Micro-Operation

The execution of a program consists of the sequential execution of instructions. Each instruction is executed during an instruction cycle made up of shorter sub-cycles (e.g., fetch, indirect, execute, interrupt). The performance of each sub-cycle involves one or more shorter operations, that is, micro-operations.

Micro-operations are the functional, or atomic, operations of a processor. In this section, we will examine micro-operations to gain an understanding of how the events of any instruction cycle can be described as a sequence of such micro-operations (Figure 6.1).

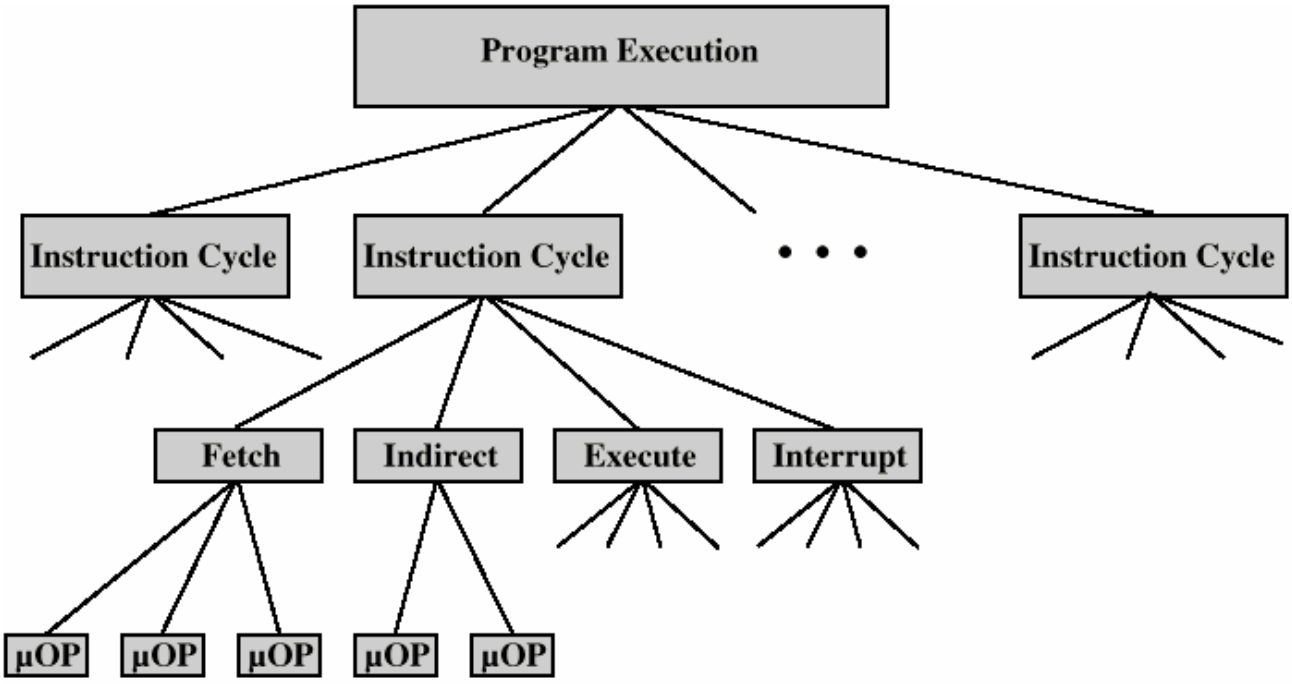

Figure 6.1 Constituent Elements of Program Execution

1.1 The Fetch Cycle

We begin by looking at the fetch cycle, which occurs at the beginning of each instruction cycle and causes an instruction to be fetched from memory. 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.

Let us look at the sequence of events for the fetch cycle from the point of view of its effect on the processor registers. An example appears in Figure 6.2.

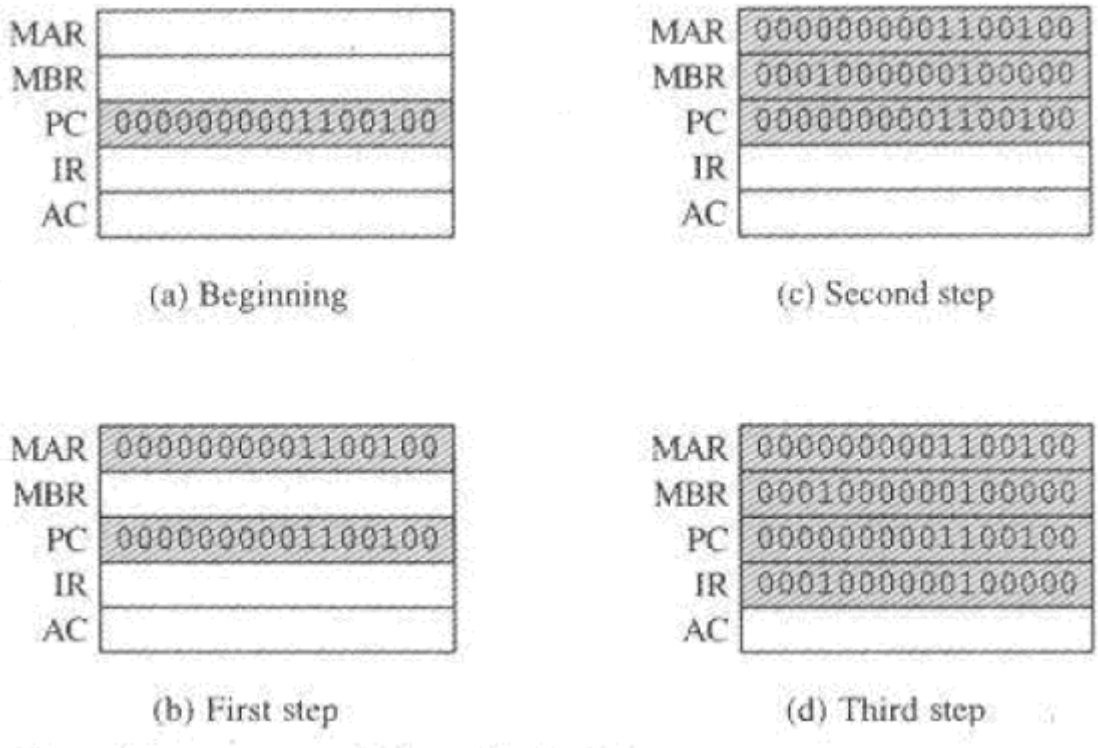

Figure 6.2 Sequence of Events, Fetch Cycle

- At the beginning of the fetch cycle, the address of the next instruction to be executed is in the program counter (PC); in this case, the address is 1100100.
- The first step is to move that address to the memory address register (MAR) because this is the only register connected lo the address lines of the system bus.
- The second step is to bring in the instruction. The desired address (in the MAR) is placed on the address bus, the control unit issues a READ command on the control bus, and the result appears on the data bus and is copied into the memory buffer register (MBR). We also need to increment the PC by 1 to get ready for the next instruction. Because these two actions (read word from memory, add 1 to PC) do not interfere with each other, we can do them simultaneously to save time.
- The third step is to move the contents of the MBR to the instruction register (IR). This frees up the MBR for use during a possible indirect cycle.

Thus, the simple fetch cycle actually consists of three steps and four micro-operations. Each micro-operation 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 lime. Symbolically, we can write this sequence of events as follows:

t1: MAR <= (PC)
t2: MBR <= Memory
PC <= (PC) + l
t3: IR <= (MBR)

where l is the instruction length. We need to make several comments about this sequence. We assume that a clock is available for timing purposes and that it emits regularly spaced clock pulses. Each clock pulse defines a time unit. Thus, all time units are of equal duration. Each micro-operation can be performed within the time of a single time unit. The notation (t1, t2, t3) represents successive time units. In words, we have

- First time unit: Move contents of PC to MAR.
- Second time unit:
    - Move contents of memory location specified by MAR to MBR.
    - Increment by l the contents of the PC.
- Third time unit: Move contents of MBR to IR.

Note that the second and third micro-operations both take place during the second time unit. The third micro-operation could have been grouped with the fourth without affecting the fetch operation:

t1: MAR <= (PC)
t2: MBR <= Memory
t3: PC <= (PC) + l
IR <= (MBR)

The groupings of micro-operations must follow two simple rules:

1. The proper sequence of events must be followed. Thus (MAR <= (PC)) must precede (MBR <= Memory) because the memory read operation makes use of the address in the MAR.
2. Conflicts must be avoided. One should not attempt to read to and write from the same register in one time unit, because the results would be unpredictable. For example, the micro-operations (MBR <= Memory) and (IR <= MBR) should not occur during the same time unit.

A final point worth noting is that one of the micro-operations involves an addition. To avoid duplication of circuitry, this addition could be performed by the ALU. The use of the ALU may involve additional micro-operations, depending on the functionality of the ALU and the organization of the processor.

1.2 The Indirect Cycle

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. The data flow includes the following micro-operations:

t1: MAR <= (IR (Address))
t2: MBR <= Memory
t3: IR(Address) <= (MBR(Address))

The address field of the instruction is transferred to the MAR. This is then used to fetch the address of the operand. Finally, 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.

1.3 The Interrupt Cycle

At the completion of the execute cycle, a test is made to determine whether any enabled interrupts have occurred. If so, the interrupt cycle occurs. The nature of this cycle varies greatly from one machine to another. We present a very simple sequence of events, we have

t1 : MBR <= (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 micro-operation. However, because most processors provide multiple types and/or levels of interrupts, it may lake one or more additional micro-operations to obtain the save_address and the routine_address 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.

1.4 The Execute Cycle

The fetch, indirect, and interrupt cycles are simple and predictable. Each involves a small, fixed sequence of micro-operations and, in each ease, the same micro-operations are repealed

each time around. This is not true of the execute cycle. For a machine with N different opcodes, there are N different sequences of micro-operations that can occur. Let us consider several hypothetical examples.

First, consider an add instruction:

ADD R1, X

which adds the contents of the location X to register Rl. The following sequence of micro-operations might occur:

t1: MAR <= (IR(address))

t2: MBR <= Memory

t3: Rl <= (Rl) + (MBR)

We begin with the IR containing the ADD instruction. In the first step, the address portion of the IR is loaded into the MAR. Then the referenced memory location is read. Finally, the contents of R1 and MBR are added by the ALU. Again, this is a simplified example. Additional micro-operations may be required to extract the register reference from the IR and perhaps to stage the ALU inputs or outputs in some intermediate registers.