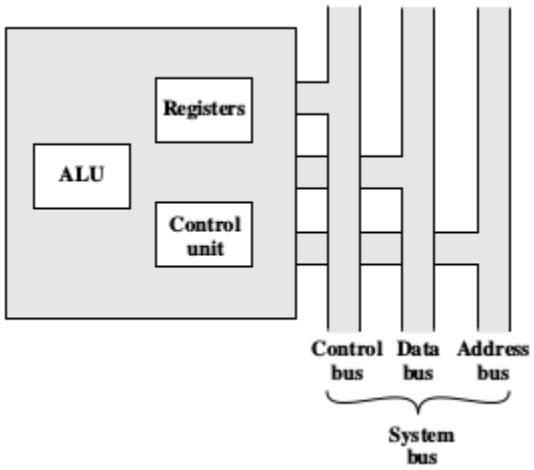
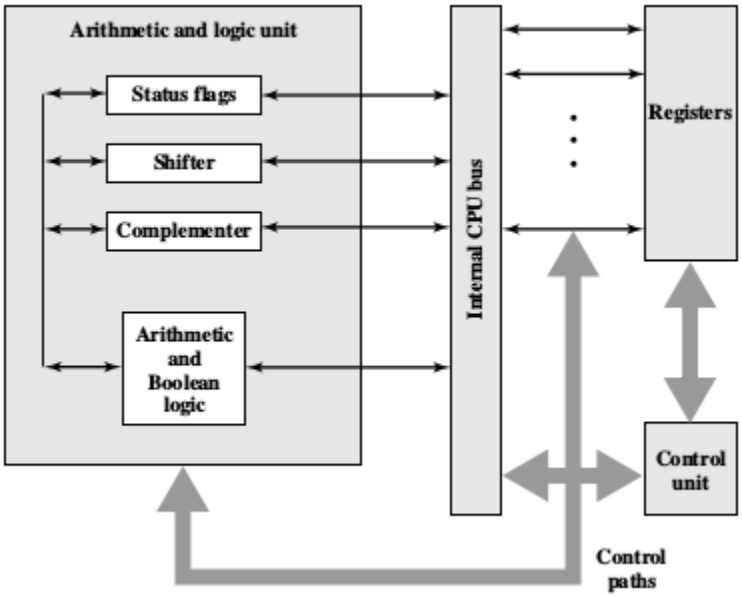
PROCESSOR ORGANIZATION

Figure below is a simplified view of a processor, indicating its connection to the rest of the system via the system bus.



The ALU does the actual computation or processing of data. The control unit controls the movement of data and instructions into and out of the processor and controls the operation of the ALU. In addition, the figure shows a minimal internal memory, consisting of a set of storage locations, called registers.

Figure below depicts is a slightly more detailed view of the processor.



The data transfer and logic control paths are indicated, including an element labeled internal processor bus. This element is needed to transfer data between the various registers and the ALU because the ALU in fact operates only on data in the internal processor memory. The figure also shows typical basic elements of the ALU. Note the similarity between the internal structure of the computer as a whole and the internal structure of the processor. In both cases, there is a small collection of major elements (computer: processor, I/O, memory; processor: control unit,ALU, registers) connected by data paths

REGISTER ORGANIZATION

At higher levels of the hierarchy, memory is faster, smaller, and more expensive (per bit). Within the processor, there is a set of registers that function as a level of memory above main memory and cache in the hierarchy. The registers in the processor perform two roles:

• **User-visible registers**: Enable the machine- or assembly language programmer to minimize main memory references by optimizing use of registers.
• **Control and status registers**: Used by the control unit to control the operation of the processor and by privileged, operating system programs to control the execution of programs.

User-Visible Registers

A user-visible register is one that may be referenced by means of the machine language that the processor executes. We can characterize these in the following categories:

•General purpose
• Data
• Address
• Condition codes

General-purpose registers can be assigned to a variety of functions by the programmer. Sometimes their use within the instruction set is orthogonal to the operation. That is, any general-purpose register can contain the operand for any opcode. This provides true general-purpose register use. Often, however, there are restrictions. For example, there may be dedicated registers for floating-point and stack operations.

In some cases, general-purpose registers can be used for addressing functions (e.g., register indirect, displacement). In other cases, there is a partial or clean separation between data registers and address registers.

Data registers may be used only to hold data and cannot be employed in the calculation of an operand address.

Address registers may themselves be somewhat general purpose, or they may be devoted to a particular addressing mode. Examples include the following:

• Segment pointers: In a machine with segmented addressing, a segment register holds the address of the base of the segment. There may be multiple registers: for example, one for the operating system and one for the current process.
• Index registers: These are used for indexed addressing and may be auto indexed.
• Stack pointer: If there is user-visible stack addressing, then typically there is a dedicated register that points to the top of the stack. This allows implicit addressing; that is, push, pop, and other stack instructions need not contain an explicit stack operand.

A final category of registers, which is at least partially visible to the user, holds **condition codes** (also referred to as **flags**). Condition codes are bits set by the processor hardware as the result of operations. For example, an arithmetic operation may produce a positive, negative, zero, or overflow result. In addition to the result itself being stored in a register or memory, a condition code is also set. The code may subsequently be tested as part of a conditional branch operation. Condition code bits are collected into one or more registers. Usually, they form part of a control register. Generally, machine instructions allow these bits to be read by implicit reference, but the programmer cannot alter them.

Control and Status Registers

There are a variety of processor registers that are employed to control the operation of the processor. Most of these, on most machines, are not visible to the user. Some of them may be visible to machine instructions executed in a control or operating system mode.

Four registers are essential to instruction execution:

• **Program counter (PC)**: Contains the address of an instruction to be fetched
• **Instruction register (IR)**: Contains the instruction most recently fetched

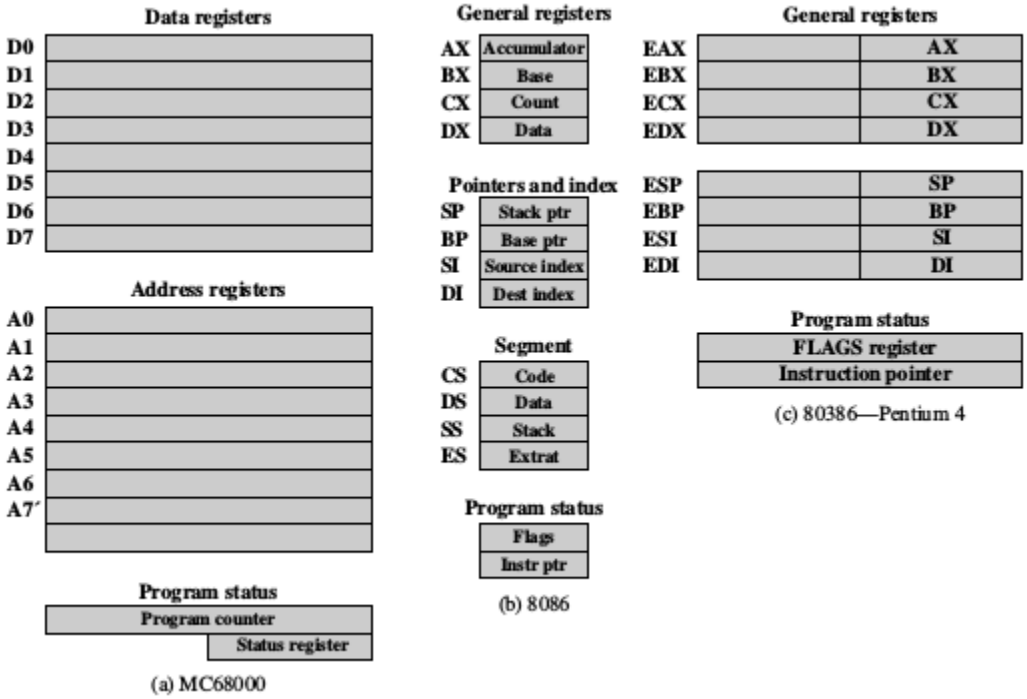
• **Memory address register (MAR)**: Contains the address of a location in memory
• **Memory buffer register (MBR)**: Contains a word of data to be written to memory or the word most recently read

The four registers just mentioned are used for the movement of data between the processor and memory. Within the processor, data must be presented to the ALU for processing. The ALU may have direct access to the MBR and user-visible registers. Alternatively, there may be additional buffering registers at the boundary to the ALU; these registers serve as input and output registers for the ALU and exchange data with the MBR and user-visible registers.

Many processor designs include a register or set of registers, often known as the **program status word (PSW)**, that contain status information. The PSW typically contains condition codes plus other status information. Common fields or flags include the following:

• Sign: Contains the sign bit of the result of the last arithmetic operation.
• Zero: Set when the result is 0.
• Carry: Set if an operation resulted in a carry (addition) into or borrow (subtraction) out of a high-order bit. Used for multiword arithmetic operations.
• Equal: Set if a logical compare result is equality.
• Overflow: Used to indicate arithmetic overflow.
• Interrupt Enable/Disable: Used to enable or disable interrupts.
• Supervisor: Indicates whether the processor is executing in supervisor or user mode. Certain privileged instructions can be executed only in supervisor mode, and certain areas of memory can be accessed only in supervisor mode.

Sample microprocessor register organizations are illustrated bellow.



CONTROL OF THE PROCESSOR

Functional Requirements

The functional requirements for the control unit are those functions that the control unit must perform. A definition of these functional requirements is the basis for the design and implementation of the control unit. The following three-step process leads to a characterization of the control unit:

1. Define the basic elements of the processor.
2. Describe the micro-operations that the processor performs.
3. Determine the functions that the control unit must perform to cause the micro-operations to be performed.

The basic functional elements of the processor are the following:

- ALU
- Registers
- Internal data paths
- External data paths
- Control unit

The ALU is the functional essence of the computer. Registers are used to store data internal to the processor. Some registers contain status information needed to manage instruction sequencing (e.g., a program status word). Others contain data that go to or come from the ALU, memory, and I/O modules. Internal data paths are used to move data between registers and between register and ALU. External data paths link registers to memory and I/O modules, often by means of a system bus. The control unit causes operations to happen within the processor. The execution of a program consists of operations involving these processor elements. These operations consist of a sequence of micro-operations.

All micro-operations fall into one of the following categories:

- Transfer data from one register to another.
- Transfer data from a register to an external interface (e.g., system bus).
- Transfer data from an external interface to a register.
- Perform an arithmetic or logic operation, using registers for input and output.

All of the micro-operations needed to perform one instruction cycle, including all of the micro-operations to execute every instruction in the instruction set, fall into one of these categories.

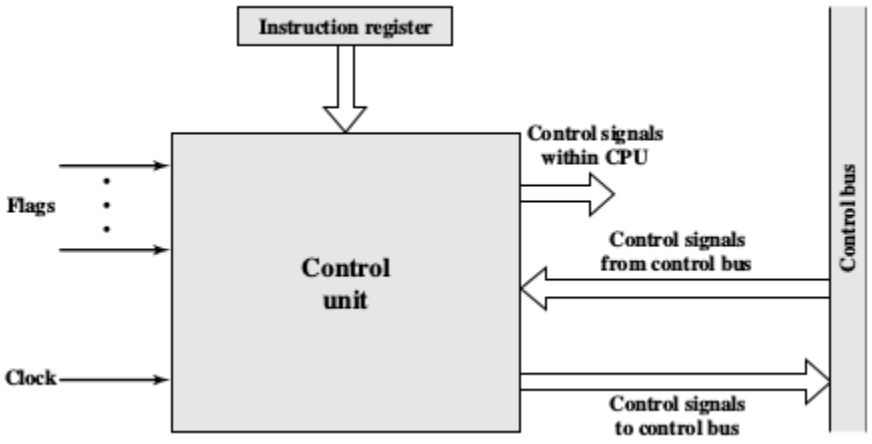
The control unit performs two basic tasks:

- **Sequencing**: The control unit causes the processor to step through a series of micro-operations in the proper sequence, based on the program being executed.
- **Execution**: The control unit causes each micro-operation to be performed.

The preceding is a functional description of what the control unit does. The key to how the control unit operates is the use of control signals.

Control Signals

The following figure shows a general model of the control unit, showing all of its inputs and outputs.



The inputs are,

• Clock: This is how the control unit "keeps time." The control unit causes one micro-operation (or a set of simultaneous micro-operations) to be performed for each clock pulse. This is sometimes referred to as the processor cycle time, or the clock cycle time.
• Instruction register: The opcode and addressing mode of the current instruction are used to determine which micro-operations to perform during the execute cycle.
• Flags: These are needed by the control unit to determine the status of the processor and the outcome of previous ALU operations. For example, for the increment-and-skip-if-zero (ISZ) instruction, the control unit will increment the PC if the zero flag is set.
• Control signals from control bus: The control bus portion of the system bus provides signals to the control unit.

The outputs are as follows:

• Control signals within the processor: These are two types: those that cause data to be moved from one register to another, and those that activate specific ALU functions.
• Control signals to control bus: These are also of two types: control signals to memory, and control signals to the I/O modules.

Three types of control signals are used: those that activate an ALU function, those that activate a data path, and those that are signals on the external system bus or other external interface. All of these signals are ultimately applied directly as binary inputs to individual logic gates.

The control unit keeps track of where it is in the instruction cycle. At a given point, it knows that the fetch cycle is to be performed next. The first step is to transfer the contents of the PC to the MAR.The control unit does this by activating the control signal that opens the gates between the bits of the PC and the bits of the MAR. The next step is to read a word from memory into the MBR and increment the PC.The control unit does this by sending the following control signals simultaneously:

- A control signal that opens gates, allowing the contents of the MAR onto the address bus
- A memory read control signal on the control bus
- A control signal that opens the gates, allowing the contents of the data bus to be stored in the MBR
- Control signals to logic that add 1 to the contents of the PC and store the result back to the PC

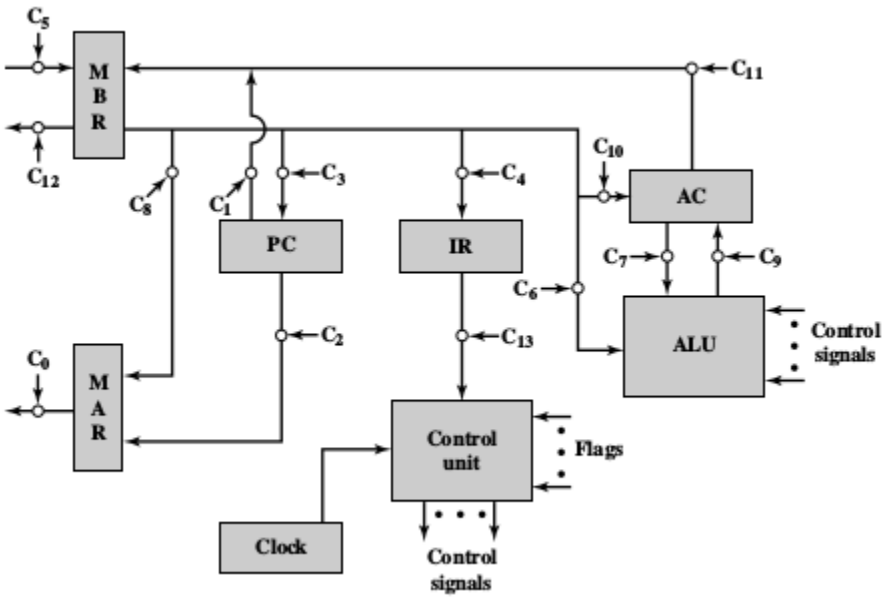
Following this, the control unit sends a control signal that opens gates between the MBR and the IR.

This completes the fetch cycle except for one thing: The control unit must decide whether to perform an indirect cycle or an execute cycle next. To decide this, it examines the IR to see if an indirect memory reference is made.

The indirect and interrupt cycles work similarly. For the execute cycle, the control unit begins by examining the opcode and, on the basis of that, decides which sequence of micro-operations to perform for the execute cycle.

A Control Signals Example

Figure below illustrates the example.



This is a simple processor with a single accumulator (AC). The data paths between elements are indicated. The control paths for signals emanating from the control unit are not shown, but the terminations of control signals are labeled C i and indicated by a circle. The control unit receives inputs from the clock, the instruction register, and flags. With each clock cycle, the control unit reads all of its inputs and emits a set of control signals. Control signals go to three separate destinations:

• Data paths: The control unit controls the internal flow of data. For example, on instruction fetch, the contents of the memory buffer register are transferred to the instruction register. For each path to be controlled, there is a switch (indicated by a circle in the figure). A control signal from the control unit temporarily opens the gate to let data pass.
• ALU: The control unit controls the operation of the ALU by a set of control signals. These signals activate various logic circuits and gates within the ALU.
• System bus: The control unit sends control signals out onto the control lines of the system bus (e.g., memory READ).

The control unit must maintain knowledge of where it is in the instruction cycle. Using this knowledge, and by reading all of its inputs, the control unit emits a sequence of control signals that causes micro-operations to occur. It uses the clock pulses to time the sequence of events, allowing time between events for signal levels to stabilize. Table below indicates the control signals that are needed for some of the micro-operation sequences described earlier. (For simplicity, the data and control paths for incrementing the PC and for loading the fixed addresses into the PC and MAR are not shown.)

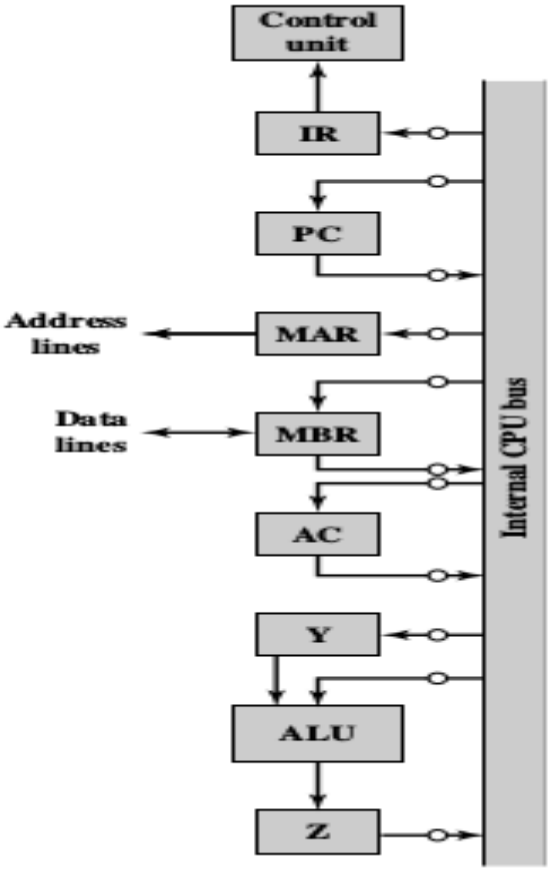
	Micro-operations	Active Control Signals
Fetch:	t_1: MAR ← (PC)	C_2
	t_2: MBR ← Memory PC ← (PC) + 1	C_5, C_R
	t_3: IR ← (MBR)	C_4
Indirect:	t_1: MAR ← (IR(Address))	C_8
	t_2: MBR ← Memory	C_5, C_R
	t_3: IR(Address) ← (MBR(Address))	C_4
Interrupt:	t_1: MBR ← (PC)	C_1
	t_2: MAR ← Save-address PC ← Routine-address	
	t_3: Memory ← (MBR)	C_{12}, C_W

C_R = Read control signal to system bus.
C_W = Write control signal to system bus.

It is worth pondering the minimal nature of the control unit. The control unit is the engine that runs the entire computer. It does this based only on knowing the instructions to be executed and the nature of the results of arithmetic and logical operations (e.g., positive, overflow, etc.). It never gets to see the data being processed or the actual results produced. And it controls everything with a few control signals to points within the processor and a few control signals to the system bus.

Internal Processor Organization

The CPU with internal bus can be illustrated as below.



A single internal bus connects the ALU and all processor registers. Gates and control signals are provided for movement of data onto and off the bus from each register. Additional control signals control data transfer to and from the system (external) bus and the operation of the ALU.

Two new registers, labeled Y and Z, have been added to the organization. These are needed for the proper operation of the ALU. When an operation involving two operands is performed, one can be obtained from the internal bus, but the other must be obtained from another source. The AC could be used for this purpose, but this limits the flexibility of the system and would not work with a processor with multiple general-purpose registers. Register Y provides temporary storage for the other input. The ALU is a combinatorial circuit with no internal storage. Thus, when control signals activate an ALU function, the input to the ALU is transformed to the output. Thus, the output of the ALU cannot be directly connected to the bus, because this output would feed back to the input. Register Z provides

temporary output storage. With this arrangement, an operation to add a value from memory to the AC would have the following steps:

t1: MAR ← (IR(address))

t2: MBR ← Memory

t3: Y ← (MBR)

t4: Z ← (AC) + (Y)

t5: AC ← (Z)

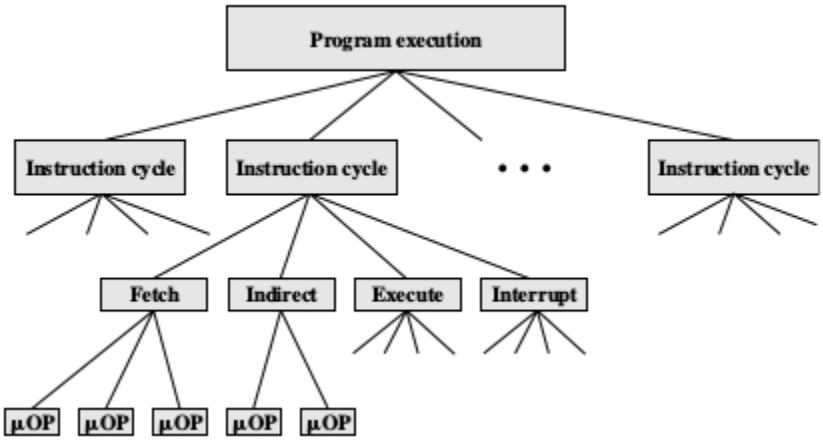
The use of common data paths simplifies the interconnection layout and the control of the processor. Another practical reason for the use of an internal bus is to save space.

MICRO-OPERATIONS

In computer central processing units, **micro-operations** (also known as micro-ops) are the functional or atomic, operations of a processor. These are low level instructions used in some designs to implement complex machine instructions. They generally perform operations on data stored in one or more registers. They transfer data between registers or between external buses of the CPU, also performs arithmetic and logical operations on registers.

In executing a program, operation of a computer consists of a sequence of instruction cycles, with one machine instruction per cycle. Each instruction cycle is made up of a number of smaller units – *Fetch*, *Indirect*, *Execute and Interrupt cycles*. Each of these cycles involves series of steps, each of which involves the processor registers. These steps are referred as micro-operations. the prefix micro refers to the fact that each of the step is very simple and accomplishes very little.

Figure below depicts constituent elements of a program execution.



The Fetch Cycle

The fetch cycle occurs at the beginning of each in-]struction 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.

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 time. Symbolically, we can write this sequence of events as follows:

t 1 : MAR ← (PC)

t 2 : MBR ← Memory

PC ← (PC) + I

t 3 : IR ← (MBR)

where I is the instruction length. 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 (t_1, t_2, t_3) 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 I the contents of the PC.
- Third time unit: Move contents of MBR to IR.

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 withut affecting the fetch operation:

t_1 : MAR ← (PC)

t_2 : MBR ← Memory

t_3 : PC ← (PC) + I

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.

The Indirect Cycle

Once an instruction is fetched, the next step is to fetch source operands. If the instruction specifies an indirect address, then an indirect cycle must precede the execute cycle. It includes the following micro-operations:

t_1 : MAR ← (IR(Address))

t_2 : MBR ← Memory

t_3 : 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.

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 have the following sequence for the interrupt cycle.

t 1 : MBR ← (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 micro-operation. However, because most processors provide multiple types and/or levels of interrupts, it may take 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.

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 case, the same micro-operations are repeated each time around. This is not true of the execute cycle. Because of the variety opcodes, there are a number of 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 R1. The following sequence of micro-operations might occur:

t 1 : MAR ← (IR(address))

t 2 : MBR ← Memory

t 3 : R1 ← (R1) + (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.

Let us look at two more complex examples.

A common instruction is increment and skip if zero:

ISZ X

The content of location X is incremented by 1. If the result is 0, the next instruction is skipped. A possible sequence of micro-operations is,

t 1 : MAR ← (IR(address))

t 2 : MBR ← Memory

t 3 : MBR ← (MBR) + 1

t 4 : Memory ← (MBR)

If ((MBR) = 0) then (PC ← (PC) + I)

The new feature introduced here is the conditional action. The PC is incremented if (MBR) = 0. This test and action can be implemented as one micro-operation. Note also that this micro-operation can be performed during the same time unit during which the updated value in MBR is stored back to memory.

Finally, consider a subroutine call instruction. As an example, consider a branch-and-save-address instruction:

BSA X

The address of the instruction that follows the BSA instruction is saved in location X, and execution continues at location X + I. The saved address will later be used for return. This is a straightforward technique for providing subroutine calls. The following micro-operations suffice:

t 1 : MAR ← (IR(address))

MBR ← (PC)

t 2 : PC ← (IR(address))

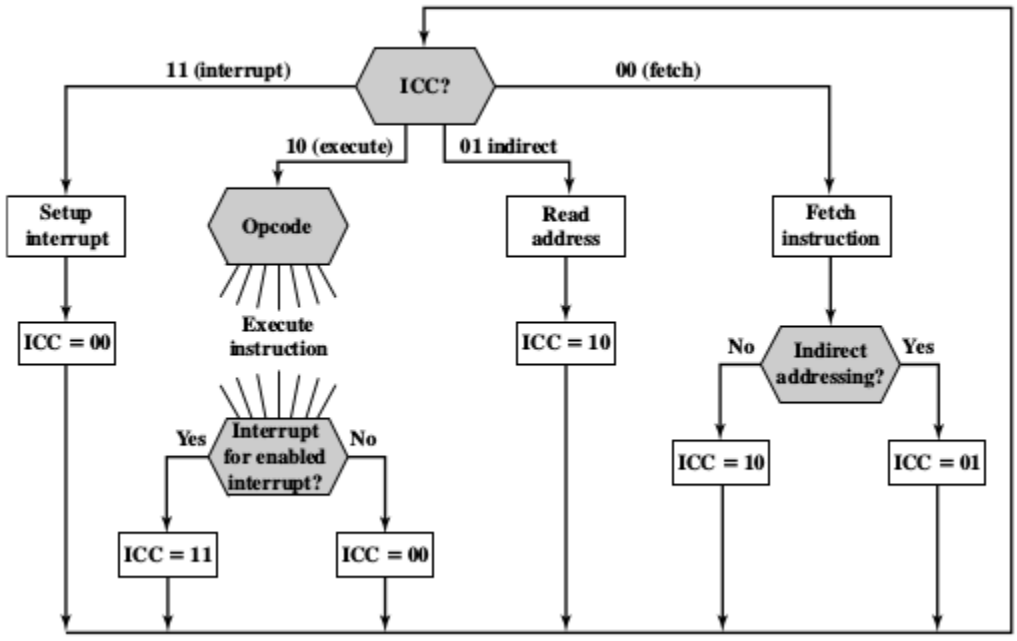
Memory ← (MBR)

t 3 : PC ← (PC) + I

The address in the PC at the start of the instruction is the address of the next instruction in sequence. This is saved at the address designated in the IR. The latter address is also incremented to provide the address of the instruction for the next instruction cycle.

The Instruction Cycle

Figure below illustrates the flowchart for instruction cycle.



We assume a new 2-bit register called the instruction cycle code (ICC). The ICC designates the state of the processor in terms of which portion of the cycle it is in:

00: Fetch

01: Indirect

10: Execute

11: Interrupt

At the end of each of the four cycles, the ICC is set appropriately. The indirect cycle is always followed by the execute cycle. The interrupt cycle is always followed by the fetch cycle. For both the fetch and execute cycles, the next cycle depends on the state of the system.

Thus, the flowchart defines the complete sequence of micro operations, depending only on the instruction sequence and the interrupt pattern.