32 bits and, with the Itanium, to 64 bits. With each newer version, improvements followed that increased the processor’s speed and performance. From all indications, this process of speed and performance improvement will continue, although the performance increases may not always come from an increased clock frequency.
* The DOS-based personal computers contain memory systems that include three main areas: TPA (transient program area), system area, and extended memory. The TPA hold: application programs, the operating system, and drivers. The system area contains memory used for video display cards, disk drives, and the BIOS ROM. The extended memory area is only available to the 80286 through the Core2 microprocessor in an AT-style or ATX-style personal computer system. The Windows-based personal computers contain memory systems that include two main areas: TPA and systems area.
* The 8086/8088 address 1M byte of memory from locations 00000H–FFFFFH. The 80286 and 80386SX address 16M bytes of memory from locations 000000H–FFFFFFH. The 80386SL addresses 32M bytes of memory from locations 0000000H–1FFFFFFH. The 80386DX through the Core2 address 4G bytes of memory from locations 00000000H–FFFFFFFFH. In addition, the Pentium Pro through the Core2 can operate with a 36-bit address and access up to 64G bytes of memory from locations 000000000H–FFFFFFFFFH. A Pentium 4 or Core2 operating with 64-bit extensions addresses memory from locations 0000000000H– FFFFFFFFFFH for 1T byte of memory.
* All versions of the 8086 through the Core2 microprocessors address 64K bytes of I/O address space. These I/O ports are numbered from 0000H to FFFFH with I/O ports 0000H–03FFH reserved for use by the personal computer system. The PCI bus allows ports 0400H–FFFFH.
* The operating system in early personal computers was either MSDOS (Microsoft disk operating system) or PCDOS (personal computer disk operating system from IBM). The operating system performs the task of operating or controlling the computer system, along with its I/O devices. Modern computers use Microsoft Windows in place of DOS as an operating system.
* The microprocessor is the controlling element in a computer system. The microprocessor performs data transfers, does simple arithmetic and logic operations, and makes simple decisions. The microprocessor executes programs stored in the memory system to perform complex operations in short periods of time.
* All computer systems contain three buses to control memory and I/O. The address bus is used to request a memory location or I/O device. The data bus transfers data between the microprocessor and its memory and I/O spaces. The control bus controls the memory and I/O, and requests reading or writing of data. Control is accomplished with (I/O read control), (I/O write control), (memory read control), and (memory write control).
* Numbers are converted from any number base to decimal by noting the weights of each position. The weight of the position to the left of the radix point is always the units position in any number system. The position to the left of the units position is always the radix times one. Succeeding positions are determined by multiplying by the radix. The weight of the position to the right of the radix point is always determined by dividing by the radix.
* Conversion from a whole decimal number to any other base is accomplished by dividing by the radix. Conversion from a fractional decimal number is accomplished by multiplying by the radix.
* Hexadecimal data are represented in hexadecimal form or in a code called binary-coded hexadecimal (BCH). A binary-coded hexadecimal number is one that is written with a 4-bit binary number that represents each hexadecimal digit.
* The ASCII code is used to store alphabetic or numeric data. The ASCII code is a 7-bit code; it can have an eighth bit that is used to extend the character set from 128 codes to 256 codes. The carriage return (Enter) code returns the print head or cursor to the left margin. The line feed code moves the cursor or print head down one line. Most modern applications use Unicode, which contains ASCII at codes 0000H–00FFH.
* Binary-coded decimal (BCD) data are sometimes used in a computer system to store decimal data. These data are stored either in packed (two digits per byte) or unpacked (one digit per byte) form.
* Binary data are stored as a byte (8 bits), word (16 bits), or doubleword (32 bits) in a computer system. These data may be unsigned or signed. Signed negative data are always stored in the two’s complement form. Data that are wider than 8 bits are always stored using the little endian format. In 32-bit Visual C++ these data are represented with char (8 bits), short (16 bits) and int (32 bits).
* Floating-point data are used in computer systems to store whole, mixed, and fractional numbers. A floating-point number is composed of a sign, a mantissa, and an exponent.
* The assembler directives DB or BYTE define bytes, DW or WORD define words, DD or DWORD define doublewords, and DQ or QWORD define quadwords.

# Memory Systems

*Key points*

* All memory devices have address inputs; data inputs and outputs, or just outputs; a pin for selection; and one or more pins that control the operation of the memory.
* Address connections on a memory component are used to select one of the memory locations within the device. Ten address pins have 1024 combinations and therefore are able to address 1024 different memory locations.
* Data connections on a memory are used to enter information to be stored in a memory location and also to retrieve information read from a memory location. Manufacturers list their memory as, for example, 4K × 4, which means that the device has 4K memory locations (4096) and that four bits are stored in each location.
* Memory selection is accomplished via a chip selection pin () on many RAMs or a chip enable pin () on many EPROM or ROM memories.
* Memory function is selected by an output enable pin () for reading data, which normally connects to the system read signal ( or ). The write enable pin (), for writing data, normally connects to the system write signal ( or ).
* An EPROM memory is programmed by an EPROM programmer and can be erased if exposed to ultraviolet light. Today, EPROMs are available in sizes from 1K × 8 all the way up to 512K × 8 and larger.
* The flash memory (EEPROM) is programmed in the system by using a 12 V or 5.0 V programming pulse.
* Static RAM (SRAM) retains data for as long as the system power supply is attached. These memory types are available in sizes up to 128K × 8.
* Dynamic RAM (DRAM) retains data for only a short period, usually 2–4 ms. This creates problems for the memory system designer because the DRAM must be refreshed periodically. DRAMs also have multiplexed address inputs that require an external multiplexer to provide each half of the address at the appropriate time.
* Memory address decoders select an EPROM or RAM at a particular area of the memory. Commonly found address decoders include the 74LS138 3-to-8 line decoder, the 74LS139 2-to-4 line decoder, and programmed selection logic in the form of a PLD.
* The PLD address decoder for microprocessors like the 8088 through the Pentium 4 reduce the number of integrated circuits required to complete a functioning memory system.
* The 8088 minimum mode memory interface contains 20 address lines, eight data lines, and three control lines: , , and IO/. The 8088 memory functions correctly only when all these lines are used for memory interface.
* The access speed of the EPROM must be compatible with the microprocessor to which it is interfaced. Many EPROMs available today have an access time of 450 ns, which is too slow for the 5 MHz 8088. In order to circumvent this problem, a wait state is inserted to increase memory access time to 660 ns.
* Error-correction features are also available for memory systems, but these require the storage of many more bits. If an 8-bit number is stored with an error-correction circuit, it actually takes 13 bits of memory: five for an error checking code and eight for the data. Most error-correction integrated circuits are able to correct only a single-bit error.
* The 8086/80286/80386SX memory interface has a 16-bit data bus and contains an M/ control pin, whereas the 8088 has an 8-bit data bus and contains an IO/ pin. In addition to these changes, there is an extra control signal, bus high enable ().
* The 8086/80386/80386SX memory is organized in two 8-bit banks: high bank and low bank. The high bank of memory is enabled by the control signal and the low bank is enabled by the A0 address signal or by the control signal.
* Two common schemes for selecting the banks in an 8086/80286/80386SX-based system include (1) a separate decoder for each bank and (2) separate control signals for each bank with a common decoder.
* Memory interfaced to the 80386DX and 80486 is 32 bits wide, as selected by a 32-bit address bus. Because of the width of this memory, it is organized in four memory banks that are each 8 bits wide. Bank selection signals are provided by the microprocessor as , , , and .
* Memory interfaced to the Pentium–Core2 is 64 bits wide, as selected by a 32-bit address bus. Because of the width of the memory, it is organized in eight banks that are each 8 bits wide. Bank selection signals are provided by the microprocessor as -.
* Dynamic RAM controllers are designed to control DRAM memory components. Many DRAM controllers today are built into the chip set and contain address multiplexers, refresh counters, and the circuitry required to do a periodic DRAM memory refresh.

# Assembly Language Programming

Assembly language programs are translated into machine language instructions by an assembler, so they must be written to conform to the assembler's specifications. We use the Microsoft Macro Assembler (MASM). Assembly language code is generally not case sensitive.

## Statements

Programs consist of statements, one per line. Each statement is either an instruction, which the assembler translates into machine code, or an assembler directive, which instructs the assembler to perform some specific task, such as allocating memory space for a variable or creating a procedure. Both instructions and directives have up to four fields:

name operation operand(s) comment

At least one blank or tab character must separate the· fields. The fields do not have to be aligned in a particular column, but they must appear in the above order. An example of an instruction is:

START: MOV CX, 5 ;initialize counter

Here, the name field consists of the label START:. The operation is MOV, the operands are CX and 5, and the comment is ;initialize counter. An example of an assembler directive is:

MAIN PROC

MAIN is the name, and the operation field contains PROC. This particular directive creates a procedure called MAIN.

*Key points*

1. Assembly language programs are made up of statements. A statement is either an instruction to be executed by the computer, or a directive for the assembler.
2. Statements have name, operation, operand(s), and comment fields.
3. A symbolic name can contain up to 31 characters. The characters can be letters, digits, and certain special symbols.
4. Numbers may be written in binary, decimal, or hex.
5. Characters and character strings must be enclosed in single or double quotes.
6. Directives on and DW are used to define byte and word variables, respectively. EQU can be used to give names to constants.
7. A program generally contains a code segment, a data segment, and a stack segment.
8. MOV and XCHG are used to transfer data. There are some restrictions for the use of these instructions; for example, they may not operate directly between memory locations.
9. ADD, SUB, INC, DEC, and NEG are some of the basic arithmetic instructions.
10. There are two ways to do input and output on the IBM PC: (1) by direct communication with 1/0 devices, (2) by using BIOS or DOS interrupt routines.
11. The direct method is fastest, but difficult to program and depends on specific hardware circuits. . ' Input and output of characters and strings may be done by the DOS routine INT 21h.
12. INT21h, function 1, causes a keyboard character to be read into AL.
13. INT21h, function 2, causes the character whose ASCII code Is in DL to be displayed. If DL contains the code of a control character, the control function is performed:
14. INT 21h, function 9, causes the string whose offset address is in DX to be displayed. The string must end with a "S" character.

# Registers, Flags, and Instruction Set Architecture

Registers are small, fast storage locations within the CPU used to hold temporary data and instructions. Flags are status various bits. Instruction set architecture is the interface between hardware and software.

1. The programming model of the 8086 through 80286 contains 8- and 16-bit registers. The programming model of the 80386 and above contains 8-, 16-, and 32-bit extended registers as well as two additional 16-bit segment registers: FS and GS.
2. The 8-bit registers are AH, AL, BH, BL, CH, CL, DH, and DL. The 16-bit registers are AX, BX, CX, DX, SP, BP, DI, and SI. The segment registers are CS, DS, ES, SS, FS, and GS. The 32-bit extended registers are EAX, EBX, ECX, EDX, ESP, EBP, EDI, and ESI. The 64-bit registers in a Pentium 4 with 64-bit extensions are RAX, RBX, RCX, RDX, RSP, RBP, RDI, RSI, and R8 through R15. In addition, the microprocessor contains an instruction pointer (IP/EIP/RIP) and flag register (FLAGS, EFLAGS, or RFLAGS).

# Assembly Language Programming Techniques

This topic explores advanced methods of programming in assembly, including loop structures, function calls, stack manipulation, and efficient use of registers.

## Flow Control Instructions

1. The jump instructions may be divided into unconditional and conditional jumps. The conditional jumps may be classified as signed, unsigned, and single-flag jumps.
2. The conditional jumps operate on the settings of the status flags. The CMP (compare} instruction is often used to set the flags just before a jump instruction.
3. The destination label of a conditional jump must be less than 126 bytes before or 127 bytes after the jump. A JMP can often be used to get around this restriction.
4. In an IF THEN decision structure, If the test condition is true, then the true-branch statements are done; otherwise, the next statement in line is done.
5. In an IF-THEN-ELSE decision structure, if the test condition is true, then the true-branch statements are done; otherwise, the false-branch statements are done. A JMP must follow the true-branch statements so that the false-branch will be bypassed.
6. In a CASE structure, branching is controlled by an expression; the branches correspond to the possible values of the expression.
7. A FOR loop is executed a known number of times. It may be implemented by the LOOP instruction. Before entering the loop, CX is initialized to the number of times to repeat the loop statements.
8. In a WHILE loop, the loop condition Is checked at the top of the loop. The loop statements are repeated as long as the condition is true. If the condition is initially false, the loop statements are not done at all.
9. In a REPEAT loop, the loop condition is checked at the bottom of the loop. The statements are repeated until the condition is true. Because the condition is checked at the bottom of the loop, the statements are done at least once.

## The Stack and Introduction to Procedures

1. The stack is a temporary storage area used by both application programs and the operating system.
2. The stack is a last-in, first-out data structure. SS:SP points to the top of the stack.
3. The stack-altering instructions are PUSH, PUSHF, POP. and POPF. PUSH adds a new top word to the stack, and POP removes the top word. PUSHF saves the FLAGS register on the stack and POPF puts the stack top into the FLAGS register.
4. SP decreases by 2 when PUSH or PUSHF is executed, and it increases by 2 when POP or POPF is executed. SP is initialized to the first word after stack segment when the program is loaded.
5. A procedure is a subprogram. Assembly language programs are typically broken into two procedures. One of the procedures is the main procedure, which contains the entry point to the program. Procedures may call other procedures, or themselves.
6. There are two kinds of procedures, NEAR and FAR. A NEAR procedure is in the same code segment as the calling program, and a FAR procedure is in a different segment.
7. The CALL instruction is used to invoke a procedure. For a NEAR procedure, execution of CALL causes the offset address of the next instruction in line after the CALL to be saved on the stack, and the IP gets the offset of the first Instruction in the procedure.
8. Procedures end with a RET instruction. Its execution causes the stack to be popped into IP and control returns to the calling program. In order for the return address to be accessible. the procedure must ensure that it is at the top of the stack when RET is executed.
9. ln assembly language, procedures often pass data through registers.

# Interrupts, Exceptions, and Peripheral Interfacing

Interrupts and exceptions are mechanisms by which the processor can be alerted to and respectfully handle events or errors that require immediate attention. Peripheral Interfacing involves connecting and communicating with external devices like keyboards, displays, and sensors.

## Hardware Interrupt

The notion of interrupt originally was conceived to allow hardware devices to interrupt the operation of the CPU. For example, whenever a key is pressed, the 8086 must be notified to read a key code into the keyboard buffer. The general hardware interrupt goes like this: (1) a hardware that needs service sends an interrupt request signal to the processor; (2) the 8086 suspends the current task it is executing and transfers control to an interrupt routine; (3) the interrupt routine services the hardware device by performing some I/O operation; and (4) control is transferred back to the original executing task at the point where it was suspended.

## Software Interrupt

Software interrupts are used by programs to request system services. A software interrupt occurs when a program calls an interrupt routine using the INT instruction. The format of the INT instruction is

INT interrupt-number

The 8086 treats this interrupt number in the same way as the interrupt number generated by a hardware device.

## Processor Exception

There is a third kind of interrupt, called a processor exception. A processor exception occurs when a condition arises inside the processor, such as divide overflow, that requires special handling. Each condition corresponds to a unique interrupt type. For example, divide overflow Is type 0, so when overflow occurs in a divide instruction the 8086 automatically executes interrupt 0 to handle the overflow condition.

## The Peripheral Component Interconnect (PCI) Bus

1. The bus systems (ISA, PCI, and USB) allow I/O and memory systems to be interfaced to the personal computer.
2. The ISA bus is either 8 or 16 bits, and supports either memory or I/O transfers at rates of 8 MHz.
3. The PCI (peripheral component interconnect) supports 32- or 64-bit transfers between the personal computer and memory or I/O at rates of 33 MHz. This bus also allows virtually any microprocessor to be interfaced to the PCI bus via the use of a bridge interface.
4. The PCI Express bus found on most computers is in the form of single lane or 16-lane ports. The single lane port is interfaced to I/O devices, whereas the 16-lane port is interfaced to the video card replacing AGP.
5. A plug-and-play (PnP) interface is one that contains a memory that holds configuration information for the system.
6. The parallel port called LPT1 is used to transfer 8-bit data in parallel to printers and other devices.
7. The serial COM ports are used for serial data transfer. The Windows API is used in a Windows Visual C++ application to effect serial data transfer through the COM ports.
8. The universal serial bus (USB) has all but replaced the ISA bus in the most advanced systems. The USB has three data transfer rates: 1.5 Mbps, 12 Mbps, and 480 Mbps.
9. The USB uses the NRZI system to encode data, and uses bit stuffing for logic 1 transmission more than 6 bits long.
10. The accelerated graphics port (AGP) is a high-speed connection between the memory system and the video graphics card.

# Assembly Language Debugging and Optimization

Debugging and optimizing assembly language code is a critical and detailed process that requires a deep understanding of computer architecture, instruction sets, and low-level programming.

## Debugging

1. Understanding the Architecture: Know the specific processor architecture (e.g., x86, ARM) you are working with, as assembly language is highly architecture-specific.
2. Using Debugging Tools: Utilize tools like GDB (GNU Debugger) for x86 assembly or specific tools for other architectures. These tools can step through code, set breakpoints, and inspect memory and registers.
3. Reading the Assembly Code Carefully: Look for common errors like incorrect instruction usage, addressing modes, register misuse, and off-by-one errors.
4. Checking Data Types and Sizes: Ensure that operations are performed on the correct data types and sizes, as assembly does not have the type safety of higher-level languages.
5. Monitoring Flags and Registers: Keep an eye on the status flags and content of registers, as they often hold clues to issues in the code.
6. Testing in Isolation: Test small parts of the code separately to isolate errors.

## Optimization

1. Efficient Instruction Usage: Choose instructions wisely to minimize the number of instructions and, thus, the execution time. For example, using shift operations for multiplication or division by powers of two.
2. Loop Optimization: Minimize the overhead in loops, unroll loops where beneficial, and optimize loop conditions and counters.
3. Minimizing Memory Access: Access to memory is slower than register access, so optimize the use of registers to minimize memory operations.
4. Parallelism and Pipelining: If the architecture supports it, use parallelism and take advantage of instruction pipelining.
5. Reducing Branching: Minimize branching instructions (like jumps) as they can disrupt the flow of the instruction pipeline in the CPU.
6. Profile-Guided Optimization: Use profiling tools to identify bottlenecks in the code and focus optimization efforts there.
7. Hardware-Specific Features: Utilize specific features of the hardware like SIMD (Single Instruction, Multiple Data) instructions if available.
8. Avoiding Redundant Operations: Identify and eliminate redundant calculations or memory accesses.

# Microprocessor Architecture and Organization

*Key points*

* All real mode memory addresses are a combination of a segment address plus an offset address. The starting location of a segment is defined by the 16-bit number in the segment register that is appended with a hexadecimal zero at its rightmost end. The offset address is a 16-bit number added to the 20-bit segment address to form the real mode memory address.
* All instructions (code) are accessed by the combination of CS (segment address) plus IP or EIP (offset address).
* Data are normally referenced through a combination of the DS (data segment) and either an offset address or the contents of a register that contains the offset address. The 8086–Core2 use BX, DI, and SI as default offset registers for data if 16-bit registers are selected. The 80386 and above can use the 32-bit registers EAX, EBX, ECX, EDX, EDI, and ESI as default offset registers for data.
* Protected mode operation allows memory above the first 1M byte to be accessed by the 80286 through the Core2 microprocessors. This extended memory system (XMS) is accessed via a segment address plus an offset address, just as in the real mode. The difference is that the segment address is not held in the segment register. In the protected mode, the segment starting address is stored in a descriptor that is selected by the segment register.
* A protected mode descriptor contains a base address, limit, and access rights byte. The base address locates the starting address of the memory segment; the limit defines the last location of the segment. The access rights byte defines how the memory segment is accessed via a program. The 80286 microprocessor allows a memory segment to start at any of its 16M bytes of memory using a 24-bit base address. The 80386 and above allow a memory segment to begin at any of its 4G bytes of memory using a 32-bit base address. The limit is a 16-bit number in the 80286 and a 20-bit number in the 80386 and above. This allows an 80286 memory segment limit of 64K bytes, and an 80386 and above memory segment limit of either 1M bytes (G = 0) or 4G bytes (G = 1). The L bit selects 64-bit address operation in the code descriptor.
* The segment register contains three fields of information in the protected mode. The leftmost 13 bits of the segment register address one of 8192 descriptors from a descriptor table. The TI bit accesses either the global descriptor table (TI = 0) or the local descriptor table (TI = 1). The rightmost 2 bits of the segment register select the requested priority level for the memory segment access.
* The program-invisible registers are used by the 80286 and above to access the descriptor tables. Each segment register contains a cache portion that is used in protected mode to hold the base address, limit, and access rights acquired from a descriptor. The cache allows the microprocessor to access the memory segment without again referring to the descriptor table until the segment register’s contents are changed.
* A memory page is 4K bytes in length. The linear address, as generated by a program, can be mapped to any physical address through the paging mechanism found within the 80386 through the Pentium 4 microprocessor.
* Memory paging is accomplished through control registers CR0 and CR3. The PG bit of CR0 enables paging, and the contents of CR3 addresses the page directory. The page directory contains up to 1024 page table addresses that are used to access paging tables. The page table contains 1024 entries that locate the physical address of a 4K-byte memory page.
* The TLB (translation look-aside buffer) caches the 32 most recent page table translations. This precludes page table translation if the translation resides in the TLB, speeding the execution of the software.
* The flat mode memory contains 1T byte of memory using a 40-bit address. In the future, Intel plans to increase the address width to 52 bits to access 4P bytes of memory. The flat mode is only available in the Pentium 4 and Core2 that have their 64-bit extensions enabled.

# References

1. Brey, B. B. (2008). *The Intel microprocessors* (8th ed). Pearson Education
2. Yu, Y. Y., & Marut, C. (1992). *Assembly language programming and organization of the IBM PC*. McGraw-Hill Education
3. Patterson, D. A., & Hennessy, J. L. (2014). *Computer organization and design: The hardware/software interface* (5th ed.). Morgan Kaufmann
4. Hyde, R. (2010). *The art of assembly language* (2nd ed.). No Starch Press