2012

The British University in Egypt - BUE
Faculty of Informatics and Computer Science

Samir Abou El-Seoud

GLOSSARY
COMPUTER ARCHITICTURE

Definition and description for most technical items used in our course.

- **Address:**
  With computer data storage: An **address** is the location pointing to where data can be accessed. With computer networks: An address refers to a network **IP address** (see definition of IP address below) or other unique network identification. On the Internet: An address is a synonym for a **web address**.
- **Absolute Address:** Alternatively known as a **direct address**, machine address or real address, an absolute address is an exact memory address.
- **Addressing Mode:** An addressing mode specifies how to calculate the effective memory address of an operand by using information held in registers and/or constants contained within a machine instruction. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced. When a microprocessor accesses memory, to either read or write data, it must specify the memory address it needs to access.
- **Arithmetic and Logic Unit (ALU):** A part of a computer that performs arithmetic operations, logic operations, and related operations.
- **Arithmetic Pipeline:** Arithmetic pipelines differ from instruction pipelines in some important ways. They are generally **synchronous.** This means that each stage executes in a fixed number of clock cycles. In a synchronous pipeline, moreover, no buffering between stages is provided. Each stage must be ready to accept the data passed from a previous stage when that data is produced.
- **Assembly Language:** A computer-oriented language whose instructions are usually in one-to-one correspondence with computer instructions and that may provide facilities such as the use of macroinstructions. Synonymous with *computer-dependent language.*
- **Asynchronous Transmission:** Alternatively referred to as a **start/stop transmission**, **asynchronous** is a data transmission where the communication can start and stop at any time. Data sent through an asynchronous transmission contains a start bit and stop bit, helping the receiving end know when it has received all of its data.
- **Asynchronous Transfer Mode (ATM): ATM** when referring to computers is a dedicated-connection switching technology that organizes digital data into 53-byte cell units and transmits them over a physical environment using digital signal technology. ATM is capable of transmitting at speeds of 155 or 622 MBps and faster. Electronic machine that dispenses money from a person's account at banks. **ATM** is commonly used in chat based communications. Below are some examples of how this could be used in chat.
- **Base Address:** When referring to computer memory, the **base address** is a memory starting point that address that references all other memory addresses.
- **Buss:** A shared communication path consisting of one or a collection of lines. In some computer systems, memory, and I/O components are connected by a common bus. Since the lines are shared by all components, only one component at a time can successfully transmit.
- **Address Buss:** That portion of a system bus used for the transfer of an address. Typically, the address identifies a main memory location or an I/O device.
- **Bus Arbitration:** The process of determining which competing bus master will permitted access to the bus. In single bus architecture when more than one device requests the bus, a controller called bus arbiter decides who gets the bus, this is called the **bus arbitration**. Arbitration is mostly done in favor of a master micro processor with the highest priority.
- **CPU:** That portion of a computer that fetches and executes instructions. It consists of ALU, a CU, and registers. Often simply referred to as a *processor*.

- **Computer Instruction Set:** A complete set of the operators of instructions of a computer together with a description of the types of meanings that can be attributed to their operands. Synonymous with *machine instruction set*.
- **Control Bus:** That portion of the system bus used to transfer of control signals.
- **Cycle**--smallest unit of time in a processor.
- **Data Bus:** That portion of a system bus used for the transfer of data or enter data into a storage device in a sequence independent of their relative position, by means of addresses that indicate the physical location of the data.
- **Direct Addressing Mode**: Direct addressing mode is where the address of the operand is contained in the instruction. In other words, the address of the data (operand) is specified within the instruction. The capability to obtain data from a storage device or to enter data into a storage device in a sequence independent of their relative position, by means of addresses that indicate the physical location of the data. Direct addressing mode is where the address of the operand is contained in the instruction. Direct addressing mode means that the value for a given instruction in assembly programming is pointed to by a given value. This means the value is variable, based on what is stored in memory at a given address. For direct addressing mode the address of the item to be accessed is an immediate encoded in the instruction, so the instruction is larger, in some cases much larger so it requires more clock cycles to access, ideally it is in the cache as it is the bytes immediately following the opcode and the fetching of the opcode normally causes at least a cache line behind it to be fetched, with anything but the oldest x86 platforms I don't see how you would get to where you are executing the instruction without the rest of the instruction and the next few/many instructions already fetched and in the pipe. Even old x86 processors had a prefetch queue of some size. In the direct mode, the address field contains the address of the operand. It requires a single memory reference to read the operand from the given location. However, it provides only a limited address space.
- **Direct Memory Address (DMA):** A form of I/O in which a special module, called a DMA module, controls the exchange of data between main memory and an I/O module. The CPU sends a request for the transfer of a block of data to the DMA module and is interrupted only after the entire block has been transferred.
- **Distributed System:** A distributed system is a computer system consisting of a collection of autonomous computers linked by a network and equipped with software that enables the computers to coordinate their activities and to share the resources of system hardware, software, and data, so that users perceive a single, integrated computing facility. In distributed computing, each processor has its own private memory (distributed memory). Information is exchanged by passing messages between the processors.
- **Dynamic RAM:** A RAM whose cells are implemented using capacitors. A dynamic RAM will gradually lose its data unless it is periodically refreshed.
- **Effective Address:** It just means the actual address used for an instruction's operand, or when the address itself is the operand, as in the case of the Load Effective Address (LEA) instruction on Intel CPU's. The effective address is the address generated by the program, after all transformations, such as index registers, offsets, addressing mode, etc. have been made. The physical address is the address generated by the hardware, after performing whatever lookups through the page table, etc. have been made. The effective address, or virtual address, is the concern of the program. The physical address, or real address, is the concern of the operating system.
- **Execute Cycle**: That portion of the instruction cycle during which the CPU performs the operations specified by the instruction cycle.

- **Fetch Cycle:** That portion of the instruction cycle during which the CPU fetches from memory the instruction to be executed.

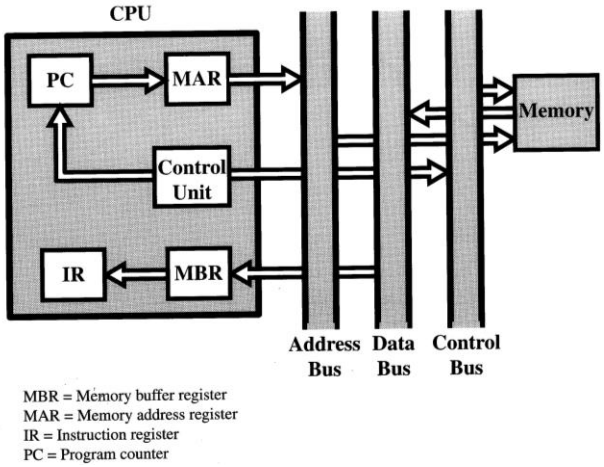
  The fetch cycle occurs at the beginning of each instruction cycle and causes an instruction to be fetched from memory. For purposes of discussion, we assume the organization depicted in the figure below. Four registers are involved:

  Memory address register (MAR): Is connected to the address lines of the system bus. It specifies the address in memory for a read or write operation.

  Memory buffer register (MBR): Is connected to the data lines of the system bus. It contains the value to be stored in memory or the last value read from memory.

  Program counter (PC): Holds the address of the next instruction to be fetched.

  Instruction register (IR): Holds the last instruction fetched.



Data Flow, Fetch Cycle

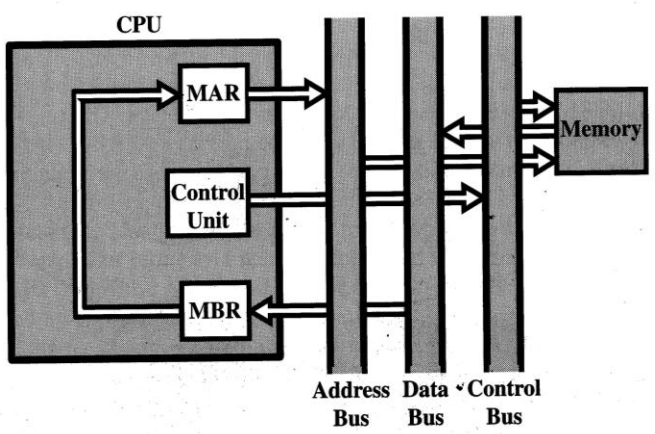
In the fetch cycle, each microoperation involves the movement of data into or out of a register. So long as these movements do not interfere with one another, several of them can take place during one step, saving time. Symbolically, we can write this sequence of events as follows:

t_1: MAR ←[PC]	//Move contents of PC to MAR.
t_2: MBR ←Memory	//Move memory location specified by MAR to MBR.
PC ←[PC]+1	//Increment the PC for the next cycle at the same time
t_3: IR ←[MBR]	//Move contents of memory location specified by MAR to IR.

- **Immediate Addressing Mode:** Immediate addressing mode is where the operand is specified within the instruction itself. Immediate addressing is so-named because the value to be stored in memory immediately follows the operation code in memory. That is to say, the instruction itself dictates what value will be stored in memory. Here the contents of an address part that contains the value of an operand rather than an address. Synonymous with *zero-level address*.
- **Indirect Addressing Mode:** An address of a storage location that contains an address. Indirect addressing mode is where the address of the operand is contained in a register pair, usually HL. This mode is similar to indirect addressing[1)]. The operand is in a memory cell pointed to by contents of a register. The register contains the effective address of the operand. This mode uses one fewer memory access than indirect addressing. This mode has a large address space, but it is limited to the

width of the registers available to store the effective address. In the indirect mode, the memory cell pointed to by the address field contains the address of (pointer) the operand, which in turn contains the full-length address of the operand. This mode has a large address space, unlike direct and immediate addressing, but because multiple memory accesses are required to find the operand it is slower.

- **Indirect Cycle:** That portion of the instruction cycle during which the CPU performs a memory access to convert an indirect address into a direct address.
  Once an instruction is fetched, the next step is to fetch source operands. Continuing our simple example, let us assume a one-address instruction format, with direct and indirect addressing allowed. If the instruction specifies an indirect address, then an indirect cycle must precede the execute cycle.



Data Flow, Indirect Cycle.

The data flow differs somewhat from that indicated in the figure below and includes the following microoperations:

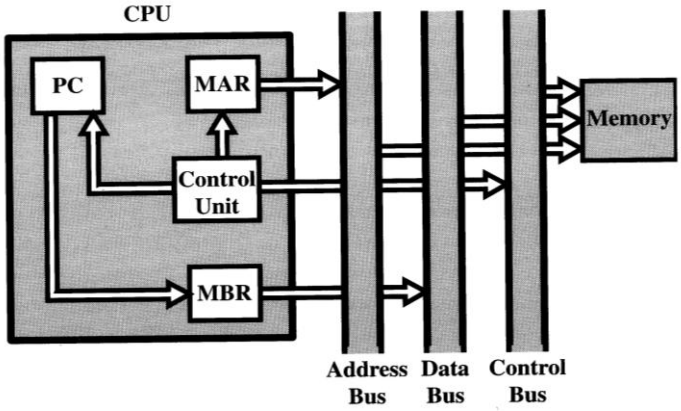
t_1: MAR ←[IR(Address)] //The address field of the instruction is transferred to the MAR.
t_2: MBR ←Memory //Fetch the address of the operand from memory and transters it to MBR.
t_3: IR(Address)←[MBR(Address)] //The address field of the IR is updated from the MBR, so that it now contains a direct rather than an indirect address

The IR is now in the same state as if indirect addressing had not been used, and it is ready for the execute cycle. We skip that cycle for a moment, to consider the interrupt cycle.

- **Instruction Cycle:** The processing performed by a CPU to execute a single instruction.
- **Instruction Format:** The layout of a computer instruction as a sequence of bits. The format divides the instruction into fields, corresponding to the constituent elements of the instruction (e.g. opcode, operands).
- **Instruction Pipeline:** An instruction pipeline is a technique used in the design of computers to increase their instruction throughput (the number of instructions that can be executed in a unit of time). Pipelining does not reduce the time to complete an instruction, but increases the number of instructions that can be processed at once. Each instruction is split into a sequence of dependent steps. The first step is always to fetch the instruction from memory; the final step is usually writing the results of the instruction to processor registers or to memory. Pipelining seeks to let the processor work on as many instructions as there are dependent steps, just as an assembly line builds many vehicles at once, rather than waiting until one vehicle has passed through the line before

admitting the next one. As the goal of the assembly line is to keep each assembler productive at all times, pipelining seeks to use every portion of the processor busy with some instruction. Pipelining lets the computer's cycle time be the time of the slowest step, and ideally lets one instruction complete in every cycle.

- **Interrupt Cycle**
  At the completion of the execute cycle, a test is made to determine whether any enabled interrupts have occurred. The nature of this cycle varies greatly from one machine to another. We present a very simple sequence of events, as illustrated in the figure below.



Data Flow, Interrupt Cycle.

We have:

t_1: MAR ←[PC]
t_2: MAR ←Save_Address
PC ←Routine_Address
t_3: Memory←[MBR]

In the first step, the contents of the PC are transferred to the MBR., so that they can be saved for return from the interrupt. Then the MAR is loaded with the address at which the contents of the PC are to be saved., and the PC is loaded with the address of the start of the interrupt-processing routine. These two actions may each be a single microoperation. However, because most processors provide multiple types and/or level of interupts, it may take one or more additional microoperations to obtain the save_address and the routine_adress before they can be transferred to the MAR and PC, respectively. In any case, once this is done, the final step is to store the MBR, which contains the old value of the PC, into memory. The processor is now ready to begin the next instruction cycle.

- **Pipeline Processor:** A pipeline processor executes instructions in an assembly line manner so multiple tasks can be performed simultaneously. However, at no given time can two of the same tasks be performed, so each task is allotted the same time as the task that takes the longest amount of time. Think of an automated car wash line, where multiple cars are serviced, but only one vehicle at a time can be shampooed, conditioned or dried.
- **Instruction Register (IR):** A register that is used to hold an instruction for interpretation. It is used to store the current instructions which are being executed. (Instruction register: holds the instruction while it is being executed)

- **Instruction Set Architecture (ISA**): ISA is the part of the processor that is visible to the programmer or compiler writer. The ISA serves as the boundary between software and hardware. A microprocessor's ISA includes the information needed to interact with the microprocessor, but not the details of how the microprocessor itself is designed and implemented. It provides the details a programmer would need to know in order to write a program for the microprocessor, or the details a compiler would need in order to compile a program written in C++, or some other high-level programming language, so it could be run by the microprocessor.
- **Integrated Circuit (IC):** A tiny piece of a solid material, such as silicon, upon which is etched or imprinted a collection of electronic components and their interconnection.
- **Interrupt:** A suspension of a process, such as the execution of a computer program, caused by an event external to that process, and performed in such a way that the process can be resumed. An **interrupt** signal in a microprocessor is a request to stop program execution and go do something else, such as service a device request or hardware condition. The interrupt return sequence restores normal program flow to where it was interrupted. An interrupt is a signal informing a program that an event has occurred. When a program receives an interrupt signal, it takes a specified action (which can be to ignore the signal). Interrupt signals can cause a program to suspend itself temporarily to service the interrupt. Interrupt handling is a very important concept. Inside each computer there are several devices like input devices (Keyboards), output devices (Monitor, Printer), and memory (Disk) sends and receives data to Processor and from processor. This transfer is happen with the help of an Interrupt call. If CPU is busy in some processing and an I/O device wants to send some data to CPU for the current program need then that Device issues an interrupt and accordingly an Interrupt subroutine is executed. And after it CPU starts its normal execution. This approach helps a lot so that maximum utilization of central processing unit (Processor) can happen. An interrupt in the 8085 microprocessor is a request to stop program execution and go do something else, such as service a device request or hardware condition. The interrupt return sequence restores normal program flow to where it was interrupted. An interrupt is a signal informing a program that an event has occurred. When a program receives an interrupt signal, it takes a specified action (which can be to ignore the signal). Interrupt signals can cause a program to suspend itself temporarily to service the interrupt. Synonymous with *interruption*.
- **Interrupt Cycle:** That portion of the instruction cycle during which the CPU checks for interrupts. If an enabled interrupt is pending, the CPU saves the current program state and resumes processing at an interrupt-handler routine. An interrupt cycle is a memory fetch sequence generated in response to the interrupt request. It can be identified in hardware by the status lines, and the expected response is an op-code, optionally followed by immediate bytes, such as the address of a CALL instruction. Indirect addressing mode is where the address of the operand is contained in a register pair, usually HL.
- **I/O Module:** One of the major component types of a computer. It is responsible for the control of one or more external devices (peripherals) and for the exchange of data between those devices and main memory and/or CPU registers.
- **Immediate Addressing Mode:** Immediate addressing mode is where the operand is specified within the instruction itself. Immediate addressing is so-named because the value to be stored in memory immediately follows the operation code in memory. That is to say, the instruction itself dictates what value will be stored in memory.

- **Instruction Cycle:** Instruction cycle (also called **fetch-and-execute cycle**, **fetch-decode-execute cycle-FDX**) is the time period during which a computer reads and processes a machine language instruction from its memory or the sequence of actions that the central processing unit (CPU) performs to execute each machine code instruction in a program. In other words, an instruction cycle is the fundamental sequence of steps that a CPU performs. Also known as the "fetch-execute cycle," it is the time in which a single instruction is fetched from memory, decoded and executed. The first half of the cycle transfers the instruction from memory to the instruction register and decodes it. The second half executes the instruction. An **instruction cycle** is the basic operation cycle of a computer. It is the process by which a computer retrieves a program instruction from its memory, determines what actions the instruction requires, and carries out those actions. This cycle is repeated continuously by the CPU, from bootup to when the computer is shut down. The name **fetch-and-execute cycle** is commonly used. The instruction must be **fetched** from main memory, and then **executed** by the CPU. This is fundamentally how a computer operates, with its CPU reading and executing a series of instructions written in its machine language. From this arise all functions of a computer familiar from the user's end.
- **IP Address:** Short for **Internet Protocol. IP** is an address of a computer or other network device on a network using IP or TCP/IP. For example, the number "166.70.10.23" is an example of such an address. These addresses are similar to an addresses used on a house and is what allows data to reach the appropriate destination on a network and the Internet. There are five classes of available IP ranges: Class A, Class B, Class C, Class D and Class E, while only A, B, and C are commonly used. Each class allows for a range of valid IP addresses. Below is a listing of these addresses.
- **Main Memory:** Program-addressable storage from which instructions and other data can be loaded directly into registers for subsequent execution or processing.

- **Memory Address:** A memory address is an exact assigned location in RAM used to track where information is stored. On a single computer memory IC, there can be 1 million, 2 million, or more memory addresses that can be accessed at randomly, which is why memory is called RAM (random access memory.)
- **Memory Address Register (MAR):** A register that contains data read from memory or data to be written to memory. The address on the memory location from which data has to be read is stored here.
- **Memory Buffer Register (MBR):** A register that contains data read from memory or data to be written to memory. The data read from the memory location is stored in this registers.
- **Memory Capacity:** The memory capacity is the maximum or minimum amount of memory a computer or hardware device is capable of having or the required amount of memory required for a program to run. For example, a computer software program may list memory requirements similar to those shown below. Recommend 32MB of memory (minimum 16MB of memory). In the above recommendations, the developer of the program recommends, for optimal performance, that the computer has 32MB of memory. However, it is capable of running with 16MB of memory, but may lack in performance.
- **Memory Cycle Time:** The inverse of the rate at which memory can be accessed. It is the minimum time between the response to one access request (read or write) and the response to the next access request.

- **Microinstruction:** Microinstruction is an instruction that controls data flow and sequencing in a processor at a more fundamental level than machine instructions. Individual machine instructions and perhaps other functions may be implemented by microprograms. **Microinstruction** is an instruction that controls data flow and sequencing in a processor at more fundamental level than machine instructions. Individual machine instructions and perhaps other functions may be implemented by microprograms. A **microinstruction** is a simple command that makes the hardware operates properly. The format is unique to each computer. Microinstruction is an instruction stored in control memory. The format is unique to each computer. Microinstruction is an instruction stored in control memory.
- **Microoperation:** An elementary CPU operation, performed during one clock pulse.
- **Microprogram:** A sequence of microinstructions that are in special storage where they can be dynamically accessed to perform various functions.
- **Non-Pipelined Processor**: In a non-pipelined processor each instruction is executed completely before execution of the next instruction begins. In a **non-pipelined** CPU, the scheduler merely chooses from the pool of waiting work each time an execution unit signals it is free.
- **Opcode:** Abbreviated form for *operation code*.
- **Operand:** An entity on which an operation is performed.
- **Operating System:** Software that controls the execution of programs and that provides services such as recourse allocation, scheduling, input/output control, and data management.
- **Operation Code:** A code used to represent the operations of a computer. Usually, abbreviated to opcode.
- **Parallel Processing:** Parallel processing involves a technique by which complex data sets are broken into individual threads and processed simultaneously across one or more cores. Both AMD and Intel processors have incorporated this technique (known as HTT) to greatly increase the speed at which they operate. Until recently, this did not always provide a significant increase in speed because the technology to properly split up data sets and then bring them back together was in its infancy.

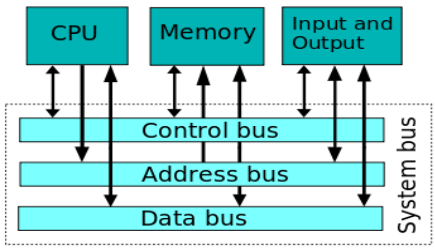
- **Parallel Computing:** Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently ("in parallel"). Parallel computers can be roughly classified according to the level at which the hardware supports parallelism, with multi-core and multi-processor computers having multiple processing elements within a single machine. In parallel computing, all processors may have access to a shared memory to exchange information between processors.
- **Parallelism**--the ability to do more than one thing at once.
- **Pipeline:** A processor organization in which the processor consists of a number of stages, allowing multiple instructions to be executed concurrently. In a **pipelined** processor, instruction execution is divided into stages and execution of the next instruction starts as soon as the current instruction has completed the first stage. This increases the rate at which instructions can be executed, improving performance. **Pipeline** in this context refers to a processor organization in which the processor consists of a number of stages, allowing multiple instructions to be executed concurrently. Thus, the efficiency of a pipelined CPU is entirely dependent on the CPU scheduler's effectiveness at predicting the outcome of each instruction (action). If the workload is such that the predictive scheduler can't do a good job, and frequent pipeline stalls occur, then it will often be the case that a non-pipelined design will perform better on that workload. The trade-off is thus: the longer the instruction pipeline for an execution unit, the better performance that unit can have, but the harder

(and more complex) the work is for the predictive scheduler, and the greater the cost (in terms of performance hit) that a pipeline stall is.

- **Pipelining:** pipelining is a technique for overlapping the execution of several instructions to reduce the overall execution time of a set of instructions. **Pipelining** refers to overlapping operations by moving data or instructions into a conceptual pipe with all stages of the pipe processing simultaneously. **Pipelining:** Overlapping parts of a large task to increase **throughput** without decreasing latency. A pipeline does not speed up an individual computation. A pipeline processor executes instructions in an assembly line manner so multiple tasks can be performed simultaneously. However, at no given time can two of the same tasks be performed, so each task is allotted the same time as the task that takes the longest amount of time. The net effect is that results are output more quickly than in a non-pipelined unit. This increases the **throughput**, the number of results generated per time unit. **Throughput** is also defined to be the rate at which operations get executed (generally expressed as operations/second or operations/cycle). Think of an automated car wash line, where multiple cars are serviced, but only one vehicle at a time can be shampooed, conditioned or dried. "**Pipelining**", in the context of a processor, means that the CPU scheduler creates a specific list of **linked** instructions (actions) to be fed to the computation units to work. Generally speaking, this list is a series of actions which require the successful completion of the prior one - so, action A completes, then action B takes the output from A and does something, while action C then does something with the output of B, etc. **Pipelining** can bring significant performance benefits, as each successive action finds all its prerequisites already satisfied, so the action is ready to go immediately. **Pipelining** does not reduce the time to complete an instruction, but increases the number of instructions that can be processed at once. **Pipelining:** Overlapping parts of a large task to increase **throughput** without decreasing latency.
- **Processor:** In a computer, a functional unit that interprets and executes instructions. A processor consists of at least an instruction control unit and an arithmetic unit.
- **Processor Cycle Time:** The time required for the shortest well-defined CPU microoperation. It is the basic unit of time for measuring all CPU actions. Synonymous with *machine cycle time*.
- **Program Counter (PC):** Its only function is to hold the memory address of the next instruction to be fetched, after executing the current instruction it is also called instructions address register, control register or sequence control register. (Program counter: holds the address of the next instruction to be executed).
- **Parallel Processing:** Parallel processing involves a technique by which complex data sets are broken into individual threads and processed simultaneously across one or more cores. Both AMD and Intel processors have incorporated this technique (known as HTT) to greatly increase the speed at which they operate. Until recently, this did not always provide a significant increase in speed because the technology to properly split up data sets and then bring them back together was in its infancy.
- **Parallel Computing:** Parallel computing is the processing of data many bits at a time as opposed to serial computing which is the processing of data one bit at a time. In other words, parallel computer runs more tasks in **parallel**, over more CPUs, for faster execution.
- **Registers:** High-speed memory internal to the CPU. Some registers are user visible; that is, available to the programmer via the machine instruction set. Other registers are used only by the CPU, for control purposes. A *register* is a very small amount of very fast *memory* that is built into the CPU (central processing unit) in order to speed up its operations by providing quick access to commonly used values. Memory refers to semiconductor devices whose contents can be accessed (i.e., read and written to) at extremely high speeds but which are held there only temporarily (i.e., while in use or only as long as the power supply remains on). Most memory consists of *main memory*, which is comprised of RAM (random access memory) chips that are connected to the CPU by a *bus* (i.e., a set of dedicated wires). Registers are the top of the *memory*

hierarchy and are the fastest way for the system to manipulate data. Below them are several levels of *cache memory*, at least some of which is also built into the CPU and some of which might be on other, dedicated chips. Cache memory is slower than registers but much more abundant. Below the various levels of cache is the main memory, which is even slower but vastly more abundant (e.g., hundreds of megabytes as compared with only 32 registers). But it, in turn, is still far faster and much less capacious than *storage* devices and media (e.g., hard disk drives and CDROMs). Registers can also be classified into general purpose and special purpose types. The former serve as temporary holding places for data that is being manipulated by the CPU. That is, they hold the inputs to the arithmetic/logic circuitry and store the results produced by that circuitry.

- **Serial Processing:** Serial processing involves a technique where data is ordered sequentially and then calculated by an individual processor. This technique works very well with ordered lists of data that use similar contractions. The biggest problem with this is that only one bit of data can be computed at a time, and complex programs cannot be split up into smaller segments.
- **Superscalar Processor:** Super processor can execute more than one instruction per cycle. **:** Super processor is a processor design that includes multiple-instruction pipelines, so that more than one instruction can be executing in the same pipeline stage simultaneously.
- **System Bus:** A system bus is a single computer bus that connects the major components of a computer system (processor, memory, I/O). It is a collection of wires through which data is transmitted from one part of a computer to another. It is a communication pathway connecting all the internal computer components to the CPU and main memory. A key characteristic of a system bus is that it is a shared transmission medium. A system bus consists, typically, of from 50 to 100 separate lines. Each line is assigned a particular meaning or function. All system buses consist of three parts – a data bus, an address bus, and a control bus. The data bus (data lines) transfers actual data whereas the address bus (address lines) transfers information about where the data should go. The control bus (control lines) is used to control the access to and the use of the data and address lines.



- **RISC (R**educed **I**nstruction **S**et **C**omputers) versus **CISC (C**omplex **I**nstruction **S**et **C**omputers)

a. The CISC Approach

The primary goal of CISC architecture is to complete a task in as few lines of assembly as possible. This is achieved by building processor hardware that is capable of understanding and executing a series of operations. For this particular task, a CISC processor would come prepared with a specific instruction (we'll call it "MUL").

When executed, this instruction loads the two values into separate registers, multiplies the operands in the execution unit, and then stores the product in the appropriate register. Thus, the entire task of multiplying two numbers can be completed with one instruction:

MUL R1, [R2], [R3]

MUL is what is known as a "complex instruction." It operates directly on the computer's memory banks and does not require the programmer to explicitly call any loading or storing functions. It closely resembles a command in a higher level language

One of the primary advantages of this system is that the compiler has to do very little work to translate a high-level language statement into assembly. Because the length of the code is relatively short, very little RAM is required to store instructions. The emphasis is put on building complex instructions directly into the hardware.

b. The RISC Approach

RISC processors only use simple instructions that can be executed within one clock cycle. Because RISC architectures are implemented using the load-store mode, a RISC processor would require several instructions to implement the single CISC MUL operation described above in the CISC approach. Thus, the "MUL" command described above could be divided into four separate commands: "LDR" which moves data from the memory bank to a register, "MUL" which finds the product of two operands located within the registers, and "STR," which moves data from a register to the memory banks. In order to perform the exact series of steps described in the CISC approach, a programmer would need to code four lines of assembly:

```
LDR R4, [R2]
LDR R5, [R3]
MUL R1, R4, R5
STR  R1
```

At first, this may seem like a much less efficient way of completing the operation. Because there are more lines of code, more RAM is needed to store the assembly level instructions. The compiler must also perform more work to convert a high-level language statement into code of this form.

CISC	**RISC**
Emphasis on hardware	Emphasis on software
Expensive processors than RISC types	Produces faster and cheaper processor
Includes multi-clock	Single-clock
Instruction set & chip hardware become more complex with each generation of computers	Simpler hardware
Complex instructions	Reduced instruction only
Memory-to-memory	Register to register
"LOAD" and "STORE" incorporated in instructions	"LOAD" and "STORE" are independent instructions
Small code sizes	Large code sizes
High cycles per second	Low cycles per second
Transistors used for storing complex instructions	Spends more transistors on memory registers

However, the RISC strategy also brings some very important advantages. Because each instruction requires only one clock cycle to execute, the entire program will execute in approximately the same amount of time as the multi-cycle "MUL command. These RISC "reduced instructions" require less transistors of hardware space than the complex instructions, leaving more room for general purpose registers. Because all of the instructions execute in a uniform amount of time (i.e. one clock), pipelining is possible.

Separating the "LDR" and "STR" instructions actually reduces the amount of work that the computer must perform. After a CISC-style "MUL" command is executed, the processor automatically erases the registers. If one of the operands needs to be used for another computation, the processor must re-load the data from the memory bank into a register. In RISC, the operand will remain in the register until another value is loaded in its place.

The Performance Equation

The following equation is commonly used for expressing a computer's performance ability:

Time / program = (time/cycle) *x* (cycles/instruction) *x* (instructions/program)

The CISC approach attempts to minimize the number of instructions per program, sacrificing the number of cycles per instruction. RISC does the opposite, reducing the cycles per instruction at the cost of the number of instructions per program.

Solved Example:
Rewrite the following CISC-style program fragment so that it executes correctly on a RISC (load-store) processor that executes the GPR ISA. Assume that the GPR ISA provides only the register addressing mode, and that there are enough registers in the processor to hold any temporary values that you need to generate.

```
DIV R1, [R2], [R3]
SUB R0, R1, [R5]
ADD R3, R0, R1
MUL [R1], R0, R3
```

Solution:

This program fragment makes four memory references as part of arithmetic operations. To execute the fragment, we need to replace each of those with an explicit **load** or **store**. Here is a code fragment that does that:

```
LDR R4, [R2]
LDR R6, [R3]
DIV R1, R4, R5
LDR R7, [R5]
```

```
SUB R0, R1, R7
ADD R3, R0, R1
LDR R8, [R1]
MUL R8, R0, R3
STR [R1], R8
```

- **Vector processing:** A vector processing is a procedure for speeding the processing of information by a computer, in which pipelined units perform arithmetic operations on uniform, linear arrays of data values, and a single instruction involves the execution of the same operation on every element of the array.
- **Vector Processor:** A vector processor, or array processor, is a central processing unit (CPU) that implements an instruction set containing instructions that operate on one-dimensional arrays of data called vectors. This is in contrast to a scalar processor, whose instructions operate on single data items.