26. What is the difference between write through and write back methods of writing into the cache memory? Which method is more advantageous? Explain.
(Hint: Write-Back method is more advantageous).
27. What are the differences between real and virtual memory? What support do you need from the processor to implement virtual memory technique?
28. Diagrammatically describe and explain floppy disk and magnetic tape along with their working concepts.
29. Draw & explain the basic structure of a hard disk & explain seek time & latency time associated with it.
30. Explain how cache memory increases the performance of a computer system.
31. Explain the difference between full associative and direct mapped cache mapping approaches.
32. Diagrammatically explain magnetic recording.
33. What is a virtual memory? Why is it so called? What do you mean by logical address space and physical address space?
34. Explain with an example how logical address is converted into physical address.
35. Write the advantages of virtual memory system.
36. Explain the need of auxiliary memory devices. How are they different from main memory?
37. Differentiate between tape drive and magnetic disk.
38. What is the limitation of direct mapping method? Explain with an example how it can be improved in set-associative mapping.
39. What is a translation look-aside buffer (TLB)? Explain how it works.

Chapter 5

CONTROL UNIT

Overview

- Central Processing Unit
- Clock, Clock Cycles and Timing Diagrams
- Instruction Cycle
- Control Unit
- Hardwired Control Unit
- Microprogrammed Control Unit

5.1 Central Processing Unit

CPU or Central Processing Unit performs the bulk of data processing operations in a computer. It is responsible for actually executing the set of instructions that make up the programs and also the operating system.

Different Components of CPU:
Processors are made up of several building blocks like execution units, register files and control logic as shown in the figure 1.

(a) Executing units:
These units contain the instruction execution hardware like hardware that fetches and decodes the instructions, along with the arithmetic logic units (ALUs) that perform the actual computations.

(b) Register Files:
Register file is a small storage area for the data that is being used by the processor. Such files allow multiple simultaneous accesses and values stored in register files that can be accessed more quickly compared to the data stored in the memory system.

(c) Control Logic:
Such logic controls the rest of the processor (i.e. its operations). It determines when and how instructions can be executed i.e. what are the operations needed to execute each instruction.

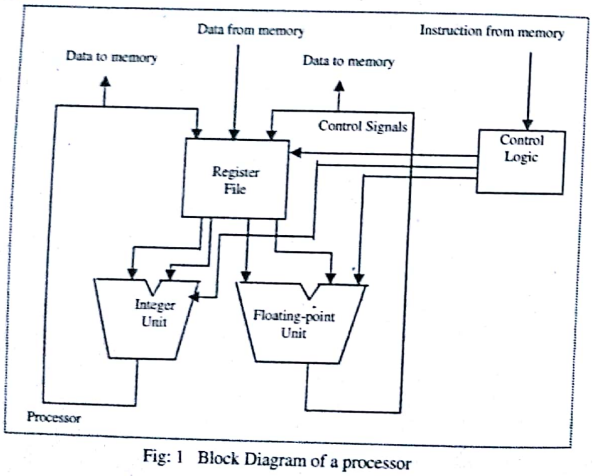


Fig: 1 Block Diagram of a processor

Registers:
A register consists of a group of flip-flops. Each of the flip-flops in the group stores one bit of information. So a k-bit register (having group of k flip-flops) can store any binary information of k-bits. In a register there are also multiple combinational gates to perform data-processing tasks. CPU has registers of either special-purpose or general-purpose types.

(a) Special-Purpose Registers:
The special-purpose registers are those that are used to store specific type of instructions and data. These are PC, AR, DR, IR, INPR and OUTR.

(b) General-Purpose Registers:
The general-purpose registers are used for storing any kind of data temporary during any processing. These are R1, R2,…,R(n-1) etc.
Accumulator (AC) though is a general-purpose register but can be sometimes used for specific purposes.

List of different registers along with their respective functions needed for a basic computer:
The different registers in a computer are:

(a) Data Register (DR):
This register holds the memory operands (or data) when they are fetched from the memory. It is a 16-bit register.

(b) Address Register (AR):
This register holds the addresses of the memory locations to be accessed. It is a 12-bit register.

(c) Instruction Register (IR):
This register holds the instructions read from the memory. It is a 16-bit register.

(d) Accumulator (AC):
This is a general-purpose processing register, which is used for arithmetic and logical operations. It is a 16-bit register.

(e) Temporary Register (TR):
This register holds all sorts of temporary data during any processing. It is a 16-bit register.

(f) Program Counter (PC):
This register holds the address of the next instruction to be fetched (or read) from the memory after the current instruction is executed. It is a 12-bit register.

(g) Input Register (INPR):
This register holds input characters, which it receives from an input device. It is an 8-bit register.

(h) Output Register (OUTR):
This register holds output characters to be sent to an output device. It is an 8-bit register.

Flag / Status Register:
This is a register in which the different status flags are stored.

Significance of a flag-register:
Flag registers store status bits called the condition-code bits or flag bits. C, S, Z and V are four status bits that are set or cleared as a result of an operation performed in the ALU.

(a) ***Bit C (carry)*** is set to 1 if the end carry is 1, else if the end carry is 0 then it is cleared to 0.
(b) ***Bit S (sign)*** is set to 1 provided the highest order bit is 1, else if the bit is 0 then it is cleared.
(c) If the output of the ALU contains all 0's then ***bit Z (zero)*** is set to 1, else it is cleared.
(d) If the Ex-OR of the last two carries is equal to 1 then ***bit V (overflow)*** is set to 1, else it is cleared to 0. This indicates the overflow condition when negative numbers are in 2's complement.

5.2 Clock

A clock is a uniformly spaced, identically shaped, repetitive, regular sequence of clock pulses that are generated by a timing device (a timer or a clock).

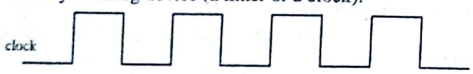


Fig: 2

As seen in figure 2, all low levels are uniformly spaced; all high levels are uniformly spaced. All leading edges & trailing edges must be identically shaped. So, a clock signal (generated by a clock or timer) helps to synchronize the events related to it.

Clock Pulse / Pulse:
A clock pulse is a signal (say a voltage signal) that has got 2 levels. One level is the ground level or the low level (binary 0-signal level) and the other level is the high level (binary 1 signal level). Generally the pulse remains in one level, goes to the other level and stays there for sometime & again comes back to the previous level.

Clock Cycle:
The clock period (T, as shown in the figure 3) from one leading edge till the next leading edge or from one trailing edge to the next trailing edge is termed as a clock cycle.

T-State:
A T-State is one clock period.
A clock frequency = 1 / clock period = 1 / T.

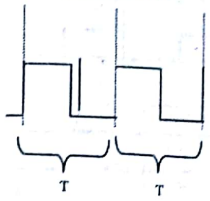


Fig: 3

Machine Cycle:
One or multiple clock cycles make a machine cycle.
Example: Generally one memory read (i.e. CPU placing address on address bus and then memory sending 'data' back to the CPU via the data bus) or one memory write (i.e. CPU placing 'data' to be written on the data bus, the address of the memory location where data is to be written on the address bus and then sending 'write' pulse on the control bus) cycle constitute a machine cycle.
So, these sequences may occur within one clock cycle or may take multiple clock cycles.

Instruction Cycle:
An Instruction Cycle consists of one or multiple machine cycles.
Example: A sequence of operations involved in processing an instruction constitutes an instruction cycle. It has 2 major phases:
(i) fetch cycle: during this phase instruction is obtained from the main memory.
(ii) execution cycle: this phase includes decoding the instruction, fetching any required operands, performing the operation specified by the instruction's opcode.

5.3 Timing Diagrams

Timing Diagrams show the timing relationship that must exist between the control signals and the data in the buses.

Timing Diagrams to show the synchronous data transfer between main memory and CPU:
In synchronous data transfer the timing signal is issued by the CPU clock i.e. memory operates according to the CPU clock. This type of data transfer occurs mainly between CPU & main memory.

Synchronous data transfer is mainly of two types:

(a) Synchronous Memory Read (i.e. CPU reading from memory):

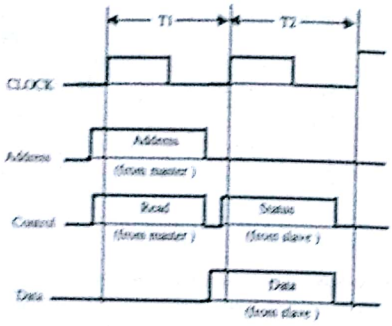


Fig: 4

Steps:

1) As shown in figure 4, in the 1st clock period, (T1) CPU sends the address of the memory location to be read, in the address bus.
2) At the same time (in T1), CPU places 'read' signal in the control line.
3) On getting the 'read' signal as well as the address from the CPU, memory sends back the required data to the CPU via the data bus in the next (T2) clock period. This is because, there is sufficient delay in main memory in receiving the CPU signals, searching for the required data and then placing the data in the data bus. So, data is sent back in the next clock period.
4) The 'status' signal is optional. It indicates whether the main memory is ready to response to the CPU request or not.

So, only if the main memory is ready to response, it will place a 'status' signal high on the control bus on getting the CPU request. Then the main memory will place the data. Hence if the main memory in not ready to response, it will not place the data on the data bus.

(b) Synchronous Memory Write (i.e. CPU writing to the memory):

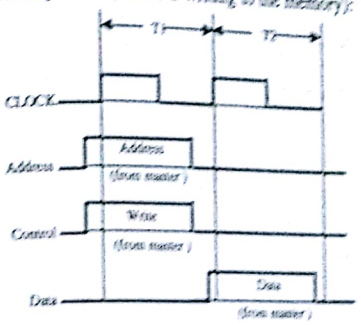


Fig: 5

Steps:

1) CPU sends 'address' of the memory location to write, in the 1st T-state (T1).
2) CPU then sends the 'write' control signal.
3) CPU, after some delay, sends 'data' to be written, in the next (T2) clock period / T2 state.

[Note: The timing diagrams showing the synchronous data transfer between main memory and CPU is explained in the 'Input-Output (I/O)' chapter]

Timing Diagram to show the Register Transfer of Data:

Suppose that MOV B, C is an instruction initiating data transfer operation. The corresponding microoperation for this instruction is [(B) ←(C)]. This transfer occurs over the CPU's internal bus (assumed to be an 8-bit bus).

Figure 6, shows the above operation.

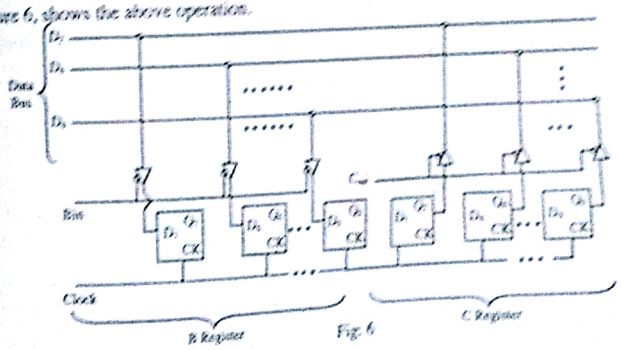


Fig: 6

The corresponding timing diagram is:

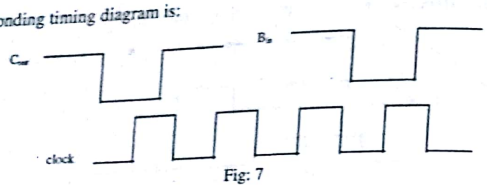


Fig: 7

5.4 Instruction Cycle

Basic Phases of an Instruction Cycle:

The basic phases of an instruction cycle (generally it consists of instruction fetch cycle and instruction execution cycle) are:

(a) To fetch an instruction from the memory.
(b) To decode the instruction.
(c) If it is an indirect address instruction, then the effective address is to be read from the memory (i.e. the operands needed to be fetched from the memory one by one).
(d) To execute the instruction.

The first three phases constitute the instruction fetch cycle and the last phase is the instruction execution cycle. On completion of step (d), the control again goes back to the step (a) to repeat the same cycle for the next instruction. This continues until a HALT instruction in encountered.

Detailed Explanation of the Instruction Cycle of any Processor:

The instruction cycle has got two main phases: the instruction fetch phase (this has three sub phases) and instruction execution phase.

Flowchart:

The flowchart shows the instruction cycle of any processor.

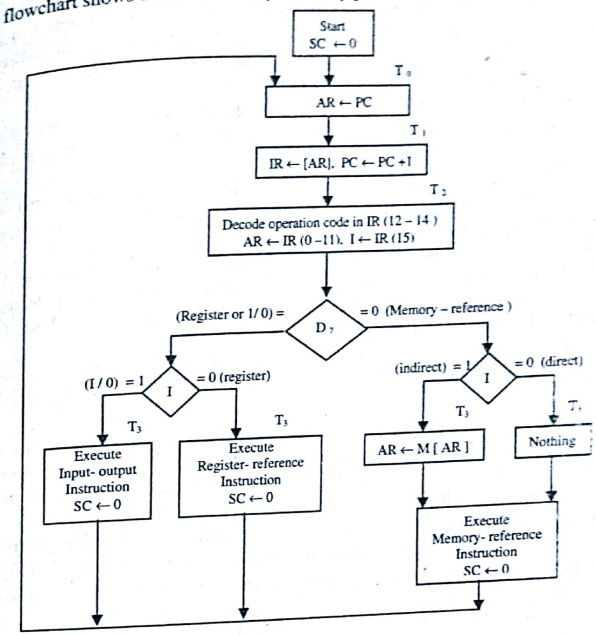


Fig: 8

The different steps of an instruction cycle are as follows:

Steps:

(a) The address of the first instruction in the program is loaded in the PC.
(b) A sequence counter (SC) is kept to track the different steps i.e. to keep track whether the steps are performed sequentially and correctly. The SC is incremented by one after each clock pulse (clock period). So, initially the SC is cleared to 0.
(c) In the first clock period (T_0), the contents of PC are placed to the AR (i.e. memory address register or MAR).
(d) In the second clock period (T_1), the content of the desired memory location (as given in the PC) is placed in the IR. Also the PC is loaded with the address of the next instruction.

(e) In (T_2), the IR content is decoded (i.e. the opcode of the instruction fetched from memory is decoded).
The flip-flop I hold the indirect bit [this specifies whether the instruction is input-output reference or register reference or memory reference (direct or indirect) in nature]. The address part of the operands (to be fetched from memory) are again transferred to the AR.

(f) Depending on the I-bit of the instruction, it is decided whether the instruction is register reference or input-output reference or direct-indirect memory reference in nature.
So, if the opcode is 111 then it is either a register reference or a input-output reference instruction, else it is a memory reference instruction (i.e. opcode varies from 000 through 110).

(g) Provided it is a input-output reference instruction (i.e. if opcode is 111 and I = 1), then at (T_3), the input-output instruction is executed. After this the SC is again cleared to 0 (as it completes the current instruction cycle and the control goes back to the beginning of the next instruction cycle).

(h) Also if it is a register reference instruction (i.e. if opcode is 111 and I = 0), then at (T_3), the register reference instruction is executed. After this, similarly, the SC is cleared to 0.

(i) On the other hand, if it is a direct memory reference instruction (i.e. if opcode varies from 000 through 110 but I = 0), then it is not needed to do anything as the operand's effective address is already there in AR and so the instruction can be directly executed. Then again the SC is to be cleared to 0.

(j) For indirect memory reference instruction (opcode varies from 000 through 110 but I = 1), it is needed to fetch the effective address again from the memory. So only after fetching the actual operand from the memory, the instruction can be executed. After this the SC is cleared to 0.
Steps (i) and (j) are performed at clock period (T_3). However for indirect memory reference instructions it may continue till T_4.

(k) Hence on completion of execution of the current instruction, SC is cleared and the address of the next instruction gets loaded in the PC and the same cycle continues.

[*Note:* Each of the micro-operations in the entire cycle are performed with the help of control signals generated either by hardware control unit or by microprogrammed control unit. So all microperations during the fetch cycle are basically set of control signals and all microperations during the execution cycle are also set of control signals. Though the set of control signals during the fetch phases are generally the same for any kind of instruction, the set of control signals generated in the execution phases are different for different instructions.]

5.5 Micro Operations

Micro operations are elementary operations performed with the data stored in one or multiple registers i.e. micro operations are the operations executed on the data stored in registers.

Different types of micro operations:
The different types of micro operations are as follows:
(a) ***Register Transfer Microoperations:***
Such micro operations transfer binary information from register to another.
(b) ***Arithmetic Microoperations:***
Such micro operations perform all kinds of arithmetic operations on numeric data that are stored in registers.
(c) ***Logic Microoperations:***
Such micro operations perform all kinds of bit manipulation operations **on non-numeric data stored in registers.**
(d) ***Shift Microoperations:***
Such micro operations perform shift operations on data that are stored in registers.

5.6 Control Unit

This is an important part of CPU. Control Unit interprets & sequences instructions. It coordinates the activities of other units in a computer and sends control signals to them as well as senses their states.

Control Memory:
Control memory is the memory that is part of the control unit or that remains within the control unit.

Control Signals:
These are the set of signals generated by the control unit to perform the various microoperations.
Control signals are generated either by hardware control unit (i.e. by means of hardware) or by microprogrammed control unit (i.e. by means of software or microinstructions).
In other words it can also be stated as, there are two types of control unit designs: one is ***hardwired control unit*** where the control signals are generated by means of hardware, and, ***microprogrammed control unit***, where the control signals are generated by means of software.

5.7 Hardwired Control Unit

In a hardwired control unit, all control signals are generated by means of hardware using conventional simple logic design techniques. Each step in the sequence of control signals is executed in one clock period.

Basic units of a hardwired control unit:

The basic units of the hardwired control are:

(a) A clock
(b) A counter
(c) A decoder
(d) An encoder

Along with these, contents of the condition code flags & external input signals like interrupt requests are also required.

Explanation of how the different functional units of a hardwired control unit, helps in generating an individual control signal:

Diagram:

Consider figure 9, below:

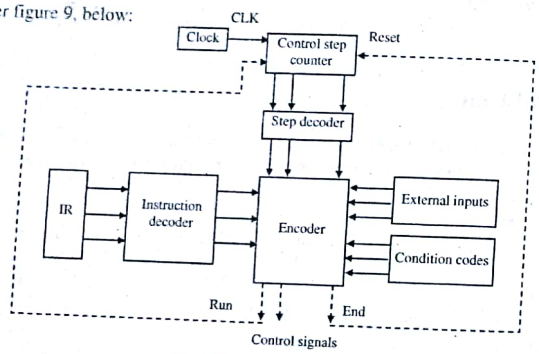


Fig: 9 Hardwired Control Unit

Each microprogram comprises of a sequence of microoperational steps. Each microoperational steps comprise of one or more microoperations and needs a number of control signals to be activated. The control signals needed to execute each microoperational step are generated simultaneously.

So, in each clock state a microoperational step is performed i.e. in each clock state, the necessary control signals are generated & the corresponding microoperations are performed.

The different units are:

(a) Control step counter:

This unit keeps track of the clock states i.e. each state or count of this counter corresponds to one control step.

So, when the counter counts 0 it means that the control signals in the 1st clock state are executed; when counter = 1, control signals in the 2nd clock state are executed and so on.

Now, it is known that executing an instruction needs execution of the respective microoperational steps and in each clock state a microoperational step is executed. Also the counter keeps track of the clock states.

It is needed that the counter must have a modulus (example: a module counter) which is greater than the number of clock states required for the execution of the longest instruction i.e. the counter must count through the number of steps, greater than the number of clock states (or number of microoperational steps) required for the execution of the longest instruction (i.e. the largest number of clock states or the largest number of microoperational steps). Suppose there are maximum 'n' clock states, T_1, T_2, ... T_n for an instruction.

(b) Step decoder:

The output lines from the step decoder indicates all the 'n' number of clock States, T_1, T_2, ... T_n.

Now, depending on the control step counter, only the required output line of the step decoder is enabled i.e. if suppose, the counter counts 0, then the 1st clock state or microoperational step is executed. So only the T_1 line of the step decoder is enabled; for counter count=1, only the T_2 line of the step decoder is enabled and so on.

(c) Instruction Register (IR):

This holds the instruction fetched from the main memory.

The opcode portion of the IR, holds the instruction opcode and is decoded by the Instruction decoder. The output of the opcode portion of IR consists of a separate line for each machine instruction i.e. a separate line for all machine instructions like ADD, SUB, MUL, MOV, ...etc.

(d) Instruction Decoder:

It decodes the machine instruction in the opcode part of the instruction register. Like the input, the output of the instruction decoder also consists of separate lines for each machine instruction.

So depending on the opcode portion of the machine instruction in the IR, one of the output lines of the decoder i.e. INS_1, through INS_m (where 'm' is the total number of machine instructions in the main memory) is enabled (set to 1) and all other lines are disabled.

(e) Encoder:

Input signals to the encoder block consist of step decoder output, instruction decoder output, external inputs and condition codes.

All these input signals in the encoder block are combined to generate the individual control signals Y_{in}, PC_{out}, Add, End etc. i.e. the output of the encoder consists of a separate line for each control signal and the encoder offers encoding all the input signals, generates the individual control signals (or enables the respective control signal line).

Remember: Output of the encoder consists of a separate line for all possible control signals.

Example:

Suppose, Intel machine has 'i' number of control signals. So, in Intel, output of the encoder will have 'i' number of control signal lines.

(f) External inputs:

These constitute of the interrupt signals, the MFC (Memory function complete i.e. memory access operations complete) and other external signals.

The output of the external input block consists of a separate line for each external input signal and only the required line is enabled i.e. if there is an interrupt, the particular external interrupt output line is enabled; for a MFC operation, only the MFC output line is enabled and so on.

(g) Condition codes:

The output of this block consists of separate lines for each of the conditional codes (i.e. JZ, JNZ, JC, JNC and all other conditions for a branch instruction) and only the required line is enabled.

The '***end***' control signal ('end' is the last control signal to be executed in any sequence of microoperational steps) is always generated on the last clock state or the counter's last count step. So after that, the counter is to be reset.

After resetting the counter to execute the next sequence of microoperational steps, '***run***' control signal must be the first one to get executed.

i.e. end ⟶ reset ⟶ run

(execution of current counter (start execution of microoperational steps) next micooperational steps)

5.8 Microprogrammed Control Unit

This is a control unit whose binary variable (i.e. the control functions that specifies a microoperation) remains stored in memory i.e. such type of control unit is software (i.e., microinstruction) based.

Control Words:

Control words are words whose bits are used to control certain specified microoperations or general operations. These are basically string of 1's and 0's. Each of the bits in different control words generates different microoperations related to the instruction. Control words are generally stored within control memory.

Routine:

Meaningful sequence of instructions is called a routine (or a program).

Microroutine:

Microinstructions are stored in control memory in groups. Each of these groups specifies a ***microroutine.*** So a meaningful sequence of microinstructions constitutes a microroutine. Now the microinstructions within a microroutine must be sequenced and there must be options of branching from one routine to another.

Microinstructions:

Microinstruction is an instruction whose bits carry out a set of microoperations at the same time.

Microprogram:

A meaningful sequence of microinstructions constitutes a ***microprogram.***

Basic Idea Behind the Microprogrammed Control Unit:

In this type of control unit, the control signals are generated by means of software. Microprogrammed control unit is microinstructions based.

Explanation of the general organization of a Microprogrammed Control Unit:

Figures 10(a) and (b) shows the general organization of a microprogrammed control unit.

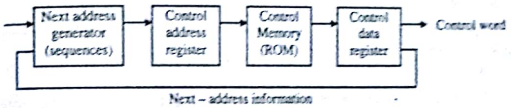


Fig: 10 (a)

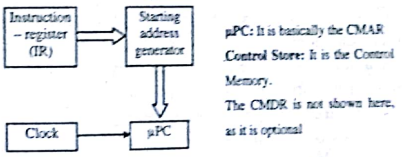


Fig: 10 (b)

A microprogrammed control unit will have the following parts:

1) ***Control Memory Address Register (CMAR):***
   Just like the main memory address register (MAR), inside the control unit there is the CMAR. It specifies the address of the microinstruction to be executed. CMAR holds, in the control memory, the starting address of the microinstruction, to be executed.
2) ***Next Address Generator (sequencer):***
   It is sometimes also called the microprogram sequencer. It determines the address sequence of the microinstructions that is read from control memory i.e. in the opcode

portion of the Instruction Register (i.e. opcode of IR), the opcode of the instruction fetched from the main memory is kept. The opcode is then decoded and thus the starting address of the first microinstruction in the control memory is found. The starting address then comes to the next address generator. So the generator determines the address sequences, i.e. as soon as one microinstruction is executed, the generator generates the address of the next microinstruction and transfers it to the CMAR.

So, the typical *functions of the generator* are:

a) It increments the CMAR by one (i.e. loads one address after another into the CMAR).

b) Loads into the CMAR, the next address of the microinstruction in the control memory to be executed.

An initial address gets loaded in the generator to start the control operations.

3) ***Control Memory Data Register (CMDR):***

This acts just like the main memory data register. CMDR holds the present microinstruction read from the control memory while the next address is computed meanwhile and read from control memory.

The data register is sometimes called a pipeline register. This is because, at a time, it allows the execution of the microoperations specified by the control word (i.e. it holds the control word fetched from the control memory and allows it to get executed) and it also allows the generation of the next microinstruction. So, it performs two works simultaneously.

4) ***Control Memory:***

A computer with a microprogrammed control unit has two separate memories: a main memory & a control memory.

The control memory present inside the control unit is assumed to be a ROM which holds fixed microprograms that cannot be altered by occasional users (i.e. while designing a control memory for a particular computer, say an Intel machine the designer knows that in the Intel machine's instruction set, there are 'n' number of instructions. So, for these 'n' instructions, the designer may develop, say 'nth' number of microinstructions or 'n' number of microprograms and stored them inside the control memory. Now while executing an instruction, only the microinstructions meant for that particular instruction gets executed. So everything in the control memory is pre-developed & they cannot be altered occasionally).

The microprogram, consists of microinstructions that specify various internal control signals for execution of register microoperations i.e. each bit of the microinstruction fetched from the control memory performs some specific logic microoperations.

Say, the CMDR holds the control word or microinstruction, 11001010, from the control memory. Now, on execution, say the first 1 will make the tri-state buffer high, the 2nd 1 will make the control bus high, the 1st 0 will make the chip select line low and so on i.e. on execution, each of the bits in the control word will perform respective microoperation.

If, suppose the microinstructions in a particular machine are 8-bit wide i.e. there are 8-bits in each microinstruction. So while execution, the output of the CMDR will have 8 separate lines. Each of the lines will be connected to respective logic circuitry to generate the control signal. Depending on the control word, respective individual line will be enabled.

Suppose, the CMDR holds 11000000.

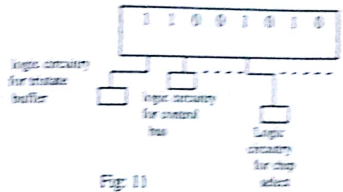


Fig: 11

So, as per figure 11, each of the 8 separate lines (as there are 8-bits in the microinstruction) is connected to different logic circuits. According to the bits in the microinstruction, the respective lines are enabled / disabled and the corresponding microoperations are performed & thus the required control signals are generated.

Working of a Microprogrammed Control Unit:

Steps:

a) Instruction is fetched from the main memory and is stored in the IR.

b) Opcode portion of the IR, which holds the operation part of the instruction, is decoded with a decoder.

c) Decoding the opcode, gives the 1st address or the starting address of the microinstructions in the control memory.

d) The starting address then comes to the 'next address generator'.

e) From there, it goes to the CMAR.

f) Then the control memory is accessed and 1st microinstruction from the starting address location in the control memory is sent to CMDR.

g) The CMDR now holds the 1st microinstruction. Simultaneously, the CMDR asks for the next address information to the next address generator.

h) While asking for the next address information, the CMDR executes the present control word stored in it.

i) On output of the microinstruction, the respective required control signal is generated.

j) Meanwhile, the next address generator on getting 'next-address information' signal from the CMDR, generates the next location address of the next microinstruction in the control memory & sent it to the CMAR.

k) This cycle continues till the execution of the current microprogram (i.e., execution of one instruction) is over.

l) Once execution of one instruction is over, execution starts for the next instruction.

Example:

To execute 'JMP' instruction

i.e. Instruction → JMP

Microinstructions for this instruction:

Diagram:

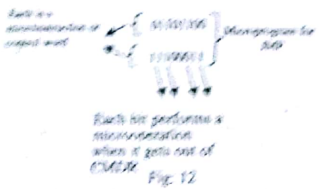


Fig: 12

So, to execute 'JMP', each of the microinstruction is to be executed i.e. each bit of the microinstructions must perform its respective microoperations.

A Microinstruction Format:

The format for an instruction is shown below:

I	Opcode	Address

Now the format for a microinstruction is also the same as the above instruction format:

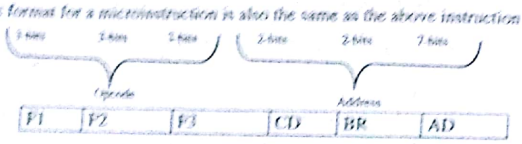


Fig: 13 Microinstruction code format (20 bits)

Explanation of the Different Fields:

(a) F1, F2 and F3 fields:

These are the Microoperation fields i.e. these three fields specify the microoperations to be performed by the computer. The microoperations are subdivided into three fields of three bits each. The three bits in each field are encoded to specify seven distinct microoperations. Thus there are a total of 21 microoperations. So while executing a microinstruction, from each field only one microoperation is to be chosen i.e. total of three microoperations can be chosen for each microinstruction.

(b) CD field:

Condition for branching i.e. if there is any branching in the microinstruction, the condition code gives the condition to be satisfied for the branching.

(c) BR field:

This is the Branch Field i.e. this field indicates what type of branching is to be performed to get the next microinstruction. If it's a sequential execution, then BR is unconditional i.e. just the next microinstruction is to be executed.

(d) AD field:

Address Field. It indicates the location address in the control memory where the next microinstruction to be executed is located.

So if it's a sequential execution then the AD field for the next microinstruction will just be incremented i.e. will just give the next location in the control memory. Else for conditional branching instructions, the next address will depend on the condition.

Different Types of Address Sequencing Methods applied in Microprogrammed Control Unit:

In an instruction set (for a particular machine), each instruction has its own microroutine in the control routine to generate the respective microoperations to execute the particular instruction. The microinstructions in the microroutines may remain stored sequentially in the control memory. However in case of fetching, they may not be fetched sequentially (like if there is a branching then fetching will not be sequential) from the control memory. In a control memory, the address sequencing method is controlled by a hardware that controls the sequence in which the microinstructions within a microroutine are executed. Also the hardware must control the branching from one microroutine to another if needed.

The entire instruction cycle (i.e. all the phases of the instruction cycle) is performed on execution of respective microroutines (set of microinstructions) fetched from the control memory. So, in each phase of the instruction cycle, address-sequencing capabilities of the control memory is required.

So addressing sequencing capabilities required in a control memory are as follows:

a) Incrementing the control address register:

This is needed for sequential access of microinstructions in the control memory. So, in this case the next microinstruction needed (after executing the current microinstruction) is just the next one in the control memory. So the next address generator will fetch the next microinstruction from the control memory by incrementing the control address register by one.

b) Unconditional or conditional branching:

This depends on status bit conditions (status conditions are special bits in the system). The information in these status bits are tested and based on their conditions, different needed actions are performed. Control unit provides decision-making capabilities through the branch logic. The status bits along with the BR field (in the microinsruction format) control the conditional branch decisions that are generated in the control branch logic.

c) A mapping process:
Instruction bits are mapped to their corresponding microinstruction addresses in the control memory.
d) Facility for subroutine call and return:
Provisions for handling subroutine calls are there.

Explanation of how the next Microinstruction Address for the Control Memory is selected while Execution of an Instruction:
Diagram:
The figure 14 shows the block diagram of the control memory and the associated different hardware needed for selecting the next microinstruction address in the control memory.

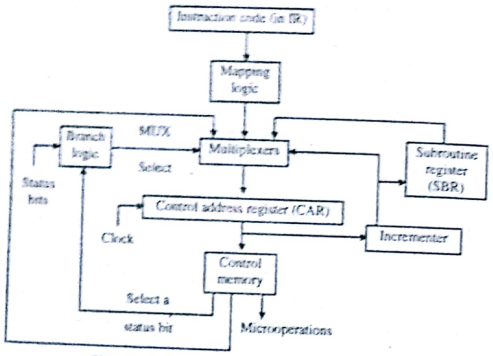


Fig: 14 Selection of next microinstruction address for control memory

Steps to select the next Microinstruction Address:
(a) The opcode portion in the instruction register is decoded. The operation to be performed is thus obtained.
(b) Mapping logic maps the instruction bits to their corresponding microinstruction address in the control memory i.e. the location of the first corresponding microinstruction in the control memory is found out. This microinstruction is first fetched from control memory after its address is placed in the control address register.
(c) As shown in figure 13, in each microinstruction format, the 'AD' field specifies the address of the next microinstruction to be fetched. Now, where the second microinstruction needed is placed in the control memory depends on the BR field and status bits.

(d) The next microinstruction address may be obtained in any way, from among the following possibilities:
(1) If it is just the next microinstruction in the control memory, then only incrementing the first microinstruction address by 1, will fetch the second one.
(2) If it is a conditional branching, then the address of the next microinstruction depends on the BR and CD fields. The particular condition (out of many) is selected by means of branch logic and a multiplexer.
(3) If it is a subroutine call, then also the control branches to the particular microinstruction address accordingly. However in this case, the subroutine register (SBR) must store the return address of the control.
(e) In all the above cases (1, 2 and 3), the branch address of the next microinstruction is placed on the branch address line (depending on the above three conditions).
(f) The control address register (CAR) will get the address of the microinstruction to be fetched from the control memory only after the multiplexer decides (from among the four different paths providing the addresses) which path to enable i.e. address of which path will be send to the CAR.
(g) Once the particular control word is selected and send to the output, the different bits of the control word perform different microoperations as needed to execute the instruction.

Horizontal and Vertical Microinstruction Formats:
The format of the control part of a microinstruction allows us to classify microinstructions into ***horizontal*** and ***vertical.***
(a) Horizontal Microinstruction Format:
In a horizontal microinstruction format, each bit in the control field of the microinstruction is attached to a specific control line. Depending on the number of internal processors, control lines and system bus control lines, are connected to single bits. Execution of horizontal microinstruction needs to activate all control lines, which are high leaving apart those, which are low. So the control signals that are activated in result (for control lines which are high) will execute one or more respective microoperations. Length of horizontal microinstructions ranges between 40 to 100 bits. Horizontal microinstructions control many resources, which operate in parallel. A single horizontal microinstruction might control the simultaneous and independent operation of one or more ALUs, input and output to main memory, conditional next address generation, etc.
Advantages:
Executions of such microinstructions are fast and need less complicated logic circuitry.
Disadvantage:
They need more bits i.e. they less compact.

Diagram:

The figure 15 shows the horizontal microinstruction format:

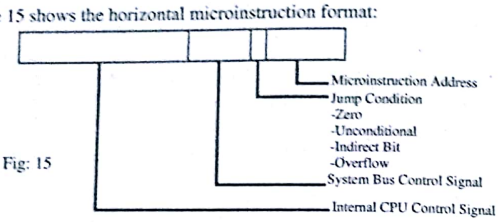


Fig: 15

(a) Horizontal Microinstruction

(b) Vertical Microinstruction Format:

Unlike horizontal microinstruction, in a *vertical microinstruction* each microoperation to get performed needs specific codes (e,g MAR ← [R1], MAR← PC etc.) (as has been shown in the diagram). Such codes get translated into individual control signals by the decoder in the control unit. Length of vertical microinstructions is short and ranges between 16 to 40 bits. They're called vertical because they are normally listed vertically on a page. Vertical microinstructions effect single operations such as load, add, store, branch, etc. They often resemble machine language instructions containing one opcode and two operands.

Advantage:

Such instructions need much less bits compared to horizontal microinstructions i.e. they are more compact.

Disadvantages:

Execution of such microinstructions is more time consuming (i.e. delay is more) and needs complex hardware circuitry.

Diagram:

Figure 16 shows the vertical microinstruction format:

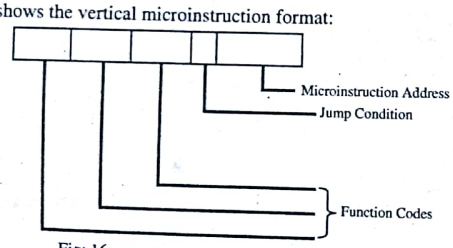


Fig: 16 Vertical Microinstruction

Related Questions & Answers

Question 1

What is meant by stored program organization?

Answer:

In a stored program organization, there is a memory that stores the programs (i.e. set of instructions) and data (i.e. operands), a simple processor register (say, an accumulator) and an instruction format with an opcode part and an operand (or address) part. The operand (data) is fetched from the memory and operated together (as per the opcode) with the data stored in the processor register.

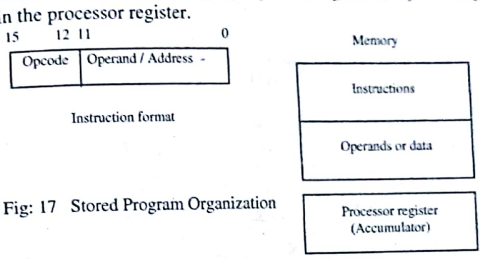


Fig: 17 Stored Program Organization

Question 2

Explain the Hardware implementation of logical micro-operation by showing one stage of logic circuit. **[WBUT 2003]**

OR,

Draw the block diagram and explain the functionality of micro-programmed control unit? **[WBUT 2004, 2010]**

Answer: Refer to section 5.7.

Question 3

Show with a flowchart, how an 'Add' instruction may be interpreted by CPU.

Answer:

The flowchart of the interpretation of 'ADD' instruction is shown below:

Terms used:

SC : Sequence Counter; PC: Program Counter; MAR: Memory Address Register; MDR: Memory Data Register; IR: Instruction Register; OPIR: Opcode part in IR;

ADDIR: Address-code part in IR; IOAR: I/O Address Register; IODR: I/O Data Register;

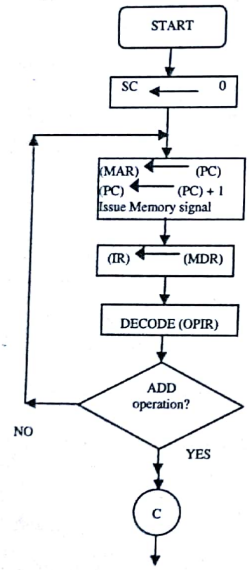
Assumptions:

(i) Each instruction has an opcode part and an address part. These are stored in the OPIR and ADDIR respectively.

(ii) The two most significant bits of the opcode jointly signify Memory, I/O, IMMEDIATE mode and Register address (i.e. 00: Immediate; 01: Register; 10: Memory; 11: I/O).

(iii) Irrespective of whether the ADDIND (the AUGEND and the SUM are assumed to be in the Accumulator) is in Memory, in I/O, is an immediate operand or in the CPU register, the Op-code (entire op-code minus the two most significant bits) of an ADD operation is unique.

Flowchart:



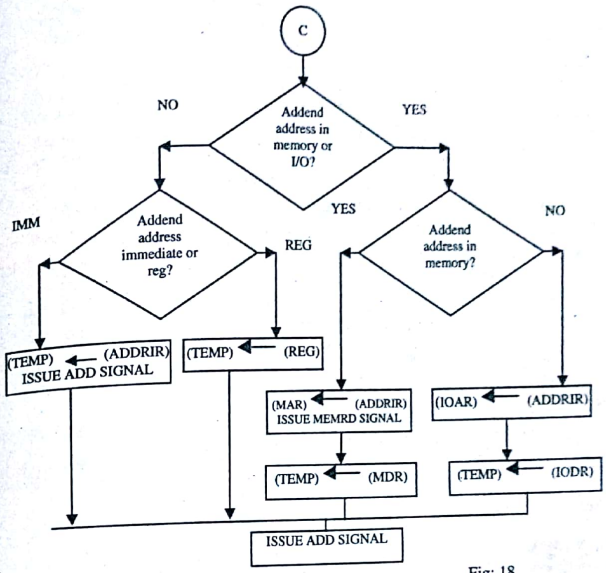


Fig: 18

Question 4

State the different phases of an instruction cycle. Explain with the help of a flowchart, how the control unit determines the type of instruction (Register, I/O, Direct or Indirect memory reference) to be executed after decoding. [WBUT 2003]

Answer: Refer to section 5.4.

Question 5

Explain the significance of Timing signals in a computer system. [WBUT 2003]

Answer: Refer to section 5.3.

Question 6

Explain the basic ideas of program execution.

Answer:

The basic ideas of program execution are as follows:

(a) A Computer executes users' programs.

(b) A user writes his program as a meaningful sequence of instructions. These instructions must belong to the Instruction set of the computer.

(c) The computer is capable of executing each and every instruction in its instruction set.

(d) A user's program is stored in the main memory.

(e) The starting address of the program is first loaded into the PC.

(f) The first instruction is first fetched from the main memory to the CPU. Then the instruction is executed in the CPU. This execution may require additional memory accesses (i.e. memory read or memory write operations) in order to fetch the operands from the memory if needed.

(g) After executing the first instruction, the CPU next fetches and executes the second instruction. This way, the CPU fetches and executes the remaining instructions in the program and thus completes the execution of the entire program.

Question 7

Define the following:
i) Micro-Program (ii) Micro-instruction (iii) Micro-operation. [WBUT 2003]

Answer: Refer to section 5.8.

Question 8

Explain the idea of execution and instruction sequencing with the help of a single-bus organization of a simple processor

Answer:

Execution and Instruction Sequencing:

(a) When an instruction residing at a certain memory address A is executed, the CPU usually executes the instruction residing at the next memory address A+1 (If the main memory is not word-organized but byte-organized, then the next address can be A+1 or A+2 or A+3, etc. depending on how many bytes wide an instruction is). This is called **"*linear sequencing*"**.

(b) Sometimes, the next instruction does not reside in at the next memory address. This situation owing to execution of instruction like unconditional jump, conditional jump, subroutine call, etc. or owing to arrival of external inputs like interrupts. This non-sequential execution of instruction is called **'*non-linear sequencing*'**.

(c) The operation of fetching and executing of each instruction is done by the CPU under the control of its control unit.

(d) The CPU has an ALU (to perform all arithmetic and logic operations), a number of special function registers (like PC, IR, ACC, MAR, MDR, SP, etc.), a number of general-purpose register (like R_0, R_1, R_2, etc.) and the control unit. Consider the figure 19, given below.

(e) All the above registers (including any dedicated register for the ALU) can exchange data between themselves using the CPU Internal Bus. Besides, any CPU can exchange data with any memory location (equivalent to a register) or any I/O register using the External Bus or the System Bus.

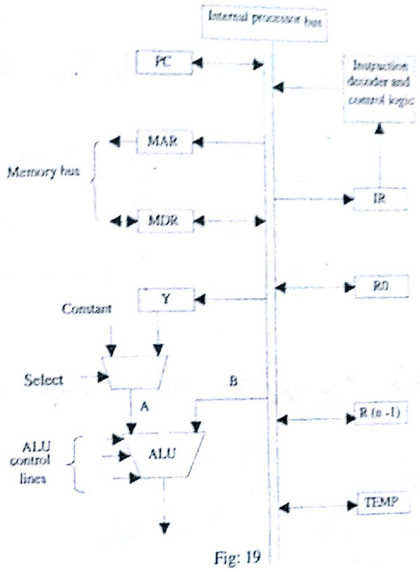


Fig: 19

Here all the registers and ALU are connected to a single, common, internal processor bus. Memory Data register (MDR) is bi-directional while Memory Address register (MAR) is unidirectional. The processor-registers R0 through R(n-1) and TEMP (and sometimes AC) are the general purpose processor registers while others are special purpose registers. The multiplexer MUX selects either the output of register Y or a constant value that is used to increment the contents of the program counter. The output of the MUX is provided as input A of the ALU while the input to B is obtained from the bus. The ALU has multiple control lines out of which one is enabled depending on the operation to be performed (as per the opcode of an instruction).

Question 9

Explain the Microinstruction format for the control memory & show how mapping is done from instruction code to Microinstruction format. [WBUT 2003]

Answer:

Refer to section 5.8 in chapter 5 for the first part of the question. Refer to questions 10 and 11 in the same chapter for the second part.

Question 10

What are the operations that can be performed to execute an instruction?

Answer:

Generally an instruction can be executed by performing one or more of the following operations in some specified sequence:

(a) A word or data is transferred from one processor register to another or to the ALU.
(b) An arithmetic or logic operation is performed and the result is stored in a processor register.
(c) Contents of a given memory location is fetched and stored into a processor register.
(d) A word is stored from a processor register into a given memory location.

Question 11

Explain how the content of one CPU register is transferred to another CPU register.

Answer:

While executing an instruction, the data from one register is transferred to another register. A sequence of steps is required for this transfer. For each register transfer operation, two control signals are needed. One to place the contents of the source register is the bus and the other to load the contents of the bus to the destination register.

Example:

Consider figure 20. Suppose the contents of register R0 are to be transferred to register Y. The steps are:

(a) Setting $R0_{out}$ to 1 enables the output of register R0. This places the R0 register contents on the processor bus.
(b) Setting Y_{in} to 1 enables the input of register Y. This loads the data from the processor bus into the register Y. The figure 20, below shows the above two steps. The 'in' and 'out' in any of the registers is controlled by two switches one at the 'input' end and the other at the 'output' end.

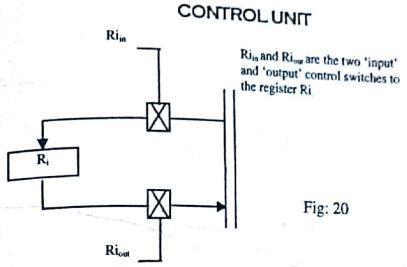


Fig: 20

Question 12

Assuming a simple single-bus organization for a CPU, explain the sequence of operations to add the contents of two registers R1 and R2 and store the sum in register R3.

Answer:

Consider the single-bus organization figure 19. The sequence of operations (control signals) to add the contents of register R1 and R2 and store the result in register R3 (i.e. the control signals for the microoperation R3 ← R1 + R2) are as follows:

(a) $\mathbf{R1_{out}}$, $\mathbf{Y_{in}}$ ⇒ The output of register R1 and input of register Y are enabled and thus the contents of register R1 are transferred to the bus and then loaded to the register Y.
(b) $\mathbf{R2_{out}}$, **Select Y, Add,** $\mathbf{Z_{in}}$ ⇒ The register Y is selected by enabling the Select signal of the multiplexer and thus the content of Y is transferred to the input A of the ALU. Simultaneously, the contents of register R2 are sent to the bus and then loaded to the input B of the ALU. As the function performed by the ALU depends on the respective signals applied to its control lines, in this case the ADD line is set to 1. This makes the ALU output line to hold the sum of the two numbers (at inputs A and B of the ALU). Then activating the input control signal of register Z, the sum is loaded into Z.
(c) $\mathbf{Z_{out}}$, $\mathbf{R3_{in}}$ ⇒ The contents of register Z are transferred to the destination register R3 via the bus.

Question 13

Give the timing diagrams of basic memory read and writing operations. [WBUT 2004, 2007]

OR,

Draw time diagram for memory read operation. [WBUT 2006]

OR,

Draw the time diagram for memory write operation. [WBUT 2008]

Answer: Refer to section 5.3.

Question 14

Explain how an encoder generates an individual control signal.

Answer:

Consider the generation of Z_{in} Control signal:

In the examples given before Z_{in} control signal is executed in the following clock states:

Example	Clock State
ADD (R3), R1	Z_{in} occurs in T6 (6^{th}) clock state.
Branch instruction	Z_{in} occurs in T_4 (4^{th}) clock state.
Both the ADD (R3), R1 and Branch instruction	Z_{in} occurs also in T_1 (1^{st}) clock state.

Diagram:

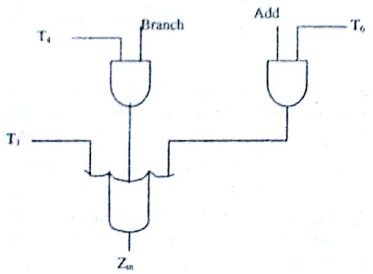


Fig: 21 Generation of the Z_{in} control signal for the single-bus processor organization

So, from the above figure 21, it can be seen that the Z_{in} control signal is generated during time T_1 slot for both ADD & branch instruction, T_4 slot for branch instruction, T_6 slot for ADD instruction and so on.

Content of IR	Output of Instruction decoder	Output of Step decoder	Output of external inputs.	Output of condition codes
ADD (R_3),R_1	'Add' line enabled	T_1 is enabled		
ADD (R_3),R_1	- DO -	T_6 is enabled		
ADD	- DO -	T_1 is enabled		'Unconditional' branch line is enabled
ADD	- DO -	T_4 is enable		

So, combination of all these inputs puts the encoder output "Z_{in}" control signal line to 1 (i.e. enables it). All other control signals are generated similarly.

Question 15

Explain how instructions are fetched and executed using microinstructions.

Answer:

Refer to section 5.4 "Detailed Explanation of the Instruction Cycle of any Processor".

Question 16

What does mapping of instruction mean?

Answer:

In each computer there is a set of instructions that its CPU is able to execute. This set of instructions is called the instruction set of the processor. Control signals are needed to be generated to fetch and execute each instruction in the instruction set. Executing a sequence of predefined microinstructions that remain stored in the control memory unit in turn generates the needed sequence of control signals for each instruction. A ***mapping*** is needed in the control unit to determine for each instruction, exactly from which location in the control memory, the corresponding sequence of microinstructions are located.

Hence, the ***mapping process*** is a rule that transforms the instruction code into a control memory address where the corresponding microroutine (for that instruction) is located.

Question 17

What is a control memory and control word? [WBUT 2004]

Answer: Refer to section 5.8.

Question 18

Explain the organization of control memory. How control unit is implemented using control memory? [WBUT 2004]

Answer: Refer to section 5.8.

Question 19

Explain the mapping from instruction code to microinstruction address.

Answer:

Suppose a mapping process to convert a 4-bit operation code (i.e. opcode) to a 7-bit control memory address is needed.

Diagram:

The figure 22, below shows the mapping process.

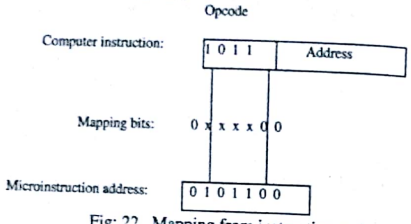


Fig: 22 Mapping from instruction code to microinstruction address

The 4-bits opcode can specify up to 16 distinct instructions. The control memory has, suppose, 128 locations (words). So 7-bit (as $2^7 = 128$) control memory address is required.

In the mapping, a 0 is placed in the most significant bit of the microinstruction address. The two least significant bits of the control address register is also cleared to 0 (this is because, as it is a 4-bit opcode and as $2^2 = 4$, hence 2 zeros must be added to the right of the control address register). So, this provides a microroutine of 4 microinstructions for each computer instruction. If more than four microinstructions are needed then addresses from 1000000 through 1111111 may be used. Also if less than 4 microinstructions are used, then the unused control memory locations will be available for other microroutines.

Question 20

What are the differences between Hardwired control unit and Microprogrammed control unit?

Answer:

The differences are as follows:

Sl. No.	Hardwired control unit	Sl. No.	Microprogrammed control unit
1.	Hardware implemented.	1.	Software (microinstructions) implemented.
2.	Non-programmable (done by logic design).	2.	Programmable.
3.	Faster.	3.	Slower.
4.	Not flexible.	4.	Flexible.
5.	Implementation is complex.	5.	Simple and easy to implement.

Question 21

Briefly explain a vertical microinstruction format and a horizontal microinstruction format. [WBUT 2004]

Answer: Refer to section 5.8.

Question 22

What are the advantages of microprogramming control over hardwired control? [WBUT 2004, 2005, 2007, 2008]

Answer:

Advantages:

As, once written, the microprograms can be changed, hence in case of microprogrammed control unit, the control memory design and implementation can be changed as needed and is very flexible.

But in case of hardwired unit to change, the entire hardware configurations are to be changed and hence such control units are not very flexible and changeable.

Disadvantages:

Hardwired control units (as hardware implemented) are faster than microprogrammed control unit (software implemented).

Question 23

What are the different status flags in a processor? Discuss overflow detection. [WBUT 2006]

Answer: Refer to section 5.1.

Question 24

What is instruction cycle? [WBUT 2006, 2007]

Answer: Refer to section 5.4.

Question 25

What are horizontal and vertical microprogrammings?

Answer:
In a microinstruction, each bit perform specific microoperation. Also the set of microinstructions (a microroutine) perform a group of operations to execute the respective instruction.

Horizontal microprogramming: In this, the micecroinstruction formats are much lengthy i.e. there is more number of bits in each microinstruction. So, more number of different microoperations can be performed at a time. Hence, here many resources can be controlled with a single microinstruction.

This approach is useful when a higher operating speed is needed and when the machine structure allows parallel use of resources. However, here more hardware is required to handle the execution of microinstructions.

Vertical microprogramming: In this length of each microinstruction is less but number of microinstrucions per instruction is more. So operations are performed sequentially (vertically) starting from the first microinstruction. Hence, here only a small number of control functions are specified in each microinstruction.

Though the vertical approach results in considerably slower operating speeds (as more microinstructions are needed to perform the desired control functions), here less hardware is required to handle the execution of microinstructions.

Question 26

Explain the difference between Hardwired control & Micro-Programmed control. [WBUT 2003, 2004, 2005]

Answer: Refer to question no 22.

Question 27

Write microprogram routines for following operations in RTL (register transfer logic) notation for (i) ADD (ii) STORE (iii) FETCH.

Answer:
(i) Microprogram routine for ADD:
Example:
ADD R1,(R3) ⇒ To add the content of address in register R3 to the content in register R1.

To execute this instruction, the following actions are to be performed:

a) To fetch this instruction from the main memory.
b) To fetch the first operand (i.e. content of the address in main memory).
c) To perform the addition in ALU
d) To load the result in R1.

The required microoperations and hence the required control signals to be executed for this are:

Step 1
Action:
PC_{out}, MAR_{in}, READ, Select4, Add, Z_{in} :

Comment:
The first address PC is sent to MAR; a READ pulse is then given to read the memory contents; SELECT4 is done to increment the PC i.e. suppose it's a 4-bit address machine. So to get the next address in PC, 4-bit is needed to add to the current PC content (suppose 3001 is the current PC content. So, here the next address will be 3001+4=3005; the next address to this will be 3009 and so on); Now the PC old content i.e. 3001 will get added to 4 in the ALU and will give 3005; the result will be stored in Z.

Step 2
Action:
Z_{out}, PC_{in}, WMFC:

Comment:
Content of Z is stored in the PC. So PC content now is 3009. WMFC ⇒ delay in 'READ'ing the memory content.

Step 3
Action:
MDR_{out}, IR_{in}

Comment:
Data fetched is stored in MDR and the MDR content is now stored in IR.

Step 4
Action:
$R3_{out}$, MAR_{in}, READ

Comment:
Address in R3 is to be fetched from the memory. So address in R3 is placed in MAR and READ pulse is given.

Step 5
Action:
$R1_{out}$, Y_{in}, WMFC

Comment:
Now content of R1 is placed in Y. WMFC ⇒ delay in reading the R1 value from the memory.

Step 6
Action:
MDR_{out}, Select Y, Add, Z_{in}

Comment:
Content in MDR i.e. the address-location content of R3, is placed in ALU via the bus (input B). Selecting Y ⇒ R1- content is placed in ALU via the MUX (input A). The two inputs are added in the ALU and the result is placed in Z.

Step 7

Action:

Z_{out}, R_{in}, End

Comment:
Content of Z i.e R1 + (R3) is placed in R1 via the bus. End ⇒ end of executing this micro program i.e ADD R1, (R3).

*[**Note:** Steps 1,2,3,…,7 gives the clock states i.e. in clock state 1, all microoperations (and all control signals), PC_{out}, MAR_{in}, READ, Select4, Add, Z_{in}, are executed. In clock state 2, all microoperations (and control signals), Z_{out}, PC_{in}, WMFC, are executed and so on.]*

Remember: In the bus, at a time data from only one register can be transferred.

(ii) Microprogram routine for FETCH:

Example:

MOV R2, (R1) ⇒ To fetch the content of the memory location whose address is the content of the R1 register and load it in register R2.

To execute this instruction, the following actions are to be performed:

a) To move the contents of register R1 to MAR [i.e. MAR ← [R1]].
b) To start a Read operation on the memory bus.
c) To wait for the MFC response from the memory.
d) To load the MDR from the memory bus.
e) To load the MDR content in the R2 register [i.e. R2 ← [MDR]].

The required microoperations and hence the required control signals to be executed for this are:

Step	*Action*
1	$R1_{out}$, MAR_{in}, Read
2	MDR_{inE}, WMFC
3	MDR_{out}, $R2_{in}$

(iii) Microprogram routine for STORE:

Example:

MOV (R1), R2 ⇒ To load the contents of R2 register in the desired memory location, the address of which is the content of R1 register.

To execute this instruction, the following actions are to be performed:

(a) To move the contents of register R1 (i.e. the desired address) to MAR [i.e. MAR ← [R1]].
(b) The data in register R2 is moved to MDR and a Write signal is issued.

(c) Data from MDR is then stored in the desired memory location.
(d) To wait for the MFC response from the memory.

The required microoperations and hence the required control signals to be executed for this are:

Step	*Action*
1.	$R1_{out}$, MAR_{in}
2.	$R2_{out}$, MDR_{in}, Write
3.	MDR_{outE}, WMFC

Question 28

Give two very commonly used applications of microprogramming.

Answer:
Two very commonly used applications of microprogramming are 'emulation' and 'operating-system support'. In emulation, microprograms on one machine can be used to execute programs originally written to run on a different machine. This can be done very simply by modifying the microcode of a machine, so that it can run or execute any software from another machine.

The operating-system support technique simplifies and improves the implementation, functions and performance of the operating systems.

Question 29

Write a symbolic micro program of a basic FETCH routine. **[WBUT 2004]**

Answer: Refer to question no 27.

Question 30

What do you mean by packed microprograms and unpacked microprograms?

Answer:
Just like the two terminologies discussed in the previous question, the terms 'packed' and 'unpacked' microprograms also signifies different design characteristics for a microprogram.

The degree of packing signifies the extent to which a given control task is correlated to the specific or required microinstruction bits. In case of packed microprograms, more 'packed' the bits are, a given number of microinstruction bits can store or contain more number of information. Hence, for packed microprograms, limited number of bits can

store more information. Also, it can be said that packed microprograms are better encoded.

On the other hand, the situation is just the opposite in case of unpacked microprograms.

Question 31

Draw a block diagram showing the Instruction Cycle, which also handles an Interrupt.

Answer:

Figure 23 shows the block diagram of the Interrupt and the Instruction Cycle. As can be seen in the figure, the entire cycle constitutes of three different sub-cycles: the Instruction Fetch Cycle, the Instruction Execution Cycle and the Interrupt Cycle or the Interrupt Service Cycle. With the explanation of the diagram being the same as in section 6.4 above, the only notable is the occurrence and suitable handling of an interrupt in the midst of an Instruction Cycle. Whenever an interrupt occurs in the midst of an instruction execution, the CPU on noticing the interrupt, stops execution of the current instruction and jumps to the interrupt service routine. On servicing (i.e. handling) the requested interrupt, the control again come back and resumes from the previous point where stopped. So, CPU continuously checking for interrupts and if needed (i.e. if there is an interrupt) processing the interrupt, are two important phases of the Interrupt Cycle. As soon as an interrupt is processed, the interrupt lines are enabled for further interrupt requests.

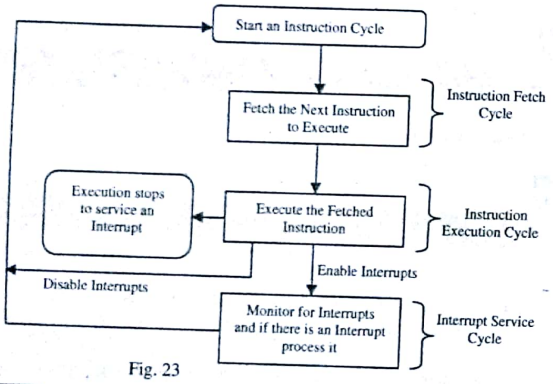


Fig. 23

Question 32

How can you characterize control unit?

Answer:

The characterization of the control unit can be readily done by defining the functions to be performed or the functional requirements for the control unit. These are:

a. The basic elements of the central processing unit should be defined clearly.
b. The different micro-operations performed by the central processing unit should be described clearly.
c. In order to make the micro-operations work properly, the different relevant functions needed to be performed by the control unit should be clearly defined and described.

Question 33

What are the general design approaches for a hardwired control unit?

Answer:

There are three general design approaches for a hardwired control unit. These are:

a. ***Traditional 'state-table' method***, which, though, can produce the minimum component design as required but is a complicated design process that is very hard to modify.
b. ***Clocked-delay element*** is a straightforward and simple layout, based on the required instruction implementation flowchart. However, it requires more number of delay elements or flip-flops than are really needed.
c. ***Sequence counter approach*** is a standard counter-decoder approach using which polyphase-clock signals can be easily derived using the master clock. In a circuit, these derived clock signals can be applied to the combinational portions.

Question 34

Explain the various shift microoperations.

Answer:

Shift microoperations are generally used for serial data transfer operations. Three types of shift microoperations are there, namely: logical, circular and arithmetic.

- ***Logical Shift:*** This type of microoperation transfers 0 through the serial input.
- ***Circular Shift***: This type of microoperation rotates or circulates the bits around the two ends taking care that no information is lost.
- ***Arithmetic Shift***: This type of microoperation shifts a signed binary number either to the left or to the right.

Table below shows the shift microoperations:

Symbolic designations	Descriptions
R1 ← shl R	Shift-left register R by one position
R1 ← shr R	Shift-right register R by one position
R ← cil R	Circular shift-left register R (i.e. rotates the

Symbolic designations	Descriptions
	bits)
R ← cir R	Circular shift-right register R
R ← ashl R	Arithmetic shift-left R (i.e. it shifts a signed binary number to the left)
R ← ashr R	Arithmetic shift-right R (i.e. it shifts a signed binary number to the right)

Question 35

Explain the stack organization of a computer system and write all the sequence of micro-operations for push and pop operations.

Answer:

A stack is actually a storage device that stores information. In a stack, information are stored in such a manner that the last item stored in a stack is the first one to be retrieved from the stack. In a stack, only one information can be accessed at a time. A set of registers constitutes a stack. A stack is called a LIFO or last-in, first-out list (also known as a pushdown list), as the last item stored in the stack is the first item to be retrieved. Stack can be organized with multiple contiguous memory registers and it is then known as memory stack or can be organized using CPU registers, then known as register stack.

Stack Pointer (SP) is a register whose value always points at the top item in stack i.e. SP always holds the address for the stack. The two basic operations that can be performed on a stack are the insertion and deletion of items. In the insertion operation, which is also known as push, information is stored in the top of stack (TOS) position in the stack and in the deletion operation, known as pop, information from the TOS is retrieved.

Figure 24 shows a 64-word stack organization. The stack grows upward from location 0 to location 63, which is the final TOS. On addition of every new element, the TOS value is incremented by one until the stack is full, marked by FULL ← 1 (FULL is a flip-flop which is 1 when the stack is full and EMTY is a flip-flop which is 1 when the stack is empty). The first element is pushed in the stack at the SP ← 1 location i.e. the first location. So when SP ← 0, it means that the stack is full and hence FULL is 1. So EMTY is 0 i.e. stack not empty.

The steps for the PUSH operation are as follows:

SP ← SP + 1 ⇒ Stack pointer is incremented.
M[SP] ← DR ⇒ Item, from data register, is stored on the TOS.
If (SP = 0) then (FULL ← 1) ⇒ To check if stack is full.
EMTY ← 0 ⇒ Means that the stack is not empty.

While popping elements from the stack, if SP ← 0, it means that the first item pushed at the initial position is popped out. Hence the stack is now empty and so EMTY ← 1. So, FULL is 0.

The steps of POP are as follows:

DR ← M[SP] ⇒ Item from TOS is popped and stored in the data register.
SP ← SP - 1 ⇒ Stack pointer is decremented.
If (SP = 0) then (EMTY ← 1) ⇒ To check if stack is empty.
FULL ← 0 ⇒ Means that the stack is not full.

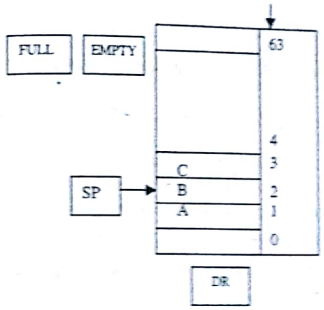


Fig. 24

Question 36

Write short note on Microinstruction. [WBUT 2006]

Answer: Refer to section 5.8.

Question 37

What do you mean by instruction cycle, machine cycles and T states? [WBUT 2008]

Answer: Refer to section 5.2.

Short Questions & Answers

1. Based on the design characteristics, classify microinstructions.
Answer:
Microinstructions can be classified in the following ways based on the different design characteristics:

a. Vertical and Horizontal
b. packed and Unpacked
c. Hard and Soft
d. Direct and Indirect encoding

2. What do you mean by hard microprograms and soft microprograms?
Answer:
The terms hard and soft microprograms emphasize different design characteristics for a microprogram.
Microprograms that are generally fixed or un-modifiable are entitled as *hard microprograms*. Such microprograms are more committed to read-only memories.
On the other hand, *soft microprograms* are highly modifiable and are meant for user microprogramming.

3. What can be the applications of hardwired and microprogrammed control unit?
Answer:
Hardwired control units are used in the RISC (Reduced Instruction Set Computer) type computers and Microprogrammed control units are used in CISC (Complex Instruction Set Computer)type computers.

4. What is a firmware?
Answer:
A meaningful sequence of microinstructions is known as a microprogram or firmware, which means that microprogram lies in midway between hardware and software.

5. Give advantages of microporgramming.
Answer:

1. Easy to modify the way instructions works, for example, in order to fix up a problem in the programming.
2. Easy to add new instructions.

6. Give disadvantages of microporgramming.
Answer:

1. High-level machine instructions are too slow or complex.
2. Too complex logic circuitry.
3. Microprogramming is slower than pipelining concept.

Multiple Choice Type Questions

1. In a Micro-processor, the address of the next instruction to be executed, is stored in *[WBUT 2003]*

(i) Stack pointer *(ii) Address hatch*
(iii) Program counter *(iv) General purpose register*

Answer: (iii)

2. In microprogramming *[WBUT 2006]*

a) horizontal microinstruction is faster
b) vertical microinstruction is faster
c) hardware control unit is faster
d) none of these

Answer: (c)

3. Which logic gate has the highest speed? *[WBUT 2006]*

a) DTL *b) RTL* *c) ECL* *d) TTL*

Answer: (c)

4. A UART is an example of *[WBUT 2010]*

a) serial asynchronous data transmission ship
b) PIO
c) DMA controller
d) none of these

Answer: (a)

5. Control program memory can be reduced by *[WBUT 2010]*

a) Horizontal format *b) Vertical format micro-program*
c) Hardwired control unit *d) None of these*

Answer: (b)