Instruction formats

An instruction consists of an OP-code and one or more operands. These operands could be explicitly specified in an instruction or they could be assumed. An instruction format is used to define the layout of bits allocated to these elements of instructions. In addition, the instruction format explicitly or implicitly indicates the addressing modes used for each operand in that instruction.

Designing of instruction format is a complex art. A computer uses a variety of instructions. Some of the issues effecting instruction are given below:

Instruction length: It is the most basic issue of format design. A longer instruction will mean more time in fetching an instruction.

For eg, an instruction of length 32 bits on a machine with word size of 16 bits will need two memory fetches to bring the instruction.

On the contrary, a smaller instruction will mean less number of operations, fewer addressing modes. Thus the basic trade off is between smaller instruction Cless space) Vs desire for more powerful and more number of instructions.

Normally a programmer desires:

- More OP-code and operands in a instruction as it will result in smaller programs.
- More addressing modes for greater flexibility in accessing various types of data.

However a 32 bit instruction although will occupy double the space and can be fetched at half the rate of 16 bit instruction, but can not be doubly useful.

Factors to be considered for deciding instruction length

Memory size: If a larger memory range is to be addressed then more bits may be required in address field. A memory of size 1K bytes will need 10 bits of address ($1K = 1024 \text{ bytes} = 2^{10}$) whereas a memory size of 1M byte will need 20 bits of address ($1M = 2^{20}$ bytes).

Memory Organisation: If the system supports virtual memory then memory range which is to be addressed by the instruction is larger than physical memory.

Memory transfer length: Instruction length should be equal to data bus length or multiple of it.

Memory transfer: The data transfer rate from the memory ideally should be equivalent to the processor speed. It can become a problem if the processor executes instructions faster than the rate of fetching the instructions. One solution for such problem is to keep instruction short.

Allocation of bits for different fields in an instructio

For fixed length Instruction, we can have either more number of opcodes or better addressing capabilities. In case we want to have both the features then we can go for variable length instructions.

Factors which are considered for selection of addressing bits are

- **Number of addressing modes:** The more are the addressing modes the more bits will be needed to select an addressing mode.

Number of operands: Fewer no of operands in an instruction will require less bits. Now a days many of machines are having two operand references in an instruction. Each of these operands may need an addressing mode indicator bits.

Register addressing versus memory addresses: If more and more register can be used for operand references then instructions are bound to be smaller as number of number of registers is far less than the memory size, therefore they require only a few bits in comparison to the bits needed for the memory addresses.

Register sets: Registers are farther divided into specialized groups. These groups could be general purpose register, data register, address registers. and segment registers. This results in farther decrease in the size of the instruction bits for register addressing. For eg in case a machine has 16 general purpose registers then a register address will require 4 bits, however if 16 registers are divided into two groups, then one of the 8 registers of a group will need 3 bits for register addressing

Range of address: The range of main memory addresses which need to be addressed directly or indirectly require a specific number of bits in instructions.

For eg if direct addressing is used then memory size determines the no of addressing bit. However in displacement or index addressing scheme, it is the offset which can control the number of addressing bits in the instruction.

Granularity of address: As far as memory references are concerned, granularity implies whether an address is referencing a byte or a word at a time. This is more relevant for machines which have 16 bit, 32 bit and higher bit words. Byte addressing although may be better for character manipulation, however requires more bits in an address. For eg memory of 16K words (1 word = 16 bits) is to be addressed directly then it requires.

Word addressing = 16K words
$= 2^{14}$ words
= 14 bits are required for word addressing

Byte addressing = 2^{14} words
$= 2^{15}$ bytes
= 15 bits are required for byte addressing.

Variable length of instructions:

With the better understanding of computer instruction sets, the designer came up with the idea of having a variety of instruction formats of different lengths. The advantages of such scheme are

- Large number of operations can be provided which have different length of instructions.
- Flexibility in addressing scheme can be provided efficiently and compactly.

However, the basic disadvantage of such scheme is a complex CPU.

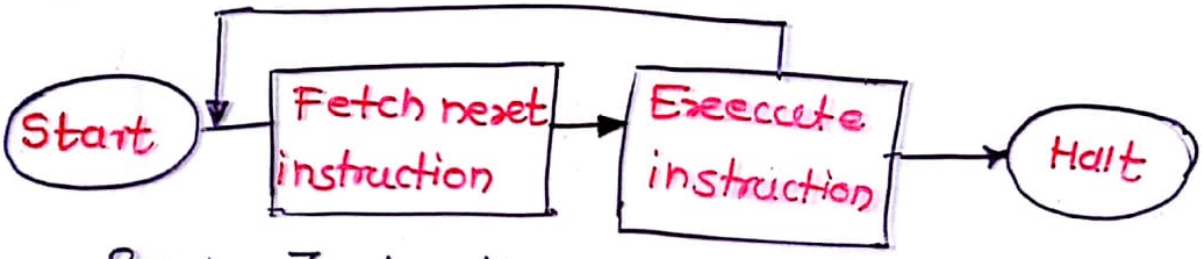
Instruction Cycle

The basic function of a computer is execution of a program. A program consists of a set of instructions. To process an instruction

① The processor reads (fetches) an instruction from memory.

② The processor executes the instruction

③ The instruction execution may involve several operations.



Basic Instruction cycle

Execution cycle for a particular operation is dependent on the type of operation. Moreover, a particular instruction may involve more than one reference to memory. Also instead of memory reference, an instruction may specify an I/O operation

for example, a simple instruction

Add x, BX ; $x \leftarrow x + BX$

Require following steps for execution:

- fetch the 'ADD' instruction
- Read the contents of memory location x into the processor
- Add contents of x and register BX.
- Write the result from the processor to memory location x.

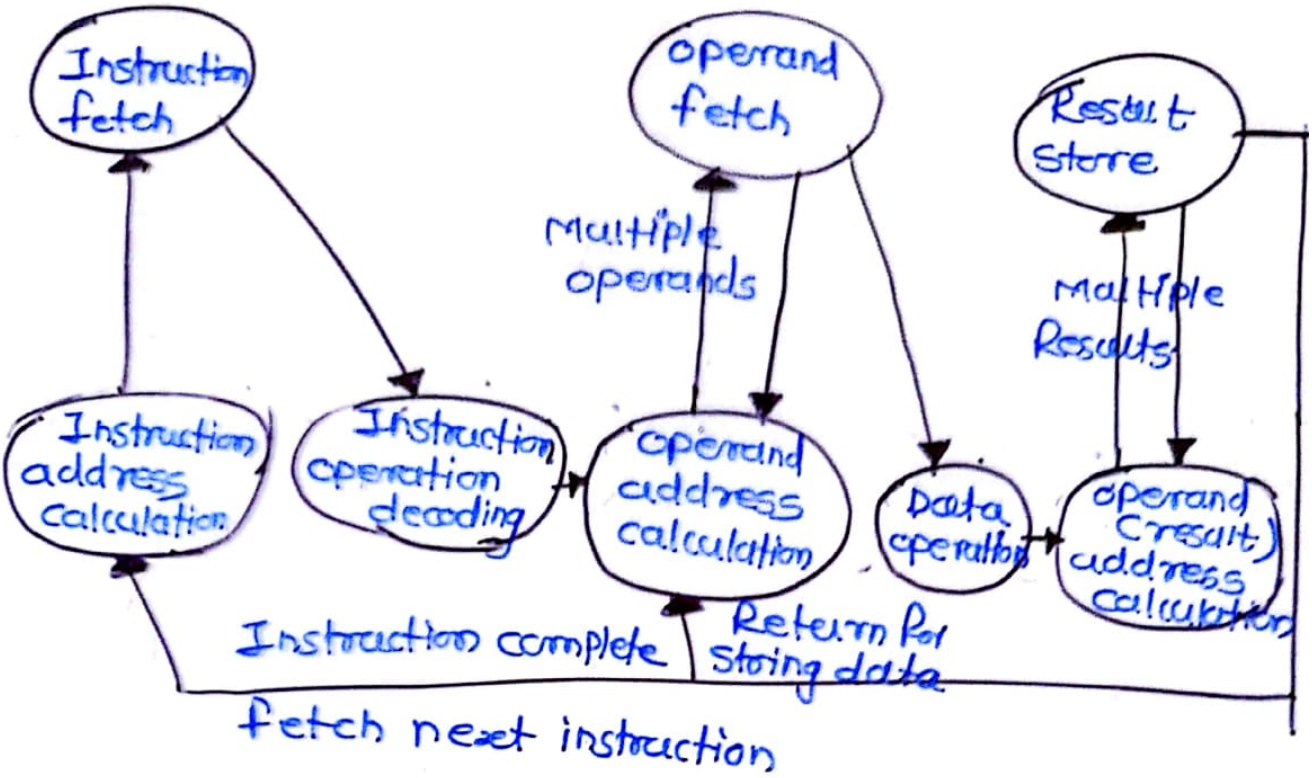


Fig Instruction cycle state diagram

For any given instruction, states can be described as follows:

- Instruction address calculation: Determine the address of the next instruction to be executed. For eg in 8086 the address of the next instruction is given by

  CS*16+IP

  where CS is the code segment register and IP is instruction pointer (program counter)
- Instruction fetch: Read instruction from memory into the processor.
- Instruction operation decoding: Analyze instruction to determine type of operation to be performed and operands to be used.
- Operand address calculation: If the operation involves reference to an operand in memory they determine the address of the operand.

Operand fetch: Fetch the operand from memory.
Data operation: Perform the operation indicated in the Instruction.
Result store: Write the result into the memory.

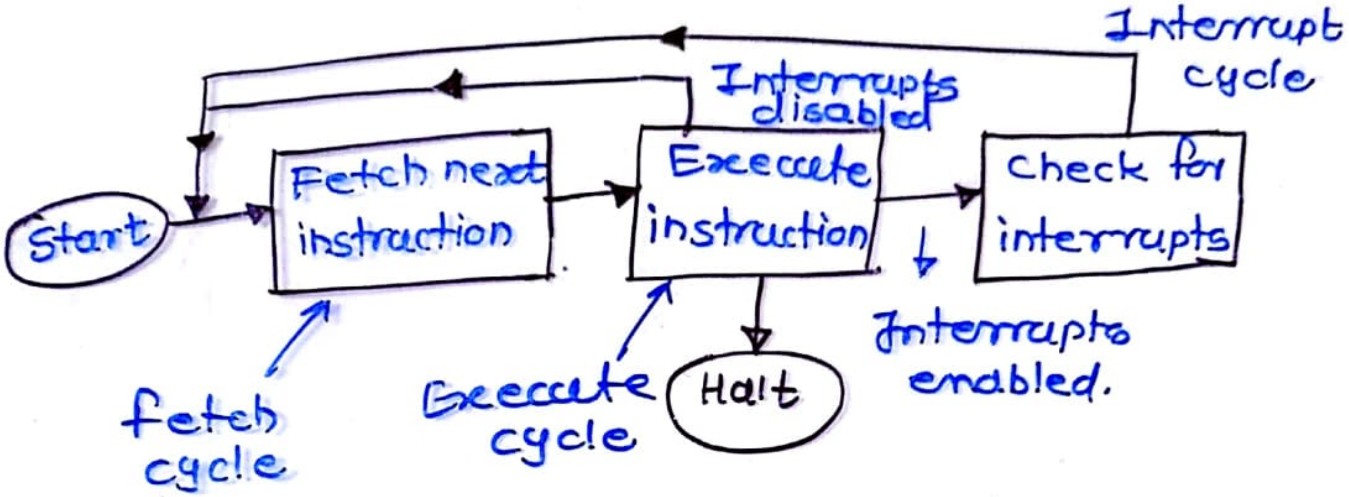
states in the upper part of the figure involves exchange of data between the processor and the memory.
states in the lower part of the diagram involve only internal processor operations.
The diagram allows for multiple operands and multiple results.

Interrupts and the instruction cycle:

Virtually all computers provide a mechanism by which I/O devices may interrupt the normal processing of a program. Whenever the processor is interrupted, the processor suspends execution of the current program, transfers control to a program to service the particular interrupt and resumes the original execution after the interrupt is serviced.

To accommodate interrupts, an interrupt cycle is added to the instruction cycle. In the interrupt cycle, the processor checks to see if any interrupts have occured. If no interrupts are pending, the processor proceeds to the fetch cycle and fetches the next instruction of the current program. If an interrupt is pending, the processor does the following:

- It suspends execution of the current program being executed and saves its context (content of program counter and other relevant registers)
- It sets the program counter to the starting address of an interrupt handler routine



Instruction cycle with Interrupts

Machine Instructions: Every machine has a set of instructions that machine can execute. A machine instruction has a number of elements

i) An operation code known as OPCODE which specifies the operation to be performed.
ii) Referenees to operands on which the operation is to be performed.
iii) A referenee to the operand which will store the result produced by the instruction
iv) A referenee to the next instruction to be executed.

A sample Instruction format

Opcode	Reference to operand	Reference to operand
4 bit	6 bits	6 bits

Types of Instructions

1. Data transfer instructions
2. Arithmetic and logic operations
3. control transfer instructions
4. Miscellaneous instructions.

(1) **Data Transfer instructions:** Since the arithmetic and logic operations are normally performed on the data stored in CPU registers; we need instructions to bring data to and from memory to registers. Most of the time, we identify a memory location by name. for eg x, y, i, j, loc

Register names may be R_0, R_1, R_2, Ac, Bx, Cx etc
The contents of a memory location or CPU register are denoted as given below:

$R_1 \leftarrow x$ means, the contents of memory location x are transferred into register R_1

$R_1 \leftarrow R_2$ means, the contents of the register R_2 are transferred to R_1

$x \leftarrow R_1$ means, the contents of register R_1 are transferred into memory location x.

(2) **Arithmetic and logic operations:**

$R_1 \leftarrow R_2 + R_1$ means, contents of CPU registers R_1 and R_2 are added and the result is stored in register R_1

$x \leftarrow R_1 + y$ means, contents of memory location and register R_1 are added and the result is stored in memory location x

(3) Control transfer instructions: In most cases, the next instruction to be executed immediately follows the current instruction. High level language constructs like:

if – then – else statement
while loop,
for loop.

Are implemented using conditional and unconditional branching instructions. Conditional branching instructions are implemented with the help of status word. Use of conditional and unconditional branch instructions are explained with the help of the small program segment given below.

Program for addition of first hundred natural numbers:

- Sum is stored in register R_1.
- Register R_2 is used for looping. R_2 is varied from 100 down to 1.
- While loop is implemented with the help of conditional and unconditional branching instruction

```
1        R1 ← 0 ; R1 is used for storing of sum
2        R2 ← 100 ; contents of R2 will be varied
                    from 100 down to 1
3  Loop: JUMP to loop1 if R2 is equal to zero
4        R1 ← R1+R2
5        R2 ← R2-1
6        JUMP to Loop
7  Loop1: Outside the loop
```

unconditional branching

Conditional branching

Implementation of conditional branching instructions

Processor keeps track of some information regarding the result of various arithmetic and logic operations. This result is used by subsequent conditional branching instructions. These information are stored in a special register known as status(flag) register. Every bit of the status register stands for a condition. Commonly used conditions (flags) are:

N (Negative) - Set to 1 if the result is negative

Z (Zero) - Set to 1 if the result is zero

O (Overflow) - Set to 1 if the arithmetic overflow has occured

C (Carry) - Set to 1 if carry out or borrow in the results from an operation.

Miscellaneous instructions: These instructions do not fit in any of the above categories. Some of these instructions are

1. Supervisory calls
2. Return from an interrupt
3. Halt instruction.
4. Privileged instructions of operating system

Operand Data Types

1. Address
2. Number
3. character
4. Logical data.

① **Address:** Actual data may be stored in memory, machine instruction may contain a reference to data. Thus addresses are operand references

② **Numbers:** All machines provide numeric data types. One special feature of numbers used in computers is that they are limited in magnitude and underflow or overflow may occur during arithmetic operations on these numbers. The maximum and minimum value is fixed for an integer number.

Three types of numerical data are common in computers:

- Integer or fixed point
- floating point
- Decimal

③ **Characters:** Another very common data type is a character. characters are represented in ASCII. It has 7 bits for coding data pattern which implies 128 different characters. The eighth bit of ASCII can be used as a parity bit.

④ **Logical Data:** In general a data word or a byte is treated as a single unit of data. Each bit of a n bit data can be considered to be a logical data.

Number of Addresses in an Instruction

Length of a machine instruction can be reduced by having less number of operands or addresses in an instruction. Most of the instructions require three operands. In instructions having less than three addresses, normally some of the operand locations are implicitly defined. Following table gives an example of zero, one, two and three address instructions and their interpretation.

Number of addresses	Instruction	Interpretation
3	Add X, Y, Z	Operation, X ← Y+Z is executed
2	Add X, Y	operation X ← X+Y is executed. [one address is implied as X]
1	Add X	operation, AC ← AC+X is executed [X is added to accumulator. Two operands are implied]
0	Add	[In such instruction, all operands are implied. Two operands are poped from stack and the final result is pushed back on stack]

Zero Address or Stack Based Instructions

Such instruction takes data from top of the stack. ALU references a stack which can be implemented in main memory. POP and PUSH operations can be used to store a value from stack to memory or to load a value from memory to stack respectively

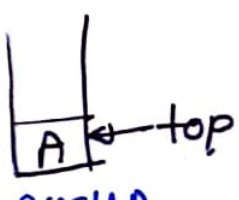
Example

To evaluate an expression using stack based instructions, expression is first converted to postfix expression.

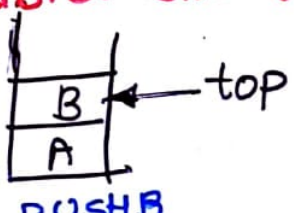
An expression A+B*C can be represented as ABC*+ in postfix notation.

Above expression can be evaluated using the following instructions.

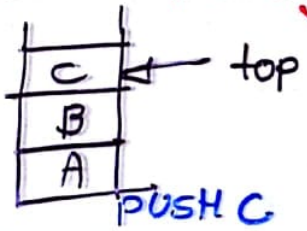
```
PUSH A   /* Transfer A on top of the stack */
PUSH B   /* Transfer B on top of the stack */
PUSH C   /* Transfer C on top of the stack */
MULT     /* Multiply top two operands from the stack
            and store the result on top of the stack */
ADD      /* Add top two operands from the stack and
            store the result on top of the stack */
POPD     /* Transfer the result to memory location D */
```



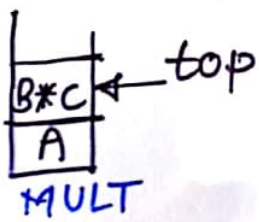
PUSH A



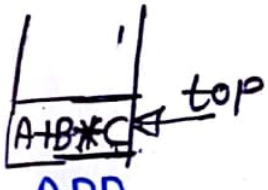
PUSH B



PUSH C



MULT

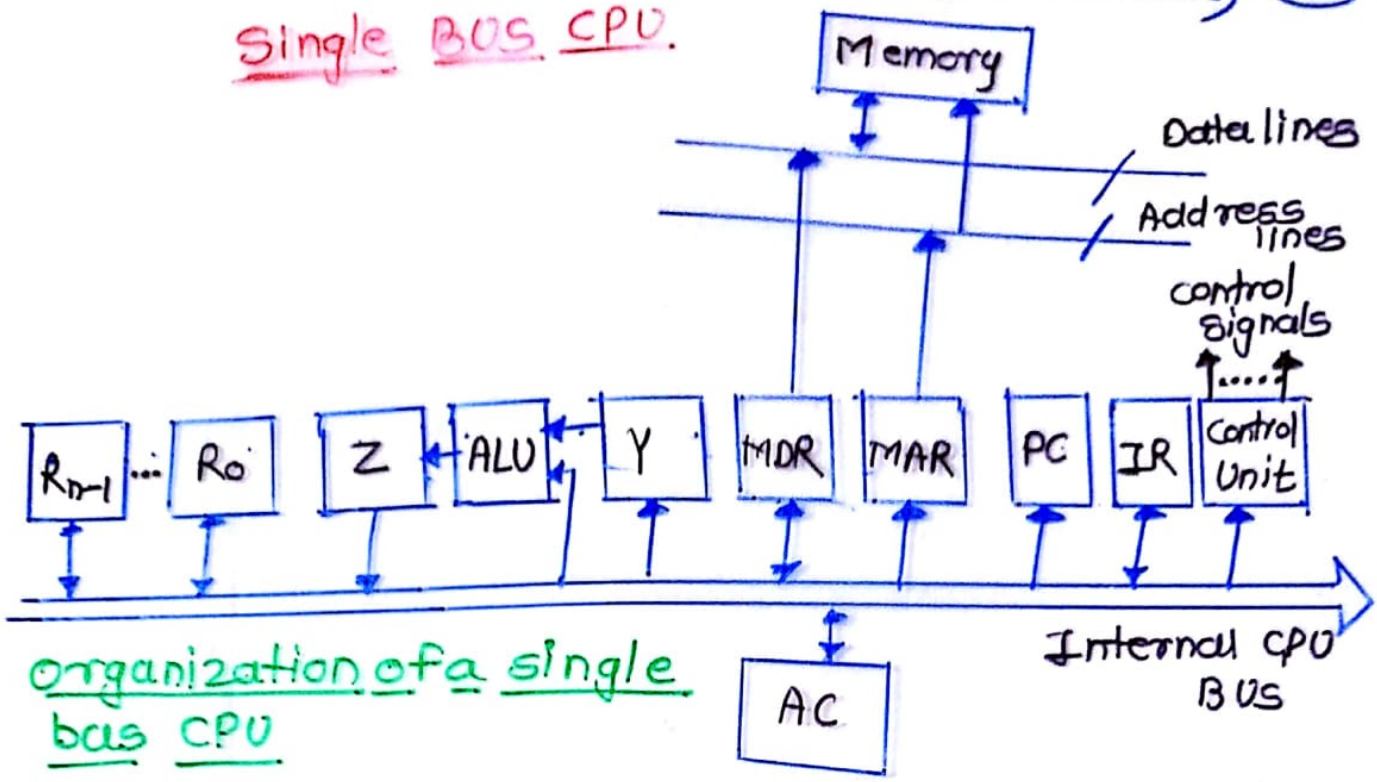


ADD


POP D

Each bit can have a value 1 (true) or 0 (false)

Single BUS CPU.



organization of a single bus CPU

Components of a CPU could be interconnected in various ways. In a single bus organization, all components of a CPU are connected to a single internal bus. This makes the interconnection structure relatively simple. The external bus, the bus connecting the CPU to the memory and I/O devices is connected to the CPU via memory Data Register (MDR) and the memory address register (MAR)

The advantages of using internal bus arrangements are:

- Simple data path interconnections which means easy layout for control
- Saving of CPU space as inter-register connection space is minimized.

Two temporary registers Y and Z are added to the ALU, otherwise the output of ALU will go back to its input, which is undesirable. With this arrangement, an operation to add a value from memory to the AC would have the following steps:

MAR ← IR(address)
MDR ← memory, read operation
Y ← MDR
Z ← AC+Y
AC ← Z

MICRO-OPERATION

Main task of a computer is instruction execution, therefore, the details on how an instruction can be executed will help in better understanding of functionality of ALU and control unit.

A micro-operation is a primitive action performed by a machine on the data stored in the registers.

A machine instruction is implemented by executing a set of micro-operations in a predefined sequence:

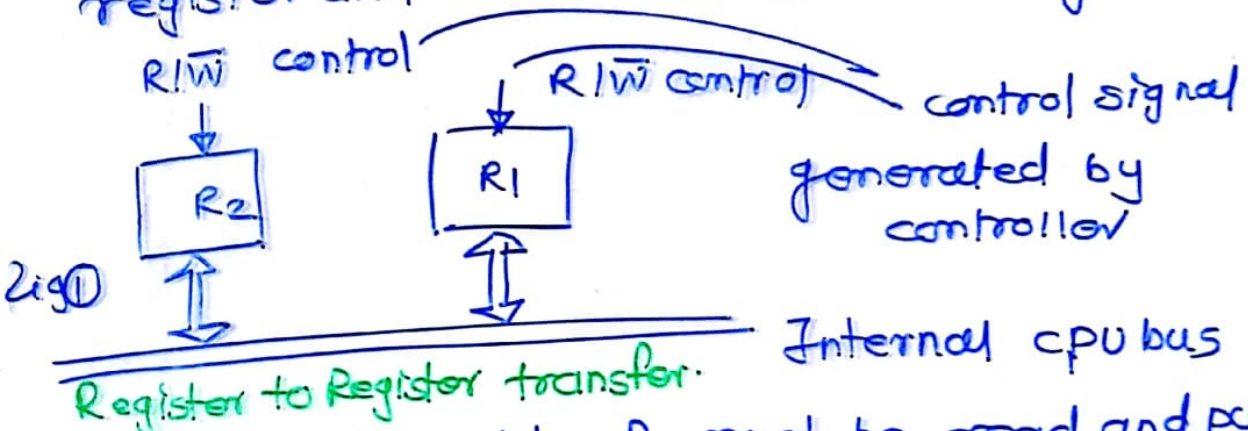
1. Register transfer micro-operations.
2. Arithmetic micro-operation
3. Logic micro-operations
4. Shift micro-operations.

Register Transfer micro operation

This micro-operation transfers information from one register to another register.

$R_{destination} \leftarrow R_{source}$

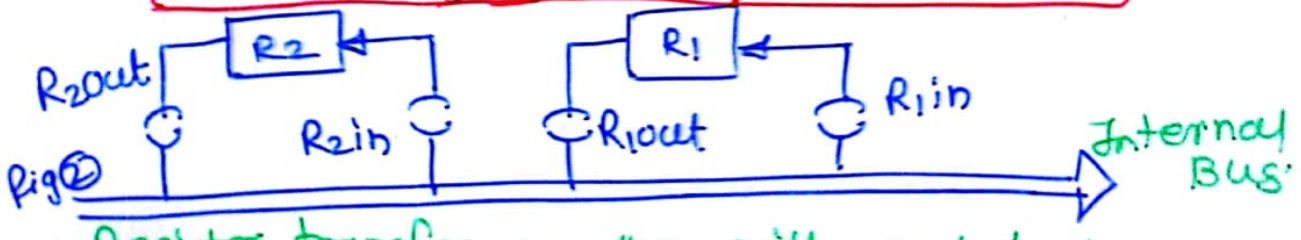
or $R_2 \leftarrow R_1$ where R_1 is the source register and R_2 is the destination register.



Register to Register transfer.

- Contents of register R_1 must be read and put on the internal CPU bus.
  When R/W control signal is 1 read operation will be performed on R_1
- Data moving on internal CPU bus can be stored in register R_1 by making the R/W control signal of register R_1 as 0

R/W of R_1	R/W of R_2	operation
1	0	$R_2 \leftarrow R_1$
0	1	$R_1 \leftarrow R_2$
1	1	Invalid
0	0	Invalid



Register transfer operation with control signals.

Fig ① is redrawn as Fig ② for easy understanding.

control signal $R_2in \rightarrow R1\overline{W}=0$

control signal $R_1out \rightarrow R1\overline{W}=1$

control signals	operation (Data Transfer)
R_1in	Internal bus to Register R_1
R_1out	Register R_1 to internal Bus
R_2in	Internal Bus to Register R_2
R_2out	Register R_2 to internal bus

Thus the operation

① $R_2 \leftarrow R_1$ can be performed using set of control signals R_1out and R_2in. ie

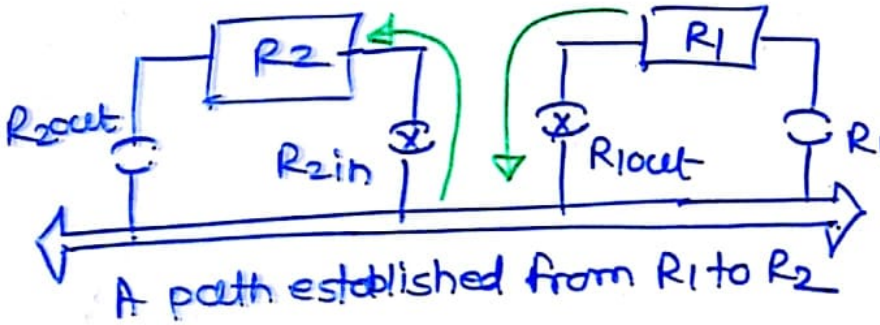


Fig ③

② for $R_1 \leftarrow R_2$ performed using R_2out & R_1in

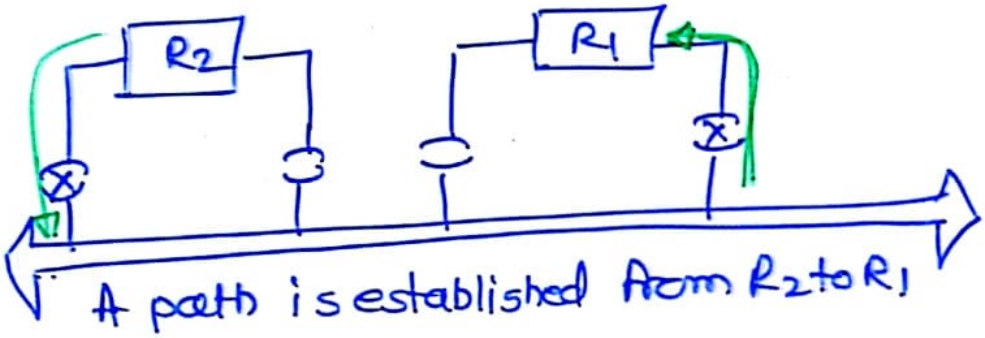


Fig ④

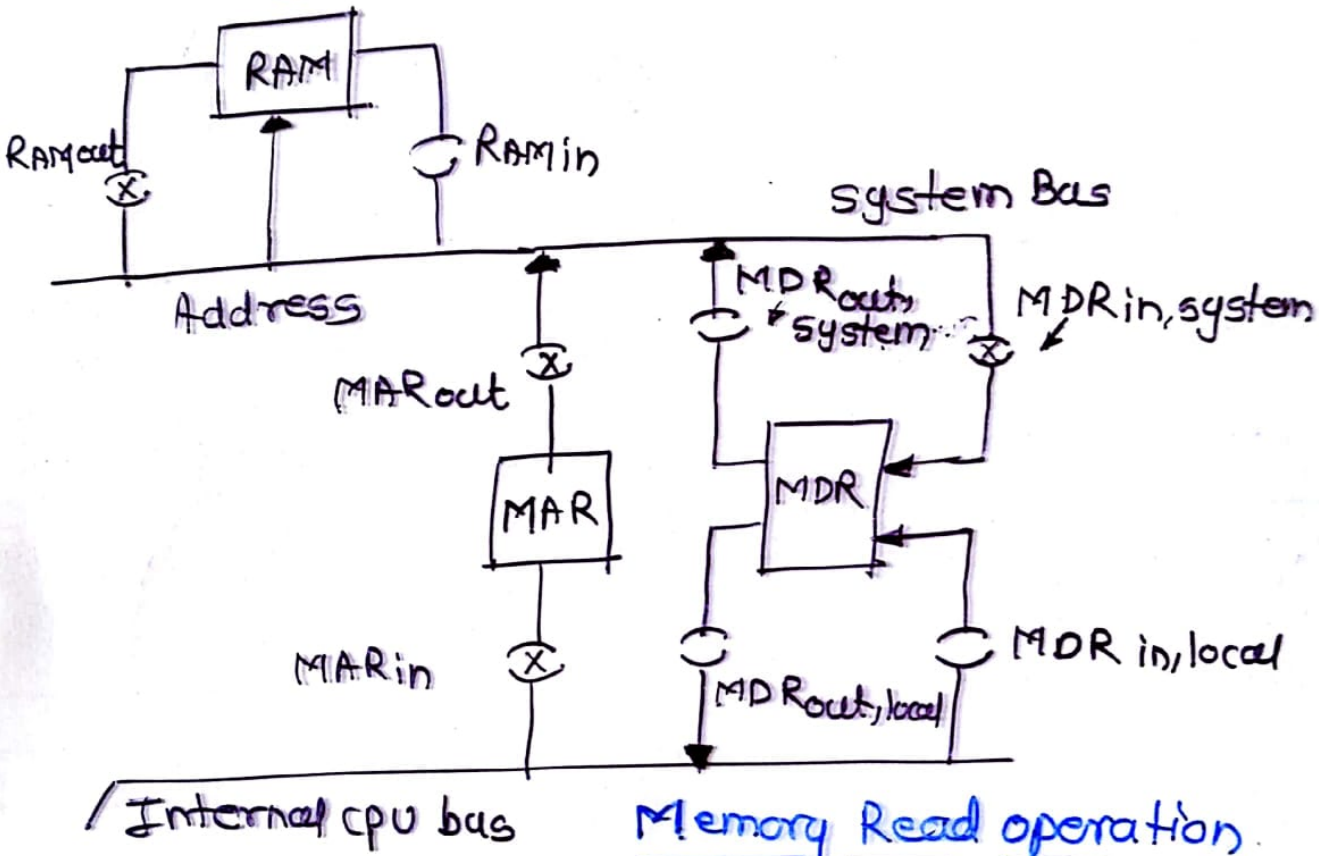
- In Fig ② control signals are represented as switches
- fig ③ shows conceptual representation of Fig ②
- when controller sends the control signal R_1out and R_2in, two switches are operated This forms path from R_1 to R_2 via Internal CPU Bus

Memory transfer micro-operation

Memory transfer is achieved via the system bus, since the main memory is a random access memory, therefore, address of the location which is to be used is supplied. The address is supplied by the CPU to the system bus through MAR (memory address Register)

There are two memory transfer operations:

1) Memory Read
2) Memory Write.



Memory Read operation

Memory Read :- Put the memory address in Memory Address Register (MAR) This can be done by generating the control signal MARin

- Read the data of memory: This operation is achieved by putting the MAR data on address bus along with a memory read control signal

on control bus (control bus is part of the system bus).

- The result of memory read operation is put on data bus (part of system bus) which in turn stores the data in memory Data Register (MDR)

MDR ← M (MAR)

↓

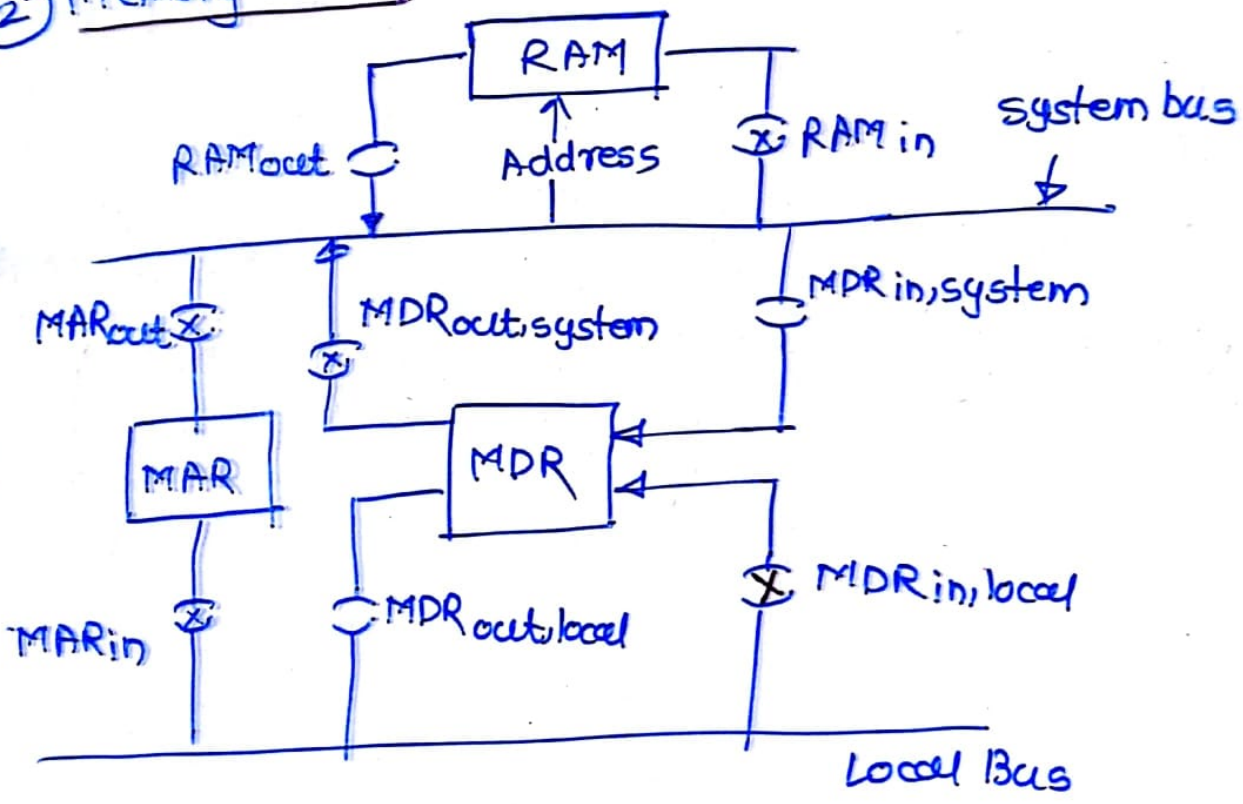
Read from memory, address supplied by MAR.

Sequence of control signals for read operation:

1st cycle: MARin

2nd cycle: MARout, RAMout, MDRin, system

(2) Memory Write



Write the data: • MAR puts the address on address bus and DR puts the data on data bus.

A write control signal along with these signals enables the data on data bus to be written into the memory location addressed by MAR. This

This operation can be written as

$M(MAR) \leftarrow MDR$

write to memory address supplied by MAR

Sequence of control signals for write operation

1st cycle: MARin

2nd cycle: MDRin, local

3rd cycle: MARout MDRout, system, RAMin

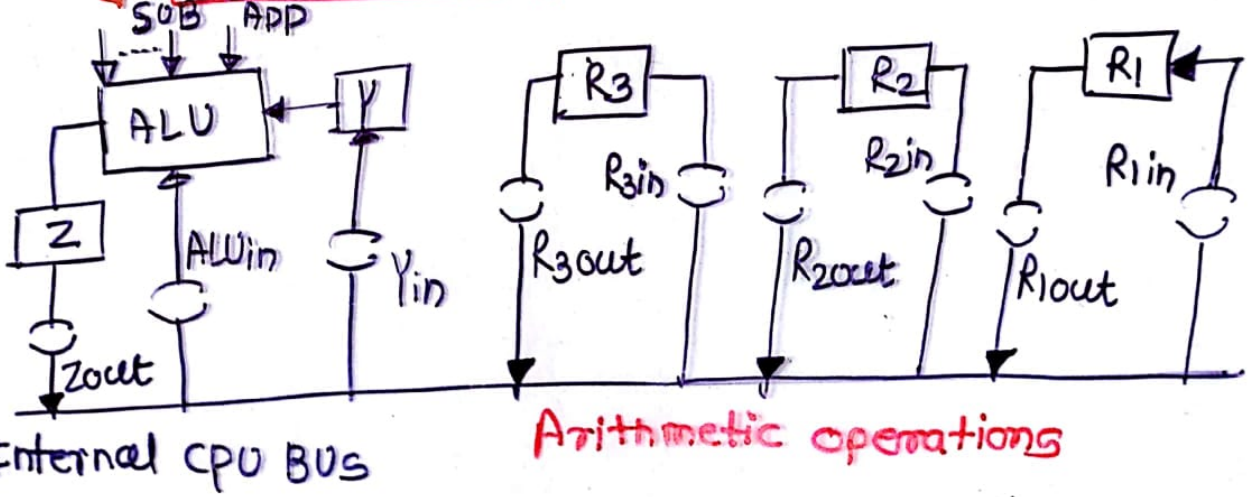
Memory read and memory write can not be performed parallely as they use the same internal CPU bus.

Arithmetic Micro-operations:

Arithmetic micro-operations are performed by ALU. These micro-operations perform some basic arithmetic operation on numeric data. These basic operations may be addition, subtraction, incrementing a number, decrementing a number and arithmetic shift operations.

control signals for various arithmetic operations

$R_3 \leftarrow R_1 + R_2$



step 1: contents of R_1 are stored in register Y
control signals: R_1out, Y_{in}

step 2: contents of R_2 and y are sent to ALU and control signals: R_2out, ALUin, ADD.
After step 2, result will be stored in register Z.

step 3: contents of register Z are transformed to the CPU register R_3.
control signals: Z_{out}, R_3in.

steps have been summarized in the table given below.

operation $R_3 \leftarrow R_1 + R_2$

step	operation	control signal
1	$Y \leftarrow R_1$	R_1out, Y_{in}
2	$Z \leftarrow Y + R_2$	R_2out, ALUin, ADD
3	$R_3 \leftarrow Z$	Z_{out}, R_3in

LOGIC MICRO-OPERATION

Logic operations are basically the binary operations which are performed on the string of bits stored in the registers. A logic micro-operation specifies AND operation to be performed on the contents of R_1 and R_2 and store the results in R_3.

$R_3 \leftarrow R_1 \wedge R_2$

Logic micro-operations are implemented in ALU and they are performed exactly the way, arithmetic operations are performed. eg AND, OR, NOT, Exclusive OR.

Shift Micro-operations

Shift is useful operation which can be used for serial transfer of data. Shift operations can also be used along with arithmetic operations. For e.g multiply operation is implemented using add and shift operations.

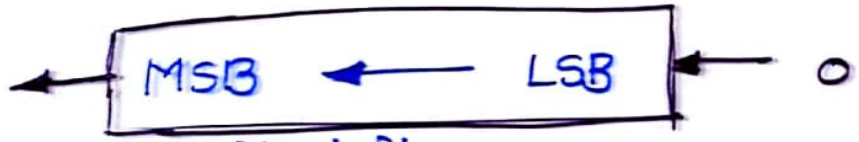
- Logical shift
- Arithmetic shift
- Circular shift.

Logical shift: Data can be shifted left or right by the number of bits specified in the instructio

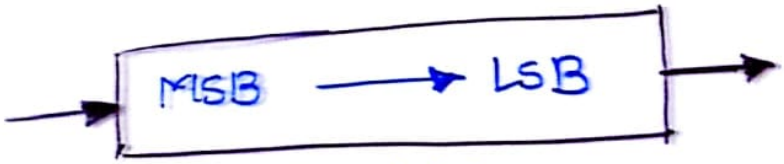
Example

SHL R1,3 ; shift left R1 by 3 bits

SHL R1,2 ; shift right R1 by 2bits.



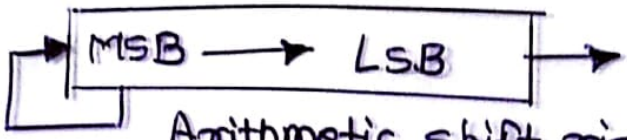
Shift left



Shift right

Initial value of R1	value of R1 after shift operation	operation
01001101	01101000	SHL R1,3
01001101	00010011	SHR R1,2

Arithmetic Shift: A signed data can be shifted left or right by the number of bits specified in the instruction. Arithmetic shift left operation is similar to logical shift operation but
in case of arithmetic shift right operation, as the data is shifted right, the bit value in MSB (sign bit) is remained so that the sign bit will not change.

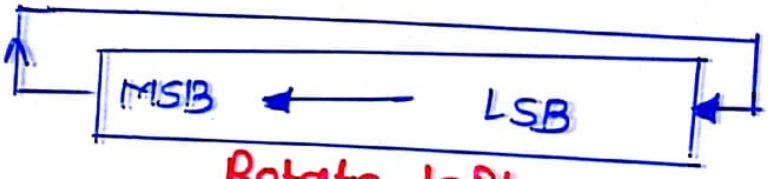


Arithmetic shift right

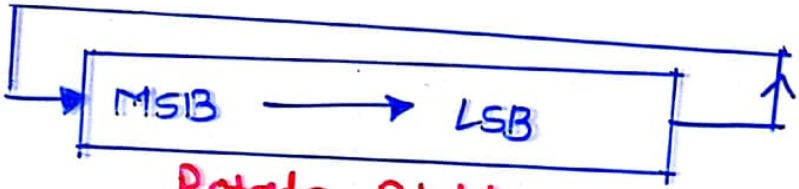
Example

Initial value of R1	value of R1 after shift operation	operation
10101100	11110101	SAR R1,3
01011011	00010110	SAR R1,2

Circular Shift (rotate): Data can be rotated left or right by the number of bits specified in the instruction



Rotate left



Rotate Right

Initial value of R1	value of R1 after shift operation	operation
10101100	10010101	ROR R1,3 → Rotate right R1 by 3 bits
01011011	01101101	ROL R1,2 → Rotate left R1 by 2 bits.