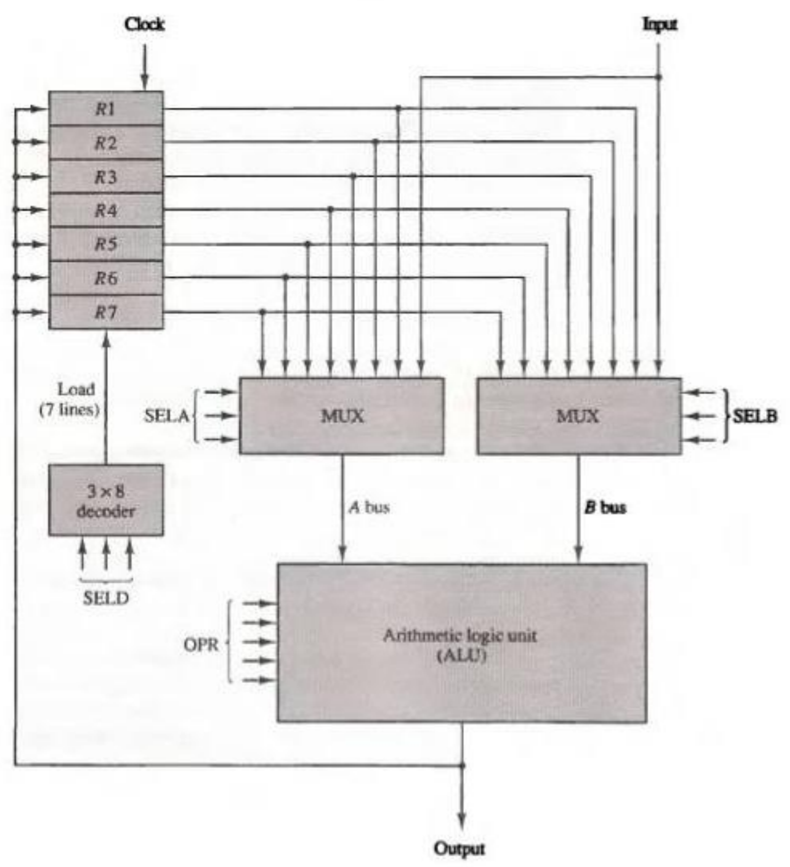
SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING

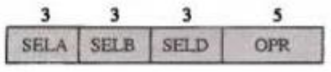
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

UNIT – V– SBS1203 – COMPUTER ARCHITECTURE

5.1.GENERAL PURPOSE REGISTERS



(a) Block diagram



(b) Control word

The output of each register is connected to two multiplexers (MUX) to form the two buses A and B.

The selection lines in each multiplexer select one register or the input data for the particular bus.

The A and B buses form the inputs to a common arithmetic logic unit (ALU). The operation selected in the ALU determines the arithmetic or logic micro operation that is to be performed.

The result of the micro operation is available for output data and also goes into the inputs of all the registers.The register that receives the information from the output bus is selected by a decoder. The decoder activates one of the register load inputs, thus providing a transfer path between the data in the output bus and the inputs of the selected destination register.

The control unit that operates the CPU bus system directs the information flow through the registers and ALU by selecting the various components in the system.

For example, to perform the operation

R1<--R2 + R3

the control must provide binary selection variables to the following selector inputs:

1. MUX A selector (SELA): to place the content of R2 into bus A.

2. MUX B selector (SELB): to place the content of R3 into bus B.

3. ALU operation selector (OPR): to provide the arithmetic addition A+ B. 4. Decoder destination selector (SELD): to transfer the content of the output bus into R 1.

5.1.1.CONTROL WORD

The combined value of a binary selection inputs specifies the control word.

It consist of four fields SELA,SELB,and SELD or SELREG contains three bit each and SELOPR field contains four bits thus the total bits in the control word are 13-bits.

The three bit of SELA select a source registers of the a input of the ALU.

The three bits of SEL B select a source registers of the b input of the ALU.

The three bits of SEL D or SEL REG select a destination register using the decoder.

The four bits of SEL OPR select the operation to be performed by ALU.

5.2. STACK ORGANIZATION

A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved.

The operation of a stack can be compared to a stack of trays. The last tray placed on top of the stack is the first to be taken off.

The stack in digital computers is essentially a memory unit with an address register that can count only (after an initial value is loaded into it). The register that holds the address for the stack is called a stack pointer (SP) because its value always points at the top item in the stack.

The physical registers of a stack are always available for reading or writing. It is the content of the word that is inserted or deleted.

The register that holds the address for the stack is called a stack pointer (SP) because its value always points at the top item in the stack.

The physical registers of a stack are always available for reading or writing. It is the content of the word that is inserted or deleted.

Types of Stack

- Register Stack
- Memory Stack
- Register Stack
- A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers.

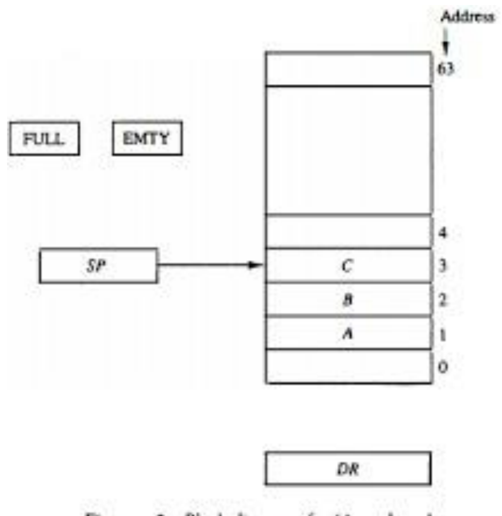


Figure 3 Block diagram of a 64-word stack.

Fig.5.2. Register Stack

The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on top of the stack. Three items are placed in the stack: A, B, and C, in that order.

Item C is on top of the stack so that the content of SP is now 3. To remove the top item, the stack is popped by reading the memory word at address 3 and decrementing the content of SP.

Item B is now on top of the stack since SP holds address 2. To insert a new item, the stack is pushed by incrementing SP and writing a word in the next-higher location in the stack. Note that item C has been read out but not physically removed.

Initially, SP is cleared to 0, EMTY is set to 1, and FULL is cleared to 0, so that SP points to the word at address 0 and the stack is marked empty and not full. If the stack is not full (if FULL = 0), a new item is inserted with a push operation.

The push operation is implemented with the following sequence of microoperations:

SP <-SP + 1 - Increment stack pointer

M[SP]<-DR - Write item on top of the stack

If (SP = 0) then (FULL <--1) - Check if stack is full

EMTY <--0 - Mark the stack not empty

A new item is deleted from the stack if the stack is not empty (if EMTY = 0). The pop operation consists of the following sequence of microoperations:

DR <--M[SP] -Read item from the top of stack

SP<--SP - 1 - Decrement stack pointer

If (SP = 0) then (EMTY <--1) -Check if stack is empty

FULL <--0 - Mark the stack not full

The top item is read from the stack into DR. The stack pointer is then decremented. If its value reaches zero, the stack is empty, so EMTY is set to 1.

This condition is reached if the item read was in location L Once this item is read out, SPis decremented and reaches the value 0, which is the initial value of SP.

Note that if a pop operation reads the item from location 0 and then SP is decremented, SP changes to 111111, which is equivalent to decimal 63.

In this configuration, the word in address 0 receives the last item in the stack. Note also that an erroneous operation will result if the stack is pushed when FULL = 1 or popped when EMTY = 1.

5.2.1.Memory Stack

The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer.

A portion of computer memory partitioned into three segments: program, data, and stack.

The program counter PC points at the address of the next instruction in the program. The address register AR points at an array of data.

The stack pointer SP points at the top of the stack. The three registers are connected to a common address bus, and either one can provide an address for memory.

PC is used during the fetch phase to read an instruction. AR is used during the execute phase to read an operand. SP is used to push or pop items into or & from the stack.

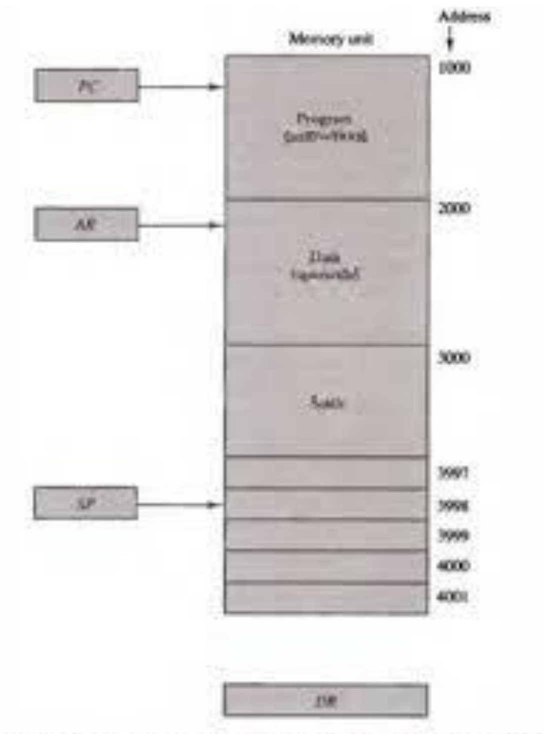


Figure 4 Computer memory with program, data, and stack segments.

Fig.5.3. Memory Stack

The initial value of SP is 4001 and the stack grows with decreasing addresses. Thus the first item stored in the stack is at address 40,00 the second item is stored at address 39,99 and the last address that can be used for the stack Is 3000. No provisions are available for stack limit checks.

We assume that the items in the stack communicate with a data register DR. A new item is inserted with the push operation as follows:

SP<-SP - 1

[SP]<-DR

The stack pointer is decremented so that it points at the address of the next word. A memory write operation inserts the word from DR into the top of the stack.

A new item is deleted with a pop operation as follows:

DR <-M[SP]

SP<-SP + 1

The top item is read from the stack into DR. The stack pointer is then incremented to point at the next item in the stack.

5.3.INSTRUCTION FORMATS

The physical and logical structure of computers is normally described in reference manuals provided with the system. Such manuals explain the internal construction of the CPU, including the processor registers available and their logical capabilities.

They list all hardware-implemented instructions, specify their binary code format, and provide a precise definition of each instruction.

The bits of the instruction are divided into groups called fields. The most common fields found in instruction formats are:

An operation code field that specifies the operation to be performed. An address field that designates a memory address or a processor register. A mode field that specifies the way the operand or the effective address is determined.

The operation code field of an instruction is a group of bits that define various processor operations, such as add, subtract, complement, and shift.

The most common operations available in computer instructions are enumerated and discussed.

Three types of CPU organizations:

Single accumulator organization.

– All operations are performed with an implied accumulator register. The instruction format in this type of computer uses one address field.

– E.g ADD X

General register organization.

– The instruction format in this type of computer needs three register address fields.

– E.g. ADD R1, R2, R3

Stack organization.

– Computers with stack organization would have PUSH and POP instructions which require an address field.

– E.g PUSH X

Three Address Instructions

Two Address Instructions

One Address Instructions

5.3.1.Three Address Instructions

Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand.

E.g. X = (A + B) • (C + D)

```
ADD    R1, A, B     R1<-- M[A] + M[B]
ADD    R2, C, D     R2<-- M[C] + M[D]
MOL    X, R1, R2    M[X] <-- R1•R2
```

The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions.

The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.

5.3.2TWO ADDRESS INSTRUCTIONS

Two-address instructions are the most common in commercial computers.

```
MOV  R1, A   R1<--M[A]
ADD  R1, B   R1 <-- R1 + M[B]
MOV  R2, C   R2<-- M[C]
ADD  R2, D   R2<-- R2 + M[D]
MUL  R1,R2   R1 <-- R1• R2
MOV  X, R1   M[X] <-- R1
```

The MOV instruction moves or transfers the operands to and from memory and processor registers.

The first symbol listed in an instruction is assumed to be both a source and the destination where the result of the operation is transferred.

5.3.3.One Address Instructions

One-address instructions use an implied accumulator (AC) register for all data manipulation.

For multiplication and division there is a need for a second register.

LOAD A	AC<-M[A]
ADD B	AC <-AC + M[B]
STORE T	M[T] <-AC
LOAD C	AC<-M[C]
ADD D	AC<-AC + M[D]
MUL T	AC<-AC•M[T]
STORE X	M[X] <-AC

All operations are done between the All operations are done between the AC register and a memory operand.

T is the address of a temporary memory location required for storing the intermediate result.

5.3.4.Zero Address Instructions

A stack-organized computer does not use an address field for the instructions ADD and MUL.

The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack.

PUSH A	TOS<-A
PUSH B	TOS<-B
ADD	TOS<-(A+B)
PUSH C	TOS <-C
PUSH D	TOS<-D
ADD	TOS<-(C+D)

MUL TOS<-(C +D)•(A+B)

POP X M[X]<-TOS

The name "zero-address" is given to this type of computer because of the absence of an address field in the computational instructions.

5.4. ADDRESSING MODES

Computers use addressing mode techniques for the purpose of accommodating one or both of the following provisions:

1. To give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data, and program relocation.

2. To reduce the number of bits in the addressing field of the instruction.

The control unit of a computer is designed to go through an instruction cycle that is divided into three major phases:

1. Fetch the instruction from memory.

2. Decode the instruction.

3. Execute the instruction.

Program Counter (PC) There is one register in the computer called the program counter or PC that keeps track of the instructions in the program stored in memory.

Mode Field : The mode field is used to locate the operands needed for the operation.

Implied Mode: In this mode the operands are specified implicitly in the definition of the instruction. For example, the instruction "complement accumulator" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction.

E.g. CLC (used to reset Carry flag to 0)

Immediate addressing mode: In this mode data is present in address field of instruction .Designed like one address instruction format.

E.g. MOV AL, 35H (move the data 35H into AL register)

Register mode: In register addressing the operand is placed in one of 8 bit or 16 bit general purpose registers. The data is in the register that is specified by the instruction. Here one register reference is required to access the data. E.g. MOV AX,CX (move the contents of CX register to AX register)

Register Indirect Mode : In this mode, the instruction specifies the register whose contents give us the address of operand which is in memory.

Thus, the register contains the address of operand rather than the operand itself.

E.g. MOV AX, [BX](move the contents of memory location s addressed by the register BX to the register AX)

Auto Increment Mode: Effective address of the operand is the contents of a register specified in the instruction.

After accessing the operand, the contents of this register are automatically incremented to point to the next consecutive memory location.

E.g. Add R1, (R2)+ // OR

R1 = R1 +M[R2]

R2 = R2 + d

Direct Addressing/Absolute Addressing Mode : The operand's offset is given in the instruction as an 8 bit or 16 bit displacement element.

In this addressing mode the 16 bit effective address of the data is the part of the instruction.

E.g ADD AL,[0301] //add the contents of offset address 0301 to AL

Indirect Address Mode: In this mode the address field of the instruction gives the address where the effective address is stored in memory. Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.

EFFECTIVE ADDRESS = address part of instruction + content of CPU register.

E.g : MOV AX, [SI +05]

Based Indexed Addressing: The operand's offset is sum of the content of a base register BX or BP and an index register SI or DI.

E.g.: ADD AX, [BX+SI]

PC Relative Addressing Mode: PC relative addressing mode is used to implement intra segment transfer of control, In this mode effective address is obtained by adding displacement to PC.EA= PC + Address field value PC= PC + Relative value.

Base Register Addressing Mode: Base register addressing mode is used to implement inter segment transfer of control.

In this mode effective address is obtained by adding base register value to address field value.

EA= Base register + Address field value.

PC= Base register + Relative value.

5.5.DATA TRNSFER AND MANIPULATION

Data manipulation instructions perform operations on data and provide the computational capabilities for the computer. The data manipulation instructions in a typical computer are usually divided into three basic types:

1. Arithmetic instructions

2. Logical and bit manipulation instructions

3. Shift instructions

5.5.1.Arithmetic Instructions

The four basic arithmetic operations are addition, subtraction, multiplication, and division. Most computers provide instructions for all four operations. Some small computers have only addition and possibly subtraction instructions.

The multiplication and division must then be generated by means of software subroutines.

Increment ,Decrement, Add ,Subtract ,Multiply, Divide, Add with carry ,Subtract with borrow Negate (2's complement).

ADDI	Add two binary integer numbers
ADDF	Add two floating-point numbers
ADDD	Add two decimal numbers in BCD

The number of bits in any register is of finite length and therefore the results of arithmetic operations are of finite precision

5.5.2.Logical and bit manipulation instructions

Logical instructions perform binary operations on strings of bits stored in registers.

They are useful for manipulating individual bits or a group of bits that represent binary-coded information. The logical instructions consider each bit of the operand separately and treat it as a Boolean variable.

Clear ,Complement, AND, OR, Exclusive-OR, Clear, carry, Set carry ,Complement carry, Enable interrupt, Disable interrupt.

The complement instruction produces the 1's complement by inverting all the bits of the operand. The OR instruction is used to set a bit or a selected group of bits of an operand.

5.5.3. Shift Instructions

Instructions to shift the content of an operand are quite useful and are often provided in several variations.

Shifts are operations in which the bits of a word are moved to the left or right.

Some computers have a multiple-field format for the shift instructions.

One field contains the operation code and the others specify the type of shift and the number of times that an operand is to be shifted. A possible instruction code format of a shift instruction may include five fields as follows: OP REG TYPE RL COUNT

Here OP is the operation code field; REG is a register address that specifies the location of the operand.

Logical shift right, Logical shift left, Arithmetic shift right, Arithmetic shift left, Rotate right, Rotate left, Rotate right through carry, Rotate left through carry.

5.6.PROGRAM CONTROL

Instructions are always stored in successive memory locations. When processed in the CPU, the instructions are fetched from consecutive memory locations and executed.

Each time an instruction is fetched from memory, the program counter is incremented so that it contains the address of the next instruction in sequence.

After the execution of a data transfer or data manipulation instruction, control returns to the fetch cycle with the program counter containing the address of the instruction next in sequence.

A program control type of instruction, when executed, may change the address value in the program counter and cause the flow of control to be altered.

Branch and jump instructions may be conditional or unconditional.

An unconditional branch instruction causes a branch to the specified address without any conditions.

The conditional branch instruction specifies a condition such as branch if positive or branch if zero.

Branch, Jump, Skip, Call, Return, Name, Compare (by subtraction), Test (by ANDing).

Status Bit Conditions

Status bits are also called condition-code bits or flag bits.

The four status bits are symbolized by C. S, Z, and V.

The bits are set or cleared as a result of an operation performed in the ALU.

Bit C (carry) is set to 1 if the end carry C8 is 1. It is cleared to 0 if the carry is 0.

Bit S (sign) is set to 1 if the highest-order bit F, is 1. It is set to 0 if the bit is 0.

Bit Z (zero) is set to 1 if the output of the ALU contains all O's. It is cleared to otherwise. In other words, Z = 1 if the output is zero and Z = 0 if the output is not zero.

Bit V (overflow) is set to 1 if the exclusive-OR of the last two carries is equal to 1, and cleared to 0 otherwise.

Conditional Branch Instructions

BZ BNZ BC BNC BP BM BY BNV

Subroutine Call And Return

A subroutine is a self-contained sequence of instructions that performs a given computational task.

The instruction that transfers program control to a subroutine is known by different names.

The most common names used are call subroutine, jump to subroutine, branch to subroutine, or branch and save address.

A call subroutine instruction consists of an operation code together with an address that specifies the beginning of the subroutine.

5.7. PROGRAM INTERRUPT

There are three major types of interrupts that cause a break in the normal execution of a program.

1.External interrupts

2. Internal interrupts

3. Software interrupts

5.8.REDUCED INSTRUCTION SET COMPUTER (RISC)

The instruction set chosen for a particular computer determines the way that machine language programs are constructed.

Early computers had small and simple instruction sets, forced mainly by the need to minimize the hardware used to implement them.

As digital hardware became cheaper with the advent of integrated circuits, computer instuctions tended to increase both in number and complexity. Many computers have instruction sets that include more than 100 and sometimes even more than 200 instructions.

A computer with a large number of instructions is classified as a complex instruction set computer, abbreviated CISC.

RISC Characteristics The concept of RISC architecture involves an attempt to reduce execution time by simplifying the instruction set of the computer.

The major characteristics of a RISC processor are:

1. Relatively few instructions

2. Relatively few addressing modes

3.Memory access limited to load and store instructions

4. All operations done within the registers of the CPU

5. Fixed-length, easily decoded instruction format

6. Single-cycle instruction execution

7.Hardwired rather than micro programmed control

5.9.PARALLEL PROCESSING

Parallel processing is a term used to denote a large class of techniques that are used to provide simultaneous data-processing tasks for the purpose of increasing the computational speed of a computer system.

The system may have two or more ALUs and be able to execute two or more instructions at the same time. The system may have two or more processors operating concurrently.

The purpose of parallel processing is to speed up the computer processing capability and increase its throughput, that is, the amount of processing that can be accomplished during a given interval of time.

Parallel processing at a higher level of complexity can be achieved by having a multiplicity of functional units that perform identical or different operations simultaneously.

Parallel processing is established by distributing the data among the multiple functional units.

For example: the arithmetic, logic, and shift operations can be separated into three units and the operands diverted to each unit under the supervision of a control unit.

Processor with Different Functional Unit

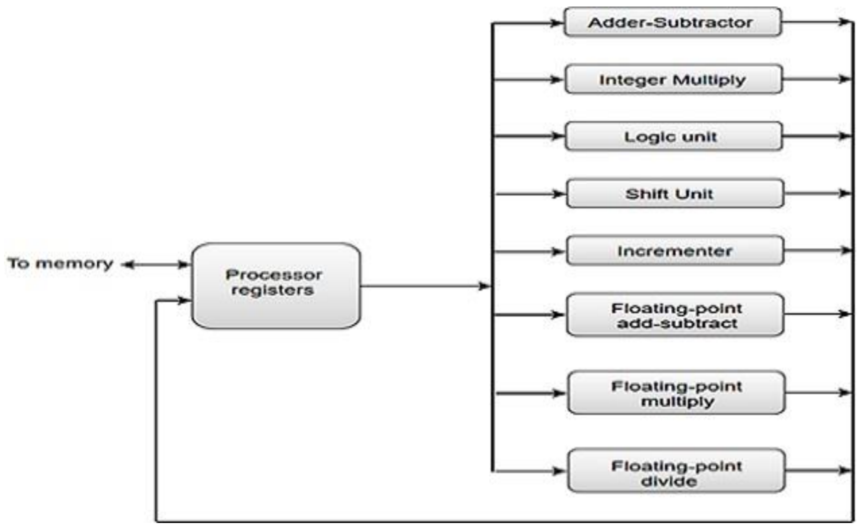


Fig.5.6. Processor with Different Functional Units

There are a variety of ways that parallel processing can be classified.

It can be considered from the internal organization of the processors, from the interconnection structure between processors, or from the flow of information through the system.

One classification introduced by M. J. Flynn considers the organization of a computer system by the number of instructions and data items that are manipulated simultaneously.

The normal operation of a computer is to fetch instructions from memory and execute them in the processor.

The sequence of instructions read from memory constitutes an instruction stream. The operations performed on the data in the processor constitutes a data stream.

Parallel processing may occur in the instruction stream, in the data stream, or in both.

Flynn's classification divides computers into four major groups as follows:

1. Single instruction stream, single data stream (SISD)
2. Single instruction stream, multiple data stream (SIMD)
3. Multiple instruction stream, single data stream (MISD)
4. Multiple instruction stream, multiple data stream (MIMD)

SISD represents the organization of a single computer containing a control unit, a processor unit, and a memory unit.

Instructions are executed sequentially and the system may or may not have internal parallel processing capabilities.

Parallel processing in this case may be achieved by means of multiple functional units or by pipeline processing.

SIMD represents an organization that includes many processing units under the supervision of a common control unit.

All processors receive the same instruction from the control unit but operate on different items of data.

The shared memory unit must contain multiple modules so that it can communicate with all the processors simultaneously.

MISD structure is only of theoretical interest since no practical system has been constructed using this organization.

MIMD organization refers to a computer system capable of processing several programs at the same time. Most multiprocessor and multicomputer systems can be classified in this category.

Flynn's classification depends on the distinction between the performance of the control unit and the data-processing unit.

- 1. Pipeline processing

- 2. Vector processing
- 3. Array processors

Pipeline processing is an implementation technique where arithmetic sub operations or the phases of a computer instruction cycle overlap in execution.

Vector processing deals with computations involving large vectors and matrices. Array processors perform computations on large arrays of data.

5.10. PIPELINING

Pipelining is a technique of decomposing a sequential process into sub operations, with each sub process being executed in a special dedicated segment that operates concurrently with all other segments.

The sub operations performed in each segment of the pipeline are as follows:

R1<--A„ R2<--B,	Input A, and B,
R3<--R1•R2, R4<--C,	Multiply and input C,
R5<--R3 + R4	Add C; to product

The five registers are loaded with new data every clock pulse

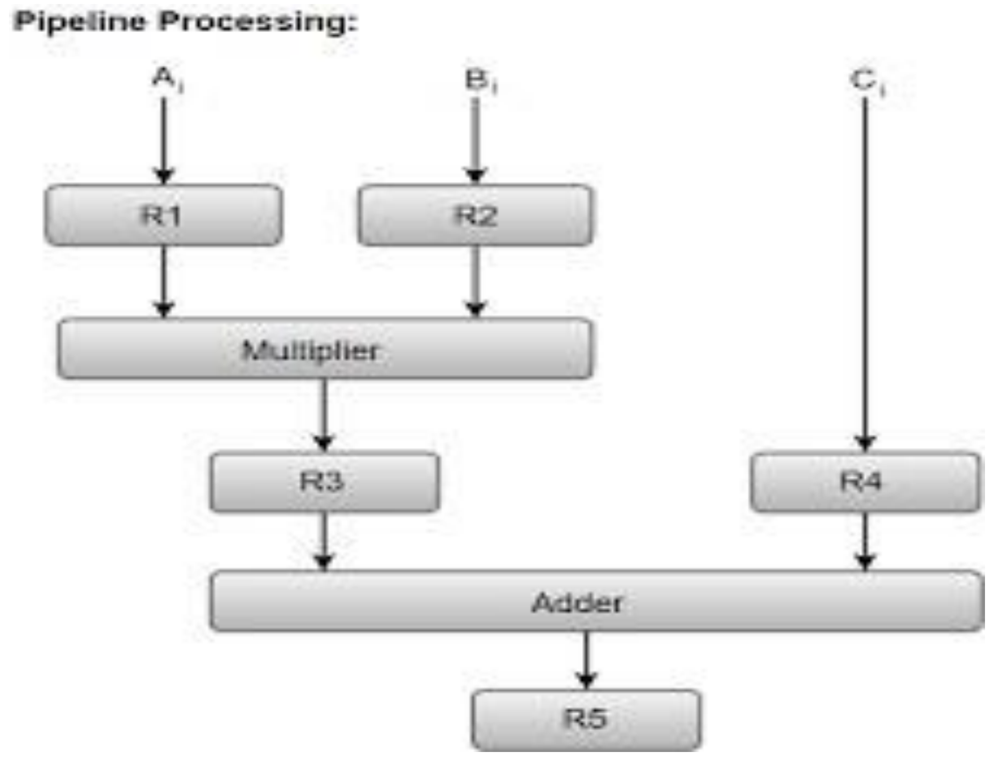


Fig.5.7. Pipeline Processing

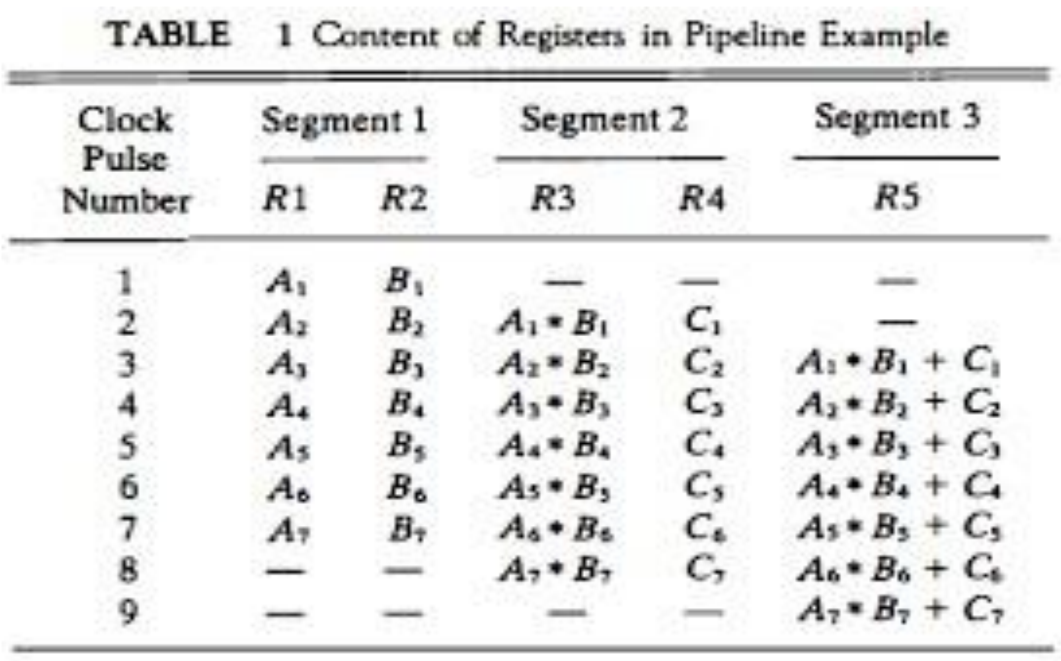


TABLE 1 Content of Registers in Pipeline Example

Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A_1	B_1	—	—	—
2	A_2	B_2	$A_1 * B_1$	C_1	—
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8	—	—	$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9	—	—	—	—	$A_7 * B_7 + C_7$

The first clock pulse transfers A1 and 81 into R 1 and R2. The second dock pulse transfers the product of R 1 and R2 into R3 and C1 into R4.

The same clock pulse transfers A2 and B2 into R 1 and R2. The third clock pulse operates on all three segments simultaneously.

It places A, and B, into R1 and R2, transfers the product of R1 and R2 into R3, transfers C, into R4, and places the sum of R3 and R4 into RS.

It takes three clock pulses to fill up the pipe and retrieve the first output from RS.

Arithmetic Pipeline

Instruction Pipeline

5.10.1.ARITHMETIC PIPELINE

Pipeline arithmetic units are usually found in very high speed computers.

They are used to implement floating-point operations, multiplication of fixed-point numbers, and similar computations encountered in scientific problems. The inputs to the floating-point adder pipeline are two normalized floating-point binary numbers.

E.g : $X=A \times 2^a$ $Y = B \times 2^b$

A and B are two fractions that represent the mantissas and a and b are the exponents. The floating-point addition and subtraction can be performed in four segments.

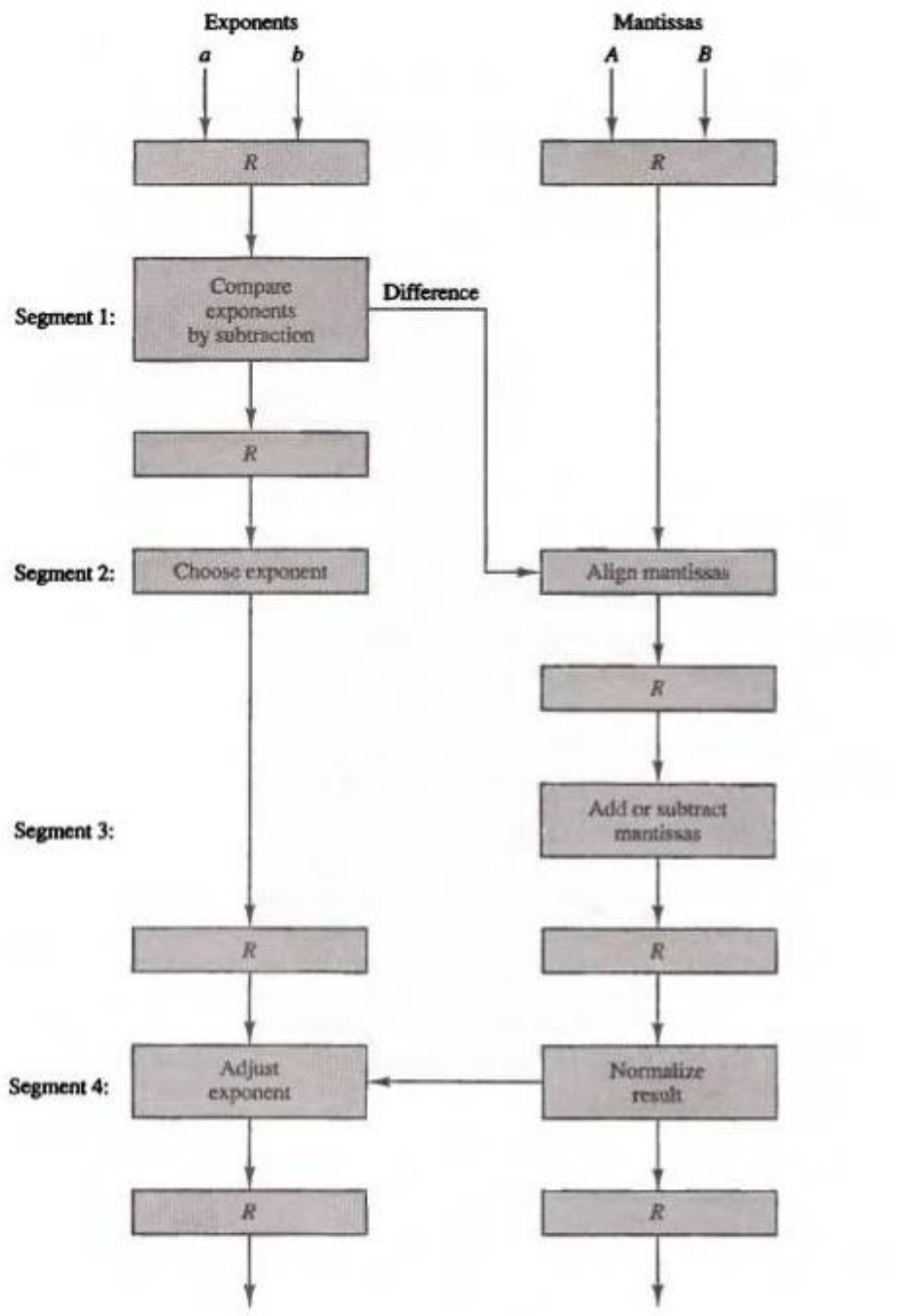


Figure 9-6 Pipeline for floating-point addition and subtraction.

The sub operations that are performed in the four segments are:

1. Compare the exponents.
2. Align the mantissas.
3. Add or subtract the mantissas.
4. Normalize the result.

The exponents are compared by subtracting them to determine their difference.

The larger exponent is chosen as the exponent of the result.

The exponent difference determines how many times the mantissa associated with the smaller exponent must be shifted to the right. This produces an alignment of the two mantissas. It should be noted that the shift must be designed as a combinational circuit to reduce the shift time.

The two mantissas are added or subtracted in segment 3.

The result is normalized in segment 4. When an overflow occurs, the mantissa of the sum or difference is shifted right and the exponent incremented by one.

If an underflow occurs, the number of leading zeros in the mantissa determines the number of left shifts in the mantissa and the number that must be subtracted from the exponent.

5.10.2INSTRUCTION PIPELINE

Pipeline processing can occur not only in the data stream but in the instruction stream as well.

An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments. This causes the instruction fetch and execute phases to overlap and perform simultaneous operations. Pipeline processing can occur not only in the data stream but in the instruction stream as well.

An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments.

This causes the instruction fetch and execute phases to overlap and perform simultaneous operations. Computer with an instruction fetch unit and an instruction execution unit designed to provide a two-segment pipeline.

The instruction fetch segment can be implemented by means of a first-in, first-out (FIFO) buffer. This is a type of unit that forms a queue rather than a stack.

Whenever the execution unit is not using memory, the control increments the program counter and uses its address value to read consecutive instructions from memory.

The instructions are inserted into the FIFO buffer so that they can be executed on a first-in, first-out basis.

Computers with complex instructions require other phases in addition to the fetch and execute to process an instruction completely.

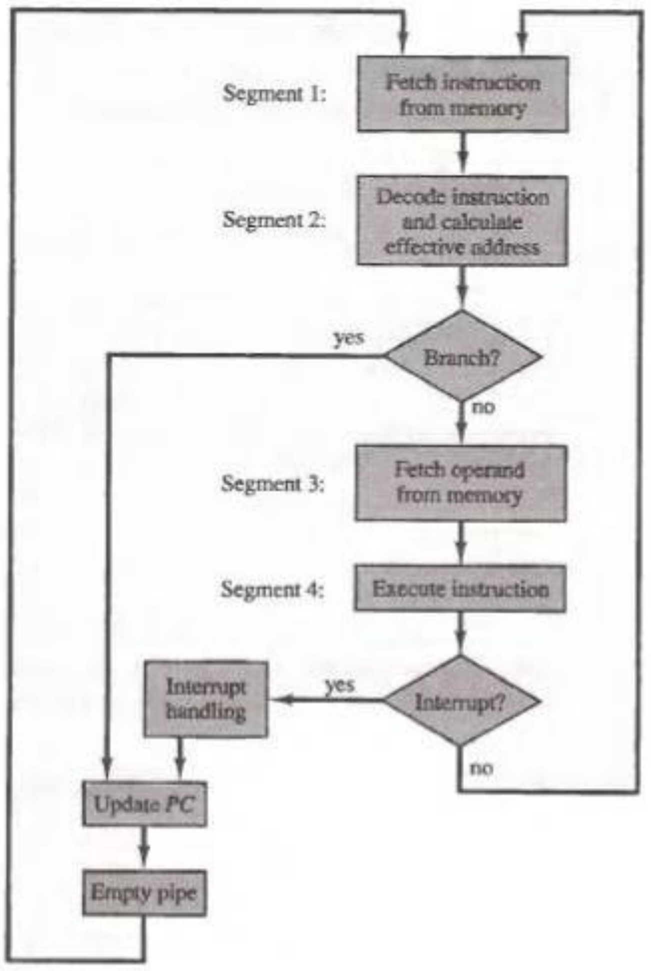


Fig.5.9. Instruction Pipelininig

In the most general case, the computer needs to process each instruction with the following sequence of steps.

1. Fetch the instruction from memory.

2. Decode the instruction.

3. Calculate the effective address.

4. Fetch the operands from memory.

5. Execute the instruction.

6. Store the result in the proper place.

There are certain difficulties that will prevent the instruction pipeline from operating at its maximum rate. Different segments may take different times to operate on the incoming information. Some segments are skipped for certain operations.

1. Fl is the segment that fetches an instruction.

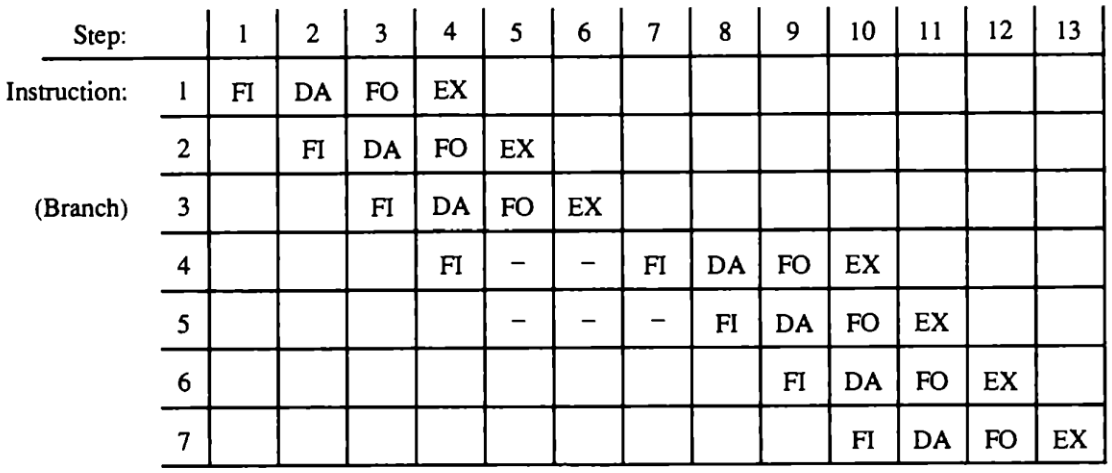
2. DA is the segment that decodes the instruction and calculates the effective

address.

3. FO is the segment that fetches the operand.

4. EX is the segment that executes the instruction.

For example, a register mode instruction does not need an effective address calculation.



Step:		1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction:	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	3			FI	DA	FO	EX							
	4				FI	–	–	FI	DA	FO	EX			
	5					–	–	–	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

Fig.5.10. Timing of Instruction Pipeline

Two or more segments may require memory access at the same time, causing one segment to wait until another is finished with the memory.

Memory access conflicts are sometimes resolved by using two memory buses for accessing instructions and data in separate modules.

5.11.VECTOR PROCESSING

There is a class of computational problems that are beyond the capabilities of a conventional computer.

These problems require vast number of computations on multiple data items, that will take a conventional computer (with scalar processor) days or even weeks to complete.

Such complex instructions, which operates on multiple data at the same time, requires a better way of instruction execution, which was achieved by Vector processors.

Scalar CPUs can manipulate one or two data items at a time, which is not very efficient. Also, simple instructions like **ADD A to B, and store into C** are not practically efficient.

Addresses are used to point to the memory location where the data to be operated will be found, which leads to added overhead of data lookup. So until the data is found, the CPU would be sitting ideal, which is a big performance issue.

Hence, the concept of **Instruction Pipeline** comes into picture, in which the instruction passes through several sub-units in turn. These sub-units perform various independent functions,

For example: the **first** one decodes the instruction, the **second** sub-unit fetches the data and the **third** sub-unit performs the math itself.

Therefore, while the data is fetched for one instruction, CPU does not sit idle, it rather works on decoding the next instruction set, ending up working like an assembly line.

Vector processor, not only use Instruction pipeline, but it also pipelines the data, working on multiple data at the same time.

A normal scalar processor instruction would be ADD A, B, which leads to addition of two operands, but what if we can instruct the processor to ADD a group of numbers(from 0 to n memory location) to another group of numbers(lets say, n to k memory location). This can be achieved by vector processors.

In vector processor a single instruction, can ask for multiple data operations, which saves time, as instruction is decoded once, and then it keeps on operating on different data items.

- Applications of Vector Processors

– Computer with vector processing capabilities are in demand in specialized applications. The following are some areas where vector processing is used:

– Petroleum exploration.

– Medical diagnosis.

– Data analysis.

- Weather forecasting.
- Aerodynamics and space flight simulations.
- Image processing.
- Artificial intelligence.

TYPES OF ARRAY PROCESSORS

There are basically two types of array processors:

1. Attached Array Processors
2. SIMD Array Processors

5.11.1.ATTACHED ARRAY PROCESSORS

An attached array processor is a processor which is attached to a general purpose computer and its purpose is to enhance and improve the performance of that computer in numerical computational tasks.

It achieves high performance by means of parallel processing with multiple functional units.

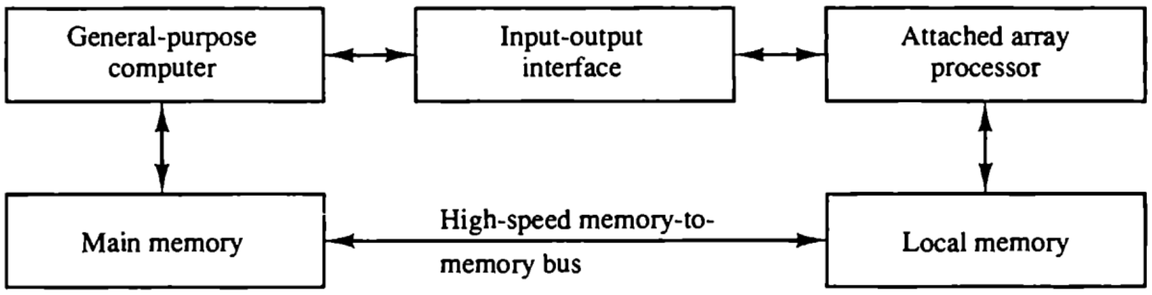


Fig.5.11. Attached Array Processor with Host Computer

5.11.2. SIMD ARRAY PROCESSORS

SIMD is the organization of a single computer containing multiple processors operating in parallel.

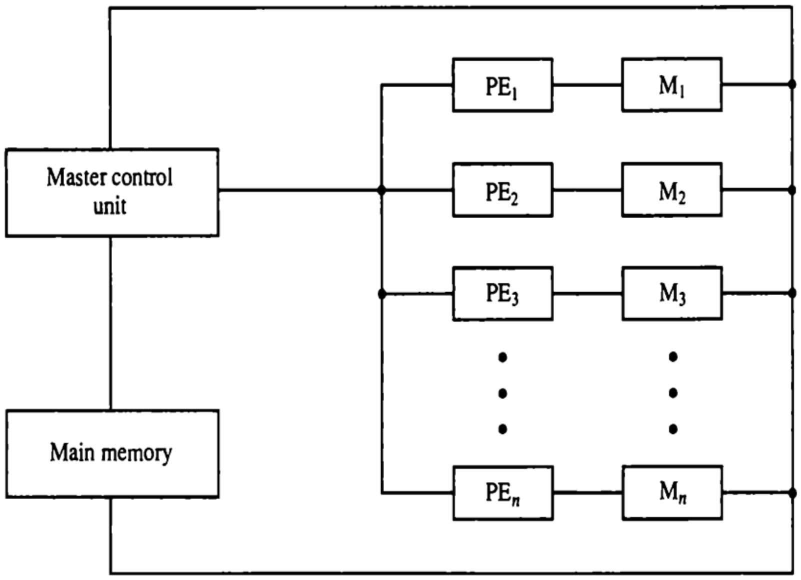


Fig.5.11. SIMD Processors

The processing units are made to operate under the control of a common control unit, thus providing a single instruction stream and multiple data streams.

A general block diagram of an array processor is shown below. It contains a set of identical processing elements (PE's), each of which is having a local memory M. Each processor element includes an ALU and registers.

The master control unit controls all the operations of the processor elements. It also decodes the instructions and determines how the instruction is to be executed.

The main memory is used for storing the program. The control unit is responsible for fetching the instructions. Vector instructions are send to all PE's simultaneously and results are returned to the memory.

The best known SIMD array processor is the ILLIAC IV computer developed by the Burroughs corps. SIMD processors are highly specialized computers. They are only suitable for numerical problems that can be expressed in vector or matrix form and they are not suitable for other types of computations.