The CPU and the fetch-execute cycle

The fetch-execute cycle

The basic operation of a computer is called the 'fetch-execute' cycle. The computer fetches the instruction from its memory and then executes it. This is done repeatedly from when the computer is booted up to when it is shut down.

Fetching the instruction

The first step the fetch-execute cycle carries out is fetching the instruction, which is either a program or data. The CPU fetches this from the main memory (the hard drive) and stores it in the CPU temporary memory, the immediate access store (the registers).

Once the instruction has been fetched, the CPU will need to understand the instruction to action it. This is called **decoding**.

Executing the instruction

When the instruction has been decoded, the CPU can carry out the action that is needed. This is called **executing** the instruction. The CPU is designed to understand a set of instructions - the **instruction set**.

A single piece of program code might require several instructions. Look at this Python (3.x) code:

```
area = length * width
```

First, the computer needs to load in the value of the variable **length** into the immediate access store (registers). Next it needs to load in the value of the variable **width**. Then it needs to multiply the two numbers together, and finally it needs to store the result in the variable **area**.

Instruction cycle

An **instruction cycle** (sometimes called **fetch-decode-execute 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 central processing unit (CPU), from bootup to when the computer is shut down.

In simpler CPUs, the instruction cycle is executed sequentially: each instruction is completely processed before the next one is started. In most modern CPUs, the instruction cycle is instead executed concurrently in parallel, as an instruction pipeline: the next instruction starts being processed before the previous instruction is finished, which is possible because the cycle is broken up into separate steps.

Circuits Used

The circuits used in the CPU during the cycle are:

- **Program counter (PC)** - an incrementing counter that keeps track of the memory address of the instruction that is to be executed next
- **Memory address register (MAR)** - holds the address of a memory block to be read from or written to
- **Memory data register (MDR)** - a two-way register that holds data fetched from memory (and ready for the CPU to process) or data waiting to be stored in memory
- **Instruction register (IR)** - a temporary holding ground for the instruction that has just been fetched from memory
- **Control unit (CU)** - decodes the program instruction in the IR, selecting machine resources such as a data source register and a particular arithmetic operation, and coordinates activation of those resources
- **Arithmetic logic unit (ALU)** - performs mathematical and logical operations

Instruction cycle is the time period during which one instruction is fetched from memory and executed when a computer is given an instruction in machine language. There are typically four stages of an instruction cycle that the CPU carries out:

1. Fetch the instruction from memory.
2. "Decode" the instruction.
3. "Read the effective address" from memory if the instruction has an indirect address.
4. "Execute" the instruction.

Initiating the cycle

The cycle starts immediately when power is applied to the system using an initial PC value that is predefined for the system architecture (in Intel IA-32 CPUs, for instance, the predefined PC value is 0xfffffff0). Typically this address points to instructions in a read-only memory (ROM) (not the random access memory or RAM) which begins the process of loading the operating system. (That loading process is called *booting*.)

Fetch the Instruction

Step 1 of the Instruction Cycle is called the Fetch Cycle. This step is the same for each instruction.

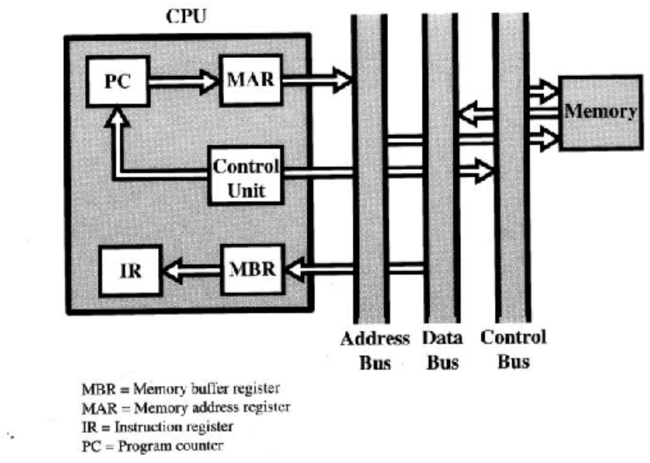
1. The CPU sends the memory address that is stored in the **PC** to the **MAR** and sends a READ command on the control bus
2. In response to the read command (with address stored in **PC**), the memory returns the data stored at the memory location indicated by PC on the databus.
3. The CPU copies the data from the databus into its **MBR.**
4. A fraction of a second later, the CPU copies the data from the **MDR** to the Instruction Register (**IR**)
5. The **PC** is incremented so that it points to the following instruction in memory. This step prepares the CPU for the next cycle.

The Control Unit fetches the instruction's address from the Memory Unit

The fetch cycle could be summarized as follows:

Fetch Cycle is 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.


CPU
PC
MAR
Control Unit
IR
MBR
Memory
Address Bus
Data Bus
Control Bus
MBR = Memory buffer register
MAR = Memory address register
IR = Instruction register
PC = Program counter

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:

```
t1: MAR ←[PC]            //Move contents of PC to MAR.
t2: MBR ←Memory          //Move memory location specified by MAR to MBR.
    PC ←[PC]+1           //Increment the PC for the next cycle at the same time
t3:  IR ←[MBR]           //Move contents of memory location specified by MAR  to IR.
```

Decode the Instruction

Step 2 of the instruction Cycle is called the Decode Cycle. The decoding process allows the CPU to determine what instruction is to be performed, so that the CPU can tell how many operands it needs to fetch in order to perform the instruction. The opcode fetched from the memory is decoded for the next steps and moved to the appropriate registers. The decoding is done by the CPU's Control Unit.

The control unit of CPU passes the decoded information as a sequence of control signals to the relevant function units of the CPU to perform the actions required by the instruction such as reading values from registers, passing them to the ALU to perform mathematical or logic functions on them, and writing the result back to a register. If the ALU is involved, it sends a condition signal back to the CU.

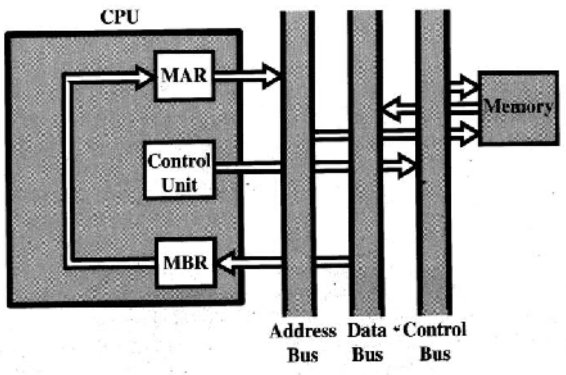
Read the effective address

Step 3 is deciding which operation it is. If this is a Memory operation - in this step the computer checks if it's a direct or indirect memory operation:

- **Direct memory instruction** - Nothing is being done.
- **Indirect memory instruction** - The effective address is being read from the memory.

If this is a I/O or Register instruction - the computer checks its kind and executes the instruction.

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:

t1: MAR ← [IR(Address)] //The address field of the instruction is transferred to the MAR.
t2: MBR ← Memory //Fetch the address of the operand from memory and transfers
// it to MBR.
t3: 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

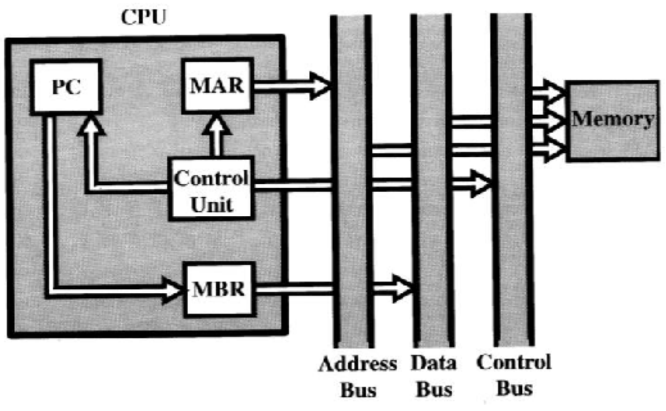
The IR is now in the same state as if indirect addressing had not been used, and it is ready for the execute cycle. We skip that cycle for a moment, to consider the interrupt cycle.

Execute the Instruction

Step 4 of the Instruction Cycle is the Execute Cycle. Here, the function of the instruction is performed. If the instruction involves arithmetic or logic, the Arithmetic Logic Unit is utilized. This is the only stage of the instruction cycle that is useful from the perspective of the end user. Everything else is overhead required to make the execute stage happen.

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.

Symbolically, we can write this sequence of events as follows

t1: MAR ← [PC]
t2: MAR ← Save_Address
PC ← Routine_Address
t3: 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 interrupts, 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.

Important remark

Once a program is in memory it has to be executed. To do this, each instruction must be looked at, decoded and acted upon in turn until the program is completed. This is achieved by the use of what is termed the 'instruction execution cycle', which is the cycle by which each instruction in turn is processed. However, to ensure that the execution proceeds smoothly, it is is also necessary to synchronize the activates of the processor.

To keep the events synchronized, the clock located within the CPU control unit is used. This produces regular pulses on the system bus at a specific frequency, so that each pulse is an equal time following the last. This clock pulse frequency is linked to the clock speed of the processor - the higher the clock speed, the shorter the time between pulses. Actions only occur when a pulse is detected, so that commands can be kept in time with each other across the whole computer unit.

The instruction execution cycle can be clearly divided into three different parts, which will now be looked at in more detail. For more on each part of the cycle click the relevant heading, or use the next arrow as before to proceed though each stage in order.

Let's now summarize the **Fetch-decode-execute cycle steps**

Steps

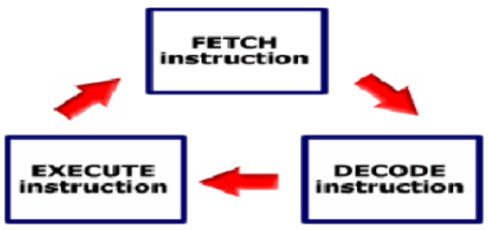
Each computer's CPU can have different cycles based on different instruction sets, but will be similar to the following cycle:

1. **Fetching the instruction**: The next instruction is fetched from the memory address that is currently stored in the program counter (PC), and stored in the instruction register (IR). At the end of the fetch operation, the PC will be incremented by 1 and points to the next instruction that will be read at the next cycle.
2. **Decode the instruction**: Here, the control unit checks the instruction that is now stored within the instruction register (IR). It determines by the decoder which opcode and addressing mode have been used, and as such what actions need to be carried out in order to execute the instruction in question. The decode cycle is used for interpreting the instruction that was fetched in the Fetch Cycle. The operands are retrieved from the addresses if the need be.
3. **Read the effective address**: In this step the computer checks if it's a direct or indirect memory operation:
   - Direct memory instruction - Nothing is being done.
   - Indirect memory instruction - The effective address is being

   If the instruction has an indirect address, the effective address is read from main memory, and any required data is fetched from main memory to be processed and then placed into data registers (Clock Pulse: T_3).
   If the instruction is direct, nothing is done at this clock pulse. If this is an I/O instruction or a Register instruction, the operation is performed (executed) at clock Pulse.

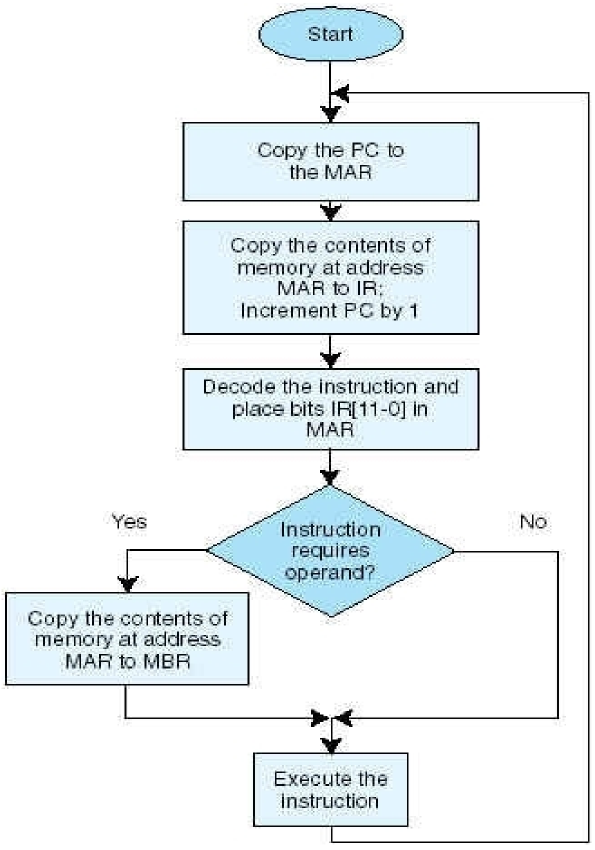
4. **Execute the instruction**: The control unit of the CPU passes the decoded information as a sequence of control signals to the relevant function units of the CPU to perform the actions required by the instruction such as reading values from registers, passing them to the ALU to perform mathematical or logic functions on them, and writing the result back to a register. If the ALU is involved, it sends a condition signal back to the CU. The result generated by the operation is stored in the main memory, or sent to an output device. Based on the condition of any feedback from the ALU, Program Counter may be updated to a different address from which the next instruction will be fetched.

The cycle is then repeated.



The above diagram shows the basics of the instruction execution cycle. Each instruction is fetched from memory, decoded, and then executed.

A diagram of the fetch execution cycle


Start
Copy the PC to the MAR
Copy the contents of memory at address MAR to IR; Increment PC by 1
Decode the instruction and place bits IR[11-0] in MAR
Instruction requires operand?
Yes
No
Copy the contents of memory at address MAR to MBR
Execute the instruction

The Fetch-Execute cycle in Transfer Notation (Expressed in register transfer notation):

t_1: MAR ← [PCT]

t_2: ***MDR ← [Memory]$_{MAR\ address}$,***

PC ← [PC] + 1 *increment the PC for the next instruction at the same time.*

t_3: MAR ←[MDR]

The registers used above, besides the ones described earlier, are the Memory Address Register (**MAR**) and the Memory Data Register (**MDR**), which are used (at least conceptually) in the accessing of memory. Often, the MDR is expressed as the MBR (Memory Buffer Register).

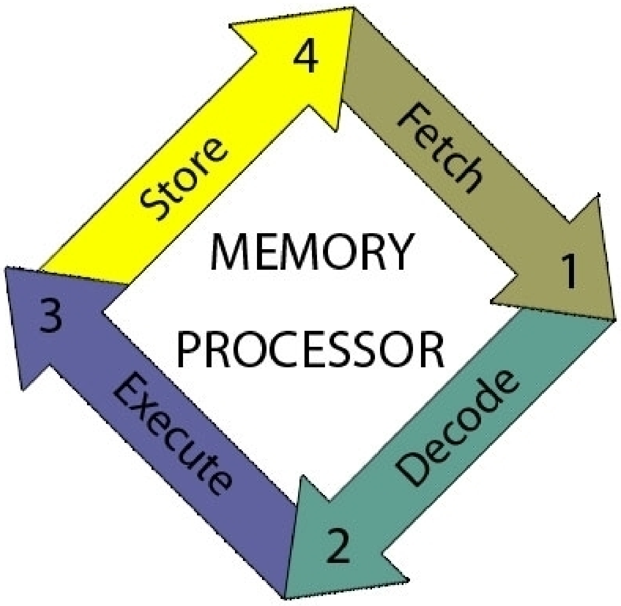
What is the fetch execute cycle on a computer? (Alternative question: Describe the fetch execute cycle?)

Brief Answer:

The fetch execute cycle is the time period of which the computer reads and processes the instructions from the memory, and executes them. This process is a continuous cycle which is used until the computer is turned off or there are no more instructions to process.

The program counter (PC) in the processor holds the address of the next instruction needed from main memory. The program counter copies its contents into the memory address register (MAR). The memory address register then sends the address along the address bus to main memory and the contents of the memory location specified by the address are sent along the data bus to the memory buffer register (MBR). The contents of the memory buffer register are then copied to the current instruction register (IR) where they are decoded and executed.

In other words, the fetch cycle begins with retrieving the address stored in the Program Counter (PC). The address stored in the PC is some valid address in the memory holding the instruction to be executed. (In case this address does not exist we would end up causing an interrupt or exception). The Central Processing Unit completes this step by fetching the instruction stored at this address from the memory and transferring this instruction to a special register - Instruction Register (IR) to hold the instruction to be executed. The program counter is incremented to point to the next address from which the new instruction is to be fetched.



What is interrupt and interrupt cycle?

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 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. *Except, see interrupt types below.*

The interrupt service routine is expected to save and restore the machine state, so that the interrupted program is not disturbed. *Except for the delay in processing.*