Lec 5

**Microprocessors**

**2nd stage**

**13-11-2014**

**Architecture**

In addition to their rated Mega Hertz (MHz) speed, and the amount and type of cache, processors can be categorized by their internal structure, or *architectur*e; it basically determines the language software programs must use to work with them. The vast majority of IBM-compatible PCs use what are called x86 processors because they are derived from Intel’s 8086 processor — the same processor found in some of the earliest IBM PCs. The very first IBM PC, however, used Intel’s 8088 processor, a predecessor to the 8086.

The 8086 and 8088 are often confused because the later chip used a lower number as its product name. The reason for this is that the 8088 was an 8-bit processor, while the 8086 is a 16-bit processor and hence the difference in the last digit.

The x86 family of processors share a common set of instructions that software programs use to run on the chip. These instructions are the basic "language" of the processor and determine what types of calculations the processor is capable of doing.

**Faking it with software**

When is an x86-compatible processor not really an x86-compatible processor? When it’s a Motorola processor (or one of several other types) pretending to be an x86-compatible via trickery, commonly called software emulation. Software emulation basically fools an application into thinking it’s running on the type of chip that the application was originally written, even though it really isn’t. The processor does this by translating one set of instructions to another. So, for example, Insignia Solution’s SoftWindows 98 or Connectix Corp.’s Virtual PC, which are programs for the Apple Macintosh, enable Mac users to run Windows programs on the Motorola processor inside the Macintosh, even though those Windows programs were written for Intel processors. Basically, the instructions the Windows program (and even Windows itself) calls for are translated into a form that the Motorola processor understands through the software emulation program. Then the Motorola processor performs the necessary calculations, and finally, the software emulator takes those results and feeds them back to the Windows application. As far as the application is concerned, it’s running under Windows on an Intel processor, as normal. Performing these translations back and forth takes time and processing power, however, and software emulators are much slower than running the same programs on PC hardware.

Other processors use different instructions. This is why programs written for the Macintosh, for example, won’t work on PC-compatible machines; Mac programs use instructions that are specific to the PowerPC family of processors. If you try to run a Mac program on a computer with an x86 chip, the x86 will think you are speaking to it in a foreign language. (Not to confuse matters, but it is possible to run applications written for one type of chip architecture on another via a technology called *software emulatio*n.)

Each generation of x86 chips has added to the original set of core instructions that previous generations could understand, thereby expanding the capabilities of the processor. This explains why some newer applications, written to work with the latest generation of processors, won’t run on older computers, even though they use the same type of chip. In other words, some applications require a Pentium or Pentium-class processor to work and won’t run, for example, on a 486.

**6-Processor Registers**

Processors have a number of registers to hold data, instructions, and state information. Registers are essentially extremely fast memory locations within the CPU that are used to create and store the results of CPU operations and other calculations.

Different computers have different register sets. They differ in the number of registers, register types, and the length of each register. They also differ in the usage of each register. Typically, we can divide the registers into general-purpose or special-purpose registers. **General-purpose registers** can be used for multiple purposes and assigned to a variety of functions by the programmer. **Special-purpose registers** are restricted to only specific functions. Some processors maintain a few special-purpose registers. For example, the Pentium uses a couple of registers to implement the processor stack. Processors also have several registers reserved for the instruction execution unit. Typically, there is an instruction register that holds the current instruction and a program counter that points to the next instruction to be executed.

In some cases, some registers are used only to hold data and cannot be used in the calculations of operand addresses. The length of a data register must be long enough to hold values of most data types. Some machines allow two contiguous registers to hold double-length values. Address registers may be dedicated to a particular addressing mode or may be used as address general purpose. Address registers must be long enough to hold the largest address. The number of registers in a particular architecture affects the instruction set design. A very small number of registers may result in an increase in memory references. Another type of registers is used to hold processor status bits, or flags. These bits are set by the CPU as the result of the execution of an operation. The status bits can be tested at a later time as part of another operation.

**1. Memory Access Registers**

Two registers are essential in memory write and read operations: the memory data register (MDR) and memory address register (MAR). The MDR and MAR are used exclusively by the CPU and are not directly accessible to programmers.

In order to perform a write operation into a specified memory location, the MDR and MAR are used as follows:

1. The word to be stored into the memory location is first loaded by the CPU into MDR.

2. The address of the location into which the word is to be stored is loaded by the CPU into a MAR.

3. A write signal is issued by the CPU.

Similarly, to perform a memory read operation, the MDR and MAR are used as follows:

1. The address of the location from which the word is to be read is loaded into the MAR.

2. A read signal is issued by the CPU.

3. The required word will be loaded by the memory into the MDR ready for use by the CPU.

**2. Instruction Fetching Registers**

Two main registers are involved in fetching an instruction for execution: the program counter (**PC**) and the instruction register (**IR**). The PC is the register that contains the address of the next instruction to be fetched. The fetched instruction is loaded in the IR for execution. After a successful instruction fetch, the PC is updated to point to the next instruction to be executed. In the case of a branch operation, the PC is updated to point to the branch target instruction after the branch is resolved, that is, the target address is known.

**3. Condition Registers**

Condition registers, or flags, are used to maintain status information. Some architectures contain a special program status word (**PSW**) register. The PSW contains bits that are set by the CPU to indicate the current status of an executing program. These indicators are typically for arithmetic operations, interrupts, memory protection information, or processor status.

**4. Special-Purpose Address Registers**

**Index Register**

In index addressing, the address of the operand is obtained by adding a constant to the content of a register, called the index register. The index register holds an address displacement. Index addressing is indicated in the instruction by including the name of the index register in parentheses and using the symbol X to indicate the constant to be added.

**Segment Pointers**

In order to support segmentation, the address issued by the processor should consist of a segment number (base) and a displacement (or an offset) within the segment. A segment register holds the address of the base of the segment.

**Stack Pointer**

A stack is a data organization mechanism in which the last data item stored is the first data item retrieved. Two specific operations can be performed on a stack. These are the Push and the Pop operations. A specific register, called the stack pointer (SP), is used to indicate the stack location that can be addressed. In the stack push operation, the SP value is used to indicate the location (called the top of the stack). After storing (pushing) this value, the SP is incremented.