Unit 5

Central Processing Unit

1. Briefly explain General Register Organization:

- When a large number of registers are included in the CPU, it is most efficient to connect them . through a common bus system.
- The registers communicate with each other not only for direct data transfers, but also while performing various microoperations. Hence it is necessary to provide a common unit that can perform all the arithmetic, logic, and shift microoperations in the processor.
- A bus organization for seven CPU registers is shown in Fig. 5-2.
- 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.

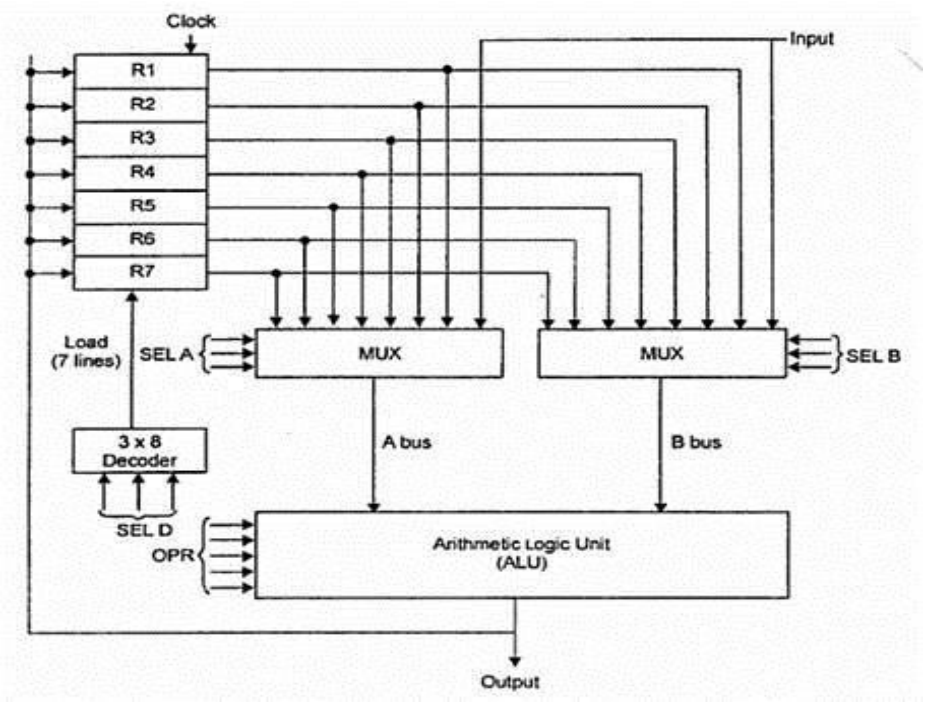


Figure 5-2 Register set with common ALU

- 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 microoperation that is to be performed.
- The result of the microoperation 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 R1.

Table 5.1 Encoding of Register Selection Fields

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

Table 5.2 Encoding of ALU Operations

OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add A + B	ADD
00101	Subtract A − B	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

EXAMPLE OF MICRO OPERATIONS:

- A control word of 14 bits is needed to specify a microoperation in the CPU. The control word for a given microoperation can be derived from the selection variables.
- For example, the subtract microoperation given by the statement.

    R1 ← R2 − R3

specifies R2 for the A input of the ALU, R3 for the B input of the ALU, R1 for the destination register, and an ALU operation to subtract A − B.

- Thus the control word is specified by the four fields and the corresponding binary value for each field is obtained from the encoding listed in Tables 5-1 and 5-2.
- The binary control word for the subtract microoperation is 010 011 001 00101 and is obtained as follows :

Field :	**SELA**	**SELB**	**SELD**	**OPR**
Symbol :	**R2**	**R3**	**R1**	**SUB**
Control word:	**010**	**011**	**001**	**00101**

Table 5-3 Examples of Microoperations for the CPU

Microoperation	SELA	SELB	SELD	OPR	Control Word			
R1←R2−R3	R2	R3	R1	SUB	010	011	001	00101
R4←R4 V R5	R4	R5	R4	OR	100	101	100	01010
R6←R6 + 1	R6	–	R6	INCA	110	000	110	00001
R7 ← R1	R1	–	R7	TSFA	001	000	111	00000
Output ←R2	R2	–	None	TSFA	010	000	000	00000
Output← Input	Input	–	None	TSFA	000	000	000	00000
R4← sh1 R4	R4	–	R4	SHLA	100	000	100	11000
R5 ← 0	R5	R5	R5	XOR	101	101	101	01100

2. What is stack? Give the organization of register stack with all necessary elements and explain the working of push and pop operations.

- A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved.
- The stack in digital computers is essentially a memory unit with an address register that can count only. 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.

Register stack:

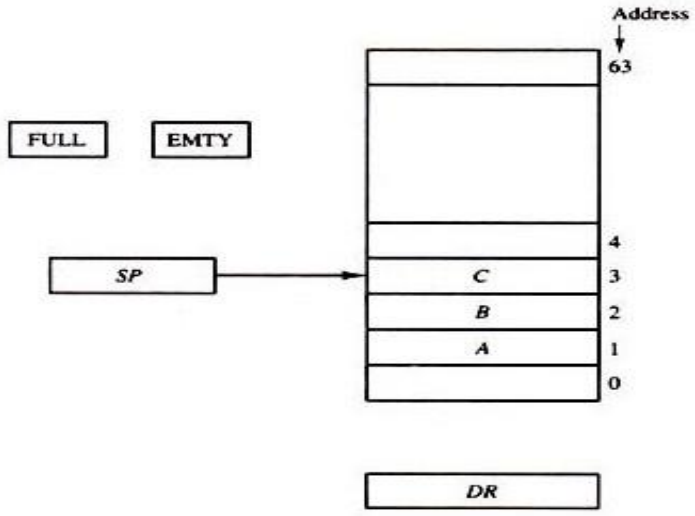


Figure 5.1: Block diagram of a 64-word stack

- 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 shows the organization of a 64-word 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.
- In a 64-word stack, the stack pointer contains 6 bits because $2^6 = 64$.
- Since SP has only six bits, it cannot exceed a number greater than 63 (111111 in binary). When 63 are incremented by 1, the result is 0 since 111111 + 1 = 1000000 in binary, but SP can accommodate only the six least significant bits.
- Similarly, when 000000 is decremented by 1, the result is 111111. The one-bit register FULL is set to 1 when the stack is full, and the one-bit register EMTY is set to 1 when the stack is empty of items.
- DR is the data register that holds the binary data to be written into or read out of the stack.

PUSH:

- If the stack is not full (FULL =0), a new item is inserted with a push operation. The push operation consists of the following sequences 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

- The stack pointer is incremented so that it points to the address of next-higher word. A memory write operation inserts the word from DR into the top of the stack.
- SP holds the address of the top of the stack and that M[SP] denotes the memory word specified by the address presently available in SP.
- The first item stored in the stack is at address 1. The last item is stored at address 0. If SP reaches 0, the stack is full of items, so FULL is set to 1. This condition is reached if the top item prior to the last push was in location 63 and, after incrementing SP, the last item is stored in location 0.
- Once an item is stored in location 0, there are no more empty registers in the stack. If an item is written in the stack, obviously the stack cannot be empty, so EMTY is cleared to 0.

POP:

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

DR ← M [SP]	Read item on top of the 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 1.

- Once this item is read out, SP is decremented and reaches the value 0, which is the initial value of SP. If a pop operation reads the item from location 0 and then SP is decremented, SP is 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.

3. Explain Memory Stack.

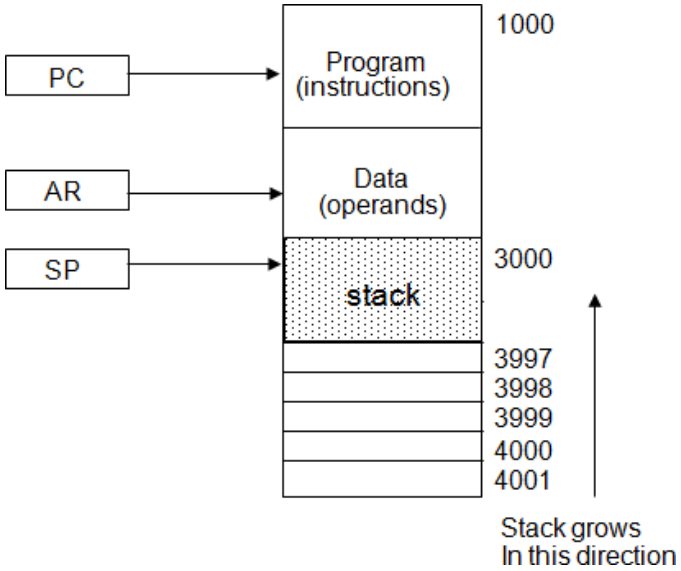


Figure 5.2: Computer memory with program, data, and stack segments

- 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.
- Figure 5.2 shows 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 which is used during the fetch phase to read an instruction.
- The address registers AR points at an array of data which is used during the execute phase to read an operand.
- The stack pointer SP points at the top of the stack which is used to push or pop items into or from the stack.
- The three registers are connected to a common address bus, and either one can provide an address for memory.
- As shown in Figure 5.2, 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 4000, the second item is stored at address 3999, and the last address that can be used for the stack is 3000.
- We assume that the items in the stack communicate with a data register DR.

PUSH

- A new item is inserted with the push operation as follows:

  $SP \leftarrow SP - 1$
  $M[SP] \leftarrow DR$
- 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.

POP

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

  $DR \leftarrow M[SP]$
  $SP \leftarrow SP + 1$
- 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.
- The two microoperations needed for either the push or pop are (1) an access to memory through SP, and (2) updating SP.
- Which of the two microoperations is done first and whether SP is updated by incrementing or decrementing depends on the organization of the stack.
- In figure. 5.2 the stack grows by decreasing the memory address. The stack may be constructed to grow by increasing the memory also.
- The advantage of a memory stack is that the CPU can refer to it without having to specify an address, since the address is always available and automatically updated in the stack pointer.

4. Explain four types of instruction formats.

Three Address Instructions:

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

The program in assembly language that evaluates X = (A + B) * (C + D) is shown below.

ADD R1, A, B	R1←M [A] + M [B]
ADD R2, C, D	R2 ← M[C] + M [D]
MUL X, R1, R2	M[X] ←R1 * R2

- The advantage of 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.
- An example of a commercial computer that uses three-address instruction is the Cyber 170.

Two Address Instructions:

- Two address instructions are the most common in commercial computers. Here again each address field can specify either a processor register or a memory word.

The program to evaluate X = (A + B) * (C + D) is as follows:

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.

One Address Instructions:

- One address instructions use an implied accumulator (AC) register for all data manipulation. For multiplication and division these is a need for a second register.
- However, here we will neglect the second register and assume that the AC contains the result of all operations. The program to evaluate X = (A + B) * (C + D) is

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 the operations are done between the AC register and a memory operand. T is the address of the temporary memory location required for storing the intermediate result.

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.

- The program to evaluate X = (A + B) * (C + D) will be written for a stack-organized computer

PUSH	A	TOS ← A
PUSH	B	TOS ←B
ADD		TOS ← (A + B)
PUSH	C	TOS ←B
PUSH	D	TOS ←D
ADD		TOS ← (C + D)
MUL		TOS ← (C + D) * (A + B)
POP	X	M [X]← TOS

- To evaluate arithmetic expressions in a stack computer, it is necessary to convert the expression into reverse polish notation.

RISC Instructions:

- All other instructions are executed within the registers of the CPU without referring to memory. A program for a RISC type CPU consists of LOAD and STORE instructions that have one memory and one register address, and computational-type instructions that have three addresses with all three specifying processor registers.
- The following is a program to evaluate X = (A + B) * (C + D).

LOAD	R1, A	R1← M [A]
LOAD	R1, B	R1 ← M [B]
LOAD	R1, C	R1 ← M [C]
LOAD	R1, D	R1 ←M [D]
ADD	R1, R1, R2	R1 ←R1 + R2
ADD	R3, R3, R2	R3 ←R3 + R4
MUL	R1, R1, R3	R1 ←R1 * R3
STORE	X, R1	M [X]← R1

- The load instructions transfer the operands from memory to CPU register.
- Add and multiply operations are executed with data in the registers without accessing memory.
- The result of the computations is then stored in memory with a store instruction.

5. Write a note on different Addressing Modes.

The general addressing modes supported by the computer processor are as follows:

1) **Implied mode:**

- In this mode the operands are specified implicitly in the definition of the definition of the instruction. For example, the instruction "complement accumulator" is an implied-mode

instruction because the operand in the accumulator is an implied mode instruction because the operand in the accumulator register is implied in the definition of the instruction.

- In fact all register later register is implied in the definition of the instruction. In fact, all register reference instructions that use an accumulator are implied mode instructions.

2) Immediate Mode:

- In this mode the operand is specified in the instruction itself. In other words, an immediate-mode instruction has an operand field rather than an address field.
- The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction.
- Immediate mode of instructions is useful for initializing register to constant value.

3) Register Mode:

- In this mode the operands are in registers that within the CPU. The particular register is selected from a register field in the instruction.
- A k-bit field can specify any one of 2^k registers.

4) Register Indirect Mode:

- In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory.
- Before using a register indirect mode instruction, the programmer must ensure that the memory address of the operand is placed in the processor register with a previous instruction.
- The advantage of this mode is that address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.

5) Autoincrement or Autodecrement Mode:

- This is similar to the register indirect mode expect that the register is incremented or decremented after (or before) its value is used to access memory.
- When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table. This can be achieved by using the increment or decrement instruction.

6) Direct Address Mode:

- In this mode the effective address is equal to the address part of the instruction. The operand resides in memory and its address is given directly by the address field of the instruction.

7) 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. The effective address in this mode is obtained from the following computational:

Effective address = address part of instruction + content of CPU register

8) Relative Address Mode:

- In this mode the content of the program counter is added to the address of the instruction in order to obtain the effective address. The address part of the instruction is usually a signed number which can be either positive or negative.
- When this number is added to the content of the program counter, the result produces an effective address whose position in memory is relative to the address of the next instruction.
- Relative addressing is often used with branch-type instruction when the branch address is in the area surrounding the instruction word itself.

9) Indexed Addressing Mode:

- In this mode the content of an index register is added to the address part of the instruction to obtain the effective address.
- The indexed register is a special CPU register that contain an index value. The address field of the instruction defines the beginning address of a data array in memory. Each operand in the array is stored in memory relative to the begging address.
- The distance between the beginning address and the address of the operand is the index value stored in the index register.

10) Base Register Addressing Mode:

- In this mode the content of a base register is added to the address part of the instruction to obtain the effective address.
- A base register is assumed to hold a base address and the address field of the instruction gives a displacement relative to this base address.
- The base register addressing mode is used in computers to facilitate the relocation of programs in memory.
- With a base register, the displacement values of instruction do not have to change. Only the value of the base register requires updating to reflect the beginning of a new memory segment.

6. Explain different types of instructions:

a) Data Transfer Instructions.

- Data transfer instructions move data from one place in the computer to another without changing the data content.
- The most common transfers are between memory and processor registers, between processor registers and input or output, and between the processor registers themselves.
- The ***load*** instruction has been used mostly to designate a transfer from memory to a processor register, usually an accumulator.
- The ***store*** instruction designates a transfer from a processor register into memory.
- The ***move*** instruction has been used in computers with multiple CPU registers to designate a

transfer from one register to another. It has also been used for data transfers between CPU registers and memory or between two memory words.

- The ***exchange*** instruction swaps information between two registers or a register and a memory word.
- The ***input and output*** instructions transfer data among processor registers and input or output terminals.
- The ***push and pop*** instructions transfer data between processor registers and a memory stack.

b) Arithmetic instructions.

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

c) Logical instructions.

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

d) Shift instructions.

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHR A
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC

Rotate left through carry	ROLC

7. What are status register bits? Draw and explain the block diagram showing all status registers.

- It is sometimes convenient to supplement the ALU circuit in the CPU with a status register where status bit conditions be stored for further analysis. Status bits are also called condition-code bits or flag bits.
- Figure 5.3 shows the block diagram of an 8-bit ALU with a 4-bit status register. 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.

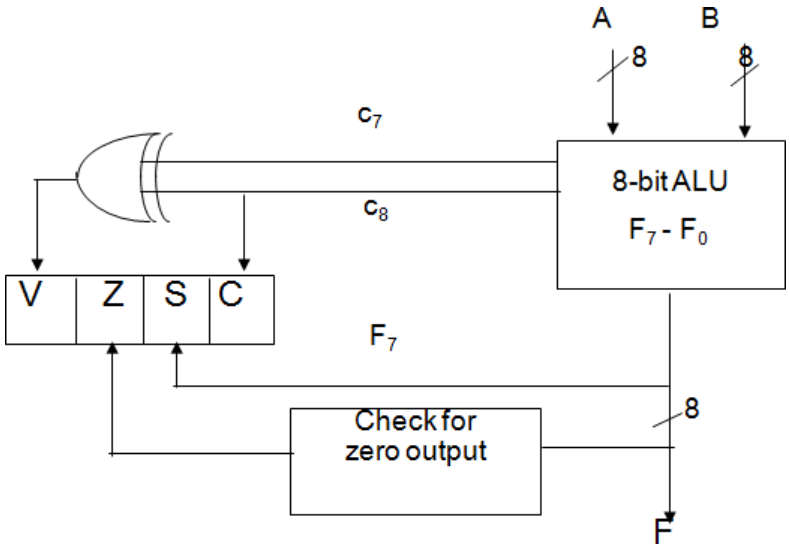


Figure 5.3: Status Register Bits

1. Bit C (carry) is set to 1 if the end carry C_8 is 1. It is cleared to 0 if the carry is 0.
2. Bit S (sign) is set to 1 if the highest-order bit F_7 is 1. It is set to 0 if set to 0 if the bit is 0.
3. Bit Z (zero) is set to 1 if the output of the ALU contains all 0's. it is cleared to 0 otherwise. In other words, Z = 1 if the output is zero and Z = 0 if the output is not zero.
4. Bit V (overflow) is set to 1 if the exclusives-OR of the last two carries is equal to 1, and cleared to 0 otherwise. This is the condition for an overflow when negative numbers are in 2's complement. For the 8-bit ALU, V = 1 if the output is greater than + 127 or less than -128.

- The status bits can be checked after an ALU operation to determine certain relationships that exist between the vales of A and B.
- If bit V is set after the addition of two signed numbers, it indicates an overflow condition.
- If Z is set after an exclusive-OR operation, it indicates that A = B.
- A single bit in A can be checked to determine if it is 0 or 1 by masking all bits except the bit in question and then checking the Z status bit.

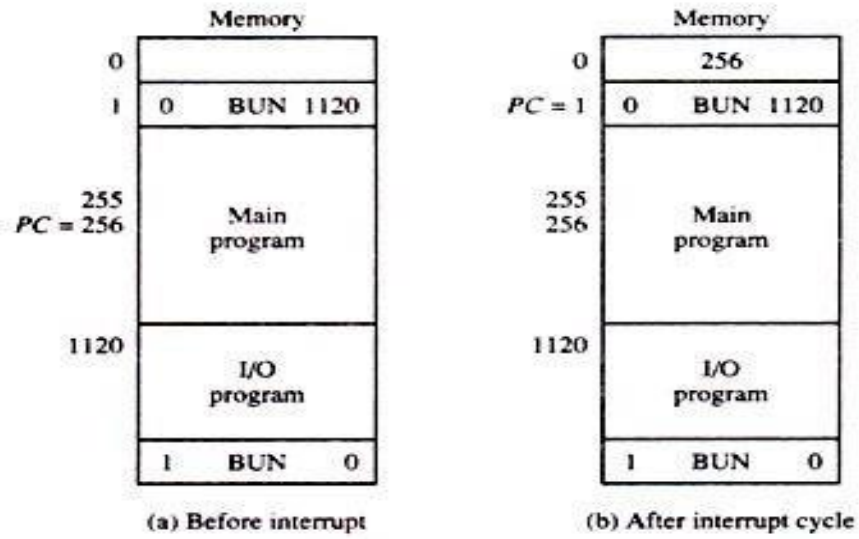
8. What is program interrupt? What happens when it comes? What are the tasks to be performed by service routine?

- The concept of program interrupt is used to handle a variety of problems that arise out of normal program sequence.
- Program interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internal generated request. Control returns to the original program after the service program is executed.
- After a program has been interrupted and the service routine been executed, the CPU must return to exactly the same state that it was when the interrupt occurred.
- Only if this happens will the interrupted program be able to resume exactly as if nothing had happened.
- The state of the CPU at the end of the execute cycle (when the interrupt is recognized) is determined from:

    1. The content of the program counter
    2. The content of all processor registers
    3. The content of certain status conditions

- The interrupt facility allows the running program to proceed until the input or output device sets its ready flag. Whenever a flag is set to 1, the computer completes the execution of the instruction in progress and then acknowledges the interrupt.
- The result of this action is that the retune address is stared in location 0. The instruction in location 1 is then performed; this initiates a service routine for the input or output transfer. The service routine can be stored in location 1.
- The service routine must have instructions to perform the following tasks:

    1. Save contents of processor registers.
    2. Check which flag is set.
    3. Service the device whose flag is set.
    4. Restore contents of processor registers.
    5. Turn the interrupt facility on.
    6. Return to the running program.



9. Explain various types of interrupts.

There are three major types of interrupts that cause a break in the normal execution of a program. They can be classified as:

1. External interrupts
2. Internal interrupts
3. Software interrupts

1) External interrupts:

- External interrupts come from input-output (I/0) devices, from a timing device, from a circuit monitoring the power supply, or from any other external source.
- Examples that cause external interrupts are I/0 device requesting transfer of data, I/o device finished transfer of data, elapsed time of an event, or power failure. Timeout interrupt may result from a program that is in an endless loop and thus exceeded its time allocation.
- Power failure interrupt may have as its service routine a program that transfers the complete state of the CPU into a nondestructive memory in the few milliseconds before power ceases.
- External interrupts are asynchronous. External interrupts depend on external conditions that are independent of the program being executed at the time.

2) Internal interrupts:

- Internal interrupts arise from illegal or erroneous use of an instruction or data. Internal interrupts are also called traps.
- Examples of interrupts caused by internal error conditions are register overflow, attempt to divide by zero, an invalid operation code, stack overflow, and protection violation. These error conditions usually occur as a result of a premature termination of the instruction execution. The service program that processes the internal interrupt determines the corrective measure to be taken.
- Internal interrupts are synchronous with the program. . If the program is rerun, the internal interrupts will occur in the same place each time.

3) Software interrupts:

- A software interrupt is a special call instruction that behaves like an interrupt rather than a subroutine call. It can be used by the programmer to initiate an interrupt procedure at any desired point in the program.
- The most common use of software interrupt is associated with a supervisor call instruction. This instruction provides means for switching from a CPU user mode to the supervisor mode. Certain operations in the computer may be assigned to the supervisor mode only, as for example, a complex input or output transfer procedure. A program written by a user must run in the user mode.
- When an input or output transfer is required, the supervisor mode is requested by means of a supervisor call instruction. This instruction causes a software interrupt that stores the old CPU state and brings in a new PSW that belongs to the supervisor mode.
- The calling program must pass information to the operating system in order to specify the particular task requested.

10. What do you understand by Reduced Instruction Set Computers? What are Complex Instruction Set Computers? List important characteristics of CISC and RISC computers.

Characteristics of RISC:

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 microprogrammed control
8. A relatively large number of registers in the processor unit
9. Use of overlapped register windows to speed-up procedure call and return
10. Efficient instruction pipeline
11. Compiler support for efficient translation of high-level language programs into machine language programs

Characteristics of CISC:

1. A larger number of instructions – typically from 100 to 250 instructions
2. Some instructions that perform specialized tasks and are used infrequently
3. A large variety of addressing modes – typically from 5 to 20 different modes
4. Variable-length instruction formats
5. Instructions that manipulate operands in memory

11. Explain Reverse Polish Notation (RPN) with appropriate example.

- The postfix RPN notation, referred to as Reverse Polish Notation (RPN), places the operator after the operands.
- The following examples demonstrate the three representations:

A + B	Infix notation
+ A B	Prefix or Polish notation
A B +	Postfix or reverse Polish notation

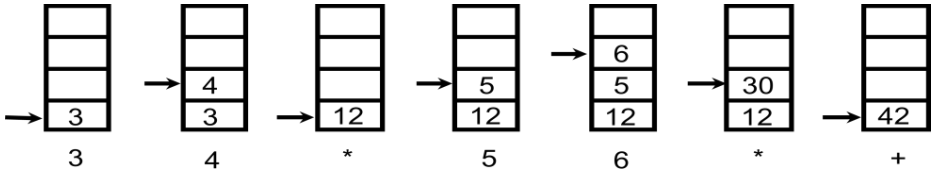
- The reverse Polish notation is in a form suitable for stack manipulation. The expression

  A * B + C * D is written in reverse Polish notation as A B * C D * +
- The conversion from infix notation to reverse Polish notation must take into consideration the operational hierarchy adopted for infix notation.
- This hierarchy dictates that we first perform all arithmetic inside inner parentheses, then inside outer parentheses, and do multiplication and division operations before addition and subtraction operations.

Evaluation of Arithmetic Expressions

- Any arithmetic expression can be expressed in parenthesis-free Polish notation, including reverse Polish notation

(3 * 4) + (5 * 6) ⇒3 4 * 5 6 * +



12. Explain Flynn's classification for computers.

Flynn's classification

- It is based on the multiplicity of Instruction Streams and Data Streams

  Instruction Stream: Sequence of Instructions read from memory

  Data Stream: Operations performed on the data in the processor

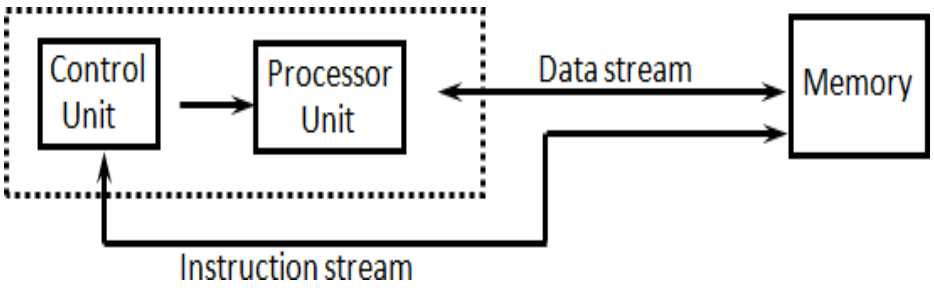
		Number of Data Streams	
		Single	**Multiple**
Number of Instruction Streams	**Single**	SISD	SIMD
	Multiple	MISD	MIMD

Figure 6.1: Flynn's Classification

SISD:

- Single instruction stream, single data stream.
- 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.

Figure 6.2: SISD Organization



SIMD:

- 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

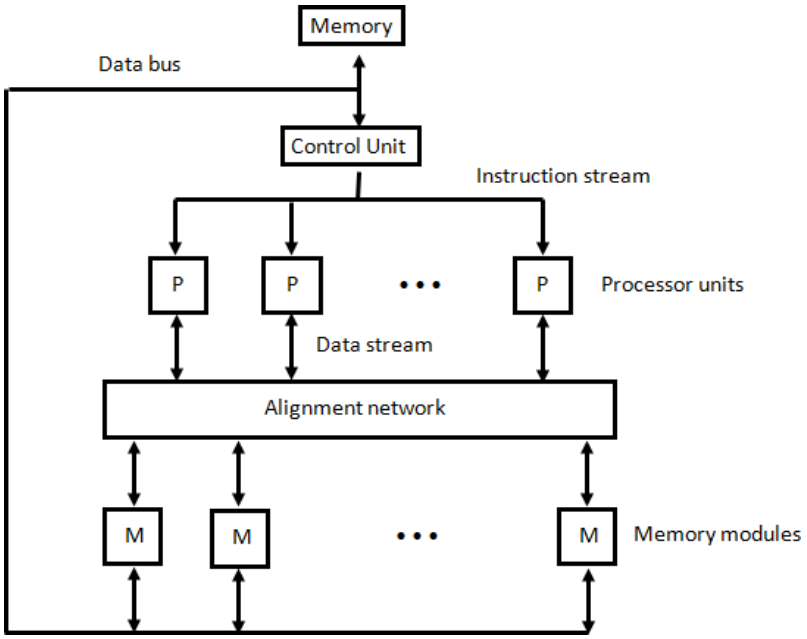


Figure 6.3: SIMD Organization

MISD:

- There is no computer at present that can be classified as MISD.
- MISD structure is only of theoretical interest since no practical system has been constructed using this organization.

MIMD:

- 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.
- Contains multiple processing units.
- Execution of multiple instructions on multiple data.

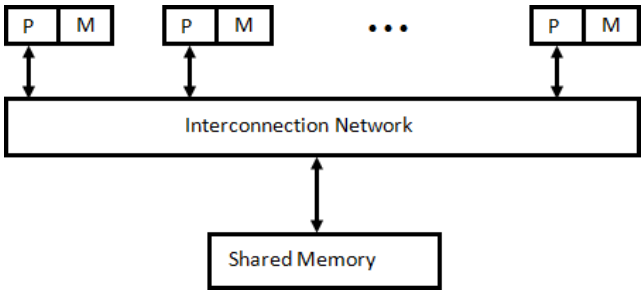


Figure 6.4: MIMD Organization

13. Explain pipelining technique. Draw the general structure of four segment pipeline.

- Pipeline 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.
- A pipeline can be visualized as a collection of processing segments through which binary information flows.
- Each segment performs partial processing dictated by the way the task is partitioned.
- The result obtained from the computation in each segment is transferred to the next segment in the pipeline.
- It is characteristic of pipelines that several computations can be in progress in distinct segments at the same time.
- The overlapping of computation is made possible by associating a register with each segment in the pipeline.
- The registers provide isolation between each segment so that each can operate on distinct data simultaneously.
- Any operation that can be decomposed into a sequence of sub operations of about the same complexity can be implemented by a pipeline processor.
- The technique is efficient for those applications that need to repeat the same task many times with different sets of data.
- The general structure of a four-segment pipeline is illustrated in Figure 6.5.

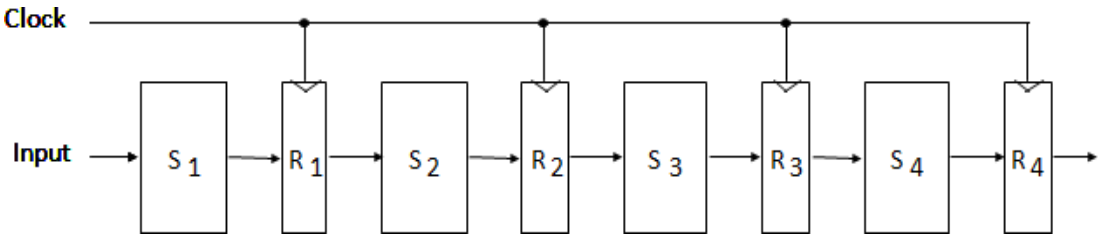


Figure 6.5: General Structure of Four-Segment Pipeline

- The operands pass through all four segments in a fixed sequence.
- Each segment consists of a combinational circuit S, which performs a sub operation over the data stream flowing through the pipe.
- The segments are separated by registers R, which hold the intermediate results between the stages.
- Information flows between adjacent stages under the control of a common clock applied to all the registers simultaneously.
- We define a task as the total operation performed going through all the segments in the pipeline.

14. Draw and explain Arithmetic Pipeline.

- The inputs to the floating-point adder pipeline are two normalized floating-point binary numbers.

$$X = A \times 2^a$$
$$Y = B \times 2^b$$

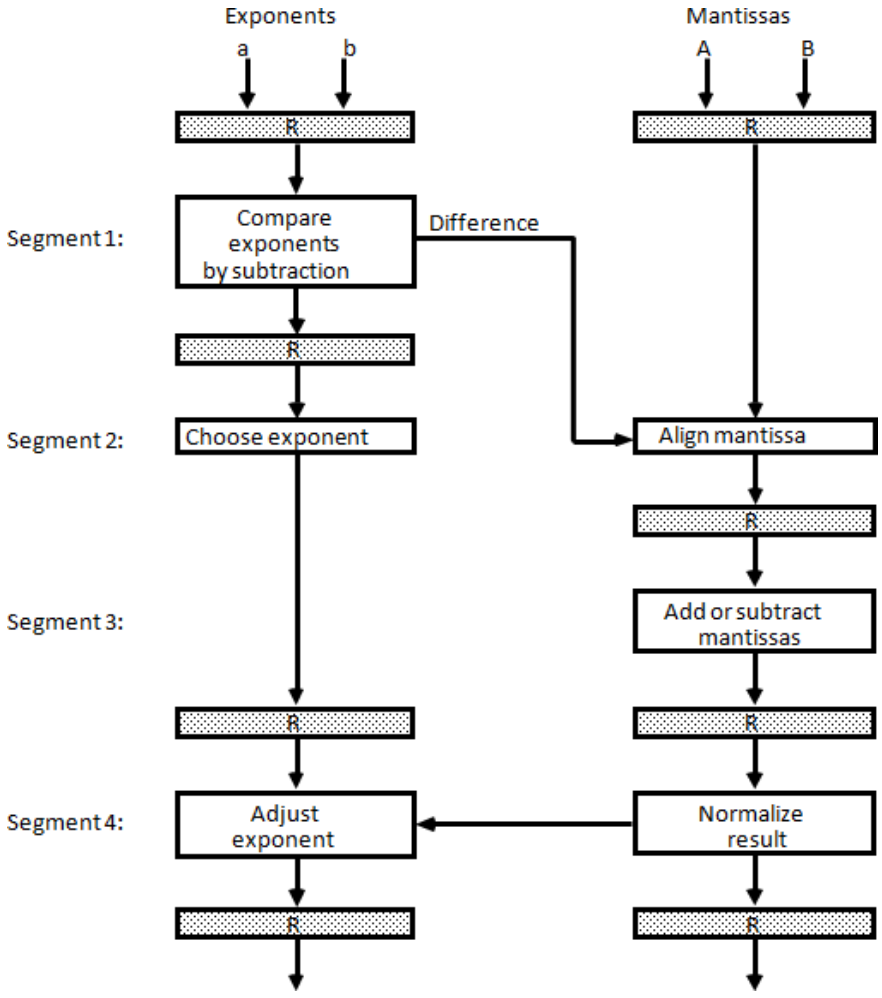


Figure 6.6: Pipeline for floating-point addition and subtraction

- 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, as shown in Figure 6.6.
- The registers labeled R are placed between the segments to store intermediate results.
- 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 following numerical example may clarify the sub-operations performed in each

segment.

- For simplicity, we use decimal numbers, although Figure 6.6 refers to binary numbers.
- Consider the two normalized floating-point numbers: $X = 0.9504 \times 10^3$
  $Y = 0.8200 \times 10^2$
- The two exponents are subtracted in the first segment to obtain 3 - 2 = 1.
- The larger exponent 3 is chosen as the exponent of the result.
- The next segment shifts the mantissa of Y to the right to obtain $X = 0.9504 \times 10^3$
  $Y = 0.0820 \times 10^3$
- This aligns the two mantissas under the same exponent. The addition of the two mantissas in segment 3 produces the sum
  $Z = 1.0324 \times 10^3$

15. Explain the Instruction Pipelining with example.

- Pipeline processing can occur in data stream as well as in instruction stream.
- An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments.
- This causes the instruction fetch and executes phases to overlap and perform simultaneous operations.
- One possible digression associated with such a scheme is that an instruction may cause a branch out of sequence.
- In that case the pipeline must be emptied and all the instructions that have been read from memory after the branch instruction must be discarded.
- Consider a 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.
- The buffer acts as a queue from which control then extracts the instructions for the execution unit.

Instruction cycle:

- The fetch and execute to process an instruction completely.
- 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.

- The design of an instruction pipeline will be most efficient if the instruction cycle is divided into segments of equal duration.
- The time that each step takes to fulfill its function depends on the instruction and the way it is executed.

Example: Four-Segment Instruction Pipeline

- Assume that the decoding of the instruction can be combined with the calculation of the effective address into one segment.
- Assume further that most of the instructions place the result into a processor registers so that the instruction execution and storing of the result can be combined into one segment.
- This reduces the instruction pipeline into four segments.
    1. FI: Fetch an instruction from memory
    2. DA: Decode the instruction and calculate the effective address of the operand
    3. FO: Fetch the operand
    4. EX: Execute the operation
- Figure 6.7 shows, how the instruction cycle in the CPU can be processed with a four-segment pipeline.

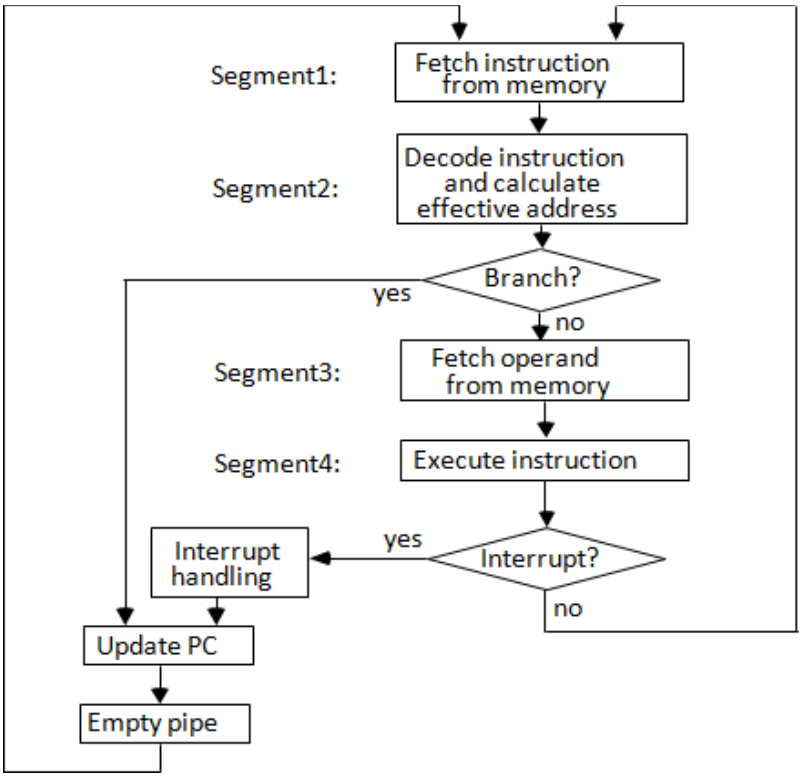


Figure 6.7: Four-segment CPU pipeline

- While an instruction is being executed in segment 4, the next instruction in sequence is busy fetching an operand from memory in segment 3.
- The effective address may be calculated in a separate arithmetic circuit for the third

instruction, and whenever the memory is available, the fourth and all subsequent instructions can be fetched and placed in an instruction FIFO.

- Thus up to four sub operations in the instruction cycle can overlap and up to four different instructions can be in progress of being processed at the same time.
- Figure 6.8 shows the operation of the instruction pipeline. The time in the horizontal axis is divided into steps of equal duration. The four segments are represented in the diagram with an abbreviated symbol.

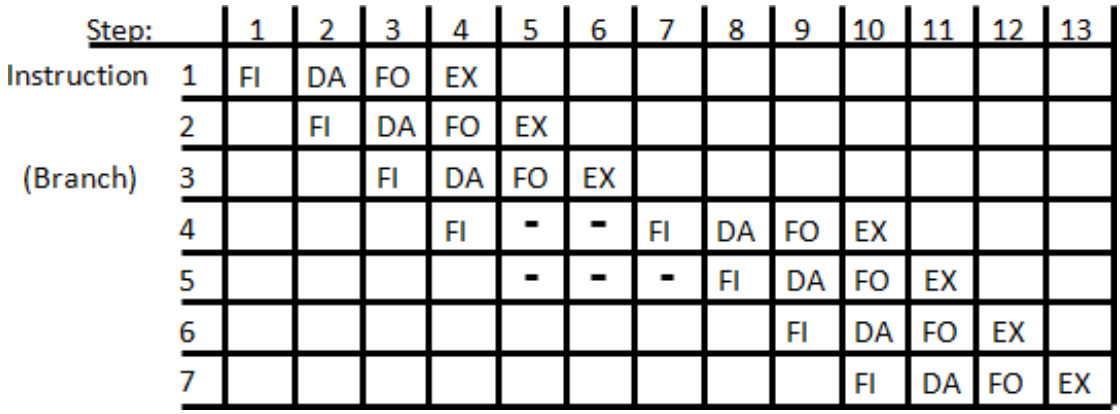


Figure 6.8: Timing of Instruction Pipeline

- It is assumed that the processor has separate instruction and data memories so that the operation in FI and FO can proceed at the same time.
- Thus, in step 4, instruction 1 is being executed in segment EX; the operand for instruction 2 is being fetched in segment FO; instruction 3 is being decoded in segment DA; and instruction 4 is being fetched from memory in segment FI.
- Assume now that instruction 3 is a branch instruction.
- As soon as this instruction is decoded in segment DA in step 4, the transfer from FI to DA of the other instructions is halted until the branch instruction is executed in step 6.
- If the branch is taken, a new instruction is fetched in step 7. If the branch is not taken, the instruction fetched previously in step 4 can be used.
- The pipeline then continues until a new branch instruction is encountered.
- Another delay may occur in the pipeline if the EX segment needs to store the result of the operation in the data memory while the FO segment needs to fetch an operand.
- In that case, segment FO must wait until segment EX has finished its operation.

16. What is pipeline conflict? Explain data dependency and handling of branch instruction in detail.

Pipeline conflict:

There are three major difficulties that cause the instruction pipeline conflicts.

- Resource conflicts caused by access to memory by two segments at the same time.

- Data dependency conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.
- Branch difficulties arise from branch and other instructions that change the value of PC.

Data Dependency:

- A collision occurs when an instruction cannot proceed because previous instructions did not complete certain operations.
- A data dependency occurs when an instruction needs data that are not yet available.
- Similarly, an address dependency may occur when an operand address cannot be calculated because the information needed by the addressing mode is not available.
- Pipelined computers deal with such conflicts between data dependencies in a variety of ways as follows.

Hardware interlocks:

- An interlock is a circuit that detects instructions whose source operands are destinations of instructions farther up in the pipeline.
- Detection of this situation causes the instruction whose source is not available to be delayed by enough clock cycles to resolve the conflict.
- This approach maintains the program sequence by using hardware to insert the required delays.

Operand forwarding:

- It uses special hardware to detect a conflict and then avoid it by routing the data through special paths between pipeline segments.
- This method requires additional hardware paths through multiplexers as well as the circuit that detects the conflict.

Delayed load:

- Sometimes compiler has the responsibility for solving data conflicts problems.
- The compiler for such computers is designed to detect a data conflict and reorder the instructions as necessary to delay the loading of the conflicting data by inserting no- operation instruction, this method is called delayed load.

Handling of Branch Instructions*:*

- One of the major problems in operating an instruction pipeline is the occurrence of branch instructions.
- A branch instruction can be conditional or unconditional.
- The branch instruction breaks the normal sequence of the instruction stream, causing difficulties in the operation of the instruction pipeline.
- Various hardware techniques are available to minimize the performance degradation caused by instruction branching.

Pre-fetch target:

- One way of handling a conditional branch is to prefetch the target instruction in addition to

the instruction following the branch.

- If the branch condition is successful, the pipeline continues from the branch target instruction.
- An extension of this procedure is to continue fetching instructions from both places until the branch decision is made.

Branch target buffer:

- Another possibility is the use of a branch target buffer or BTB.
- The BTB is an associative memory included in the fetch segment of the pipeline.
- Each entry in the BTB consists of the address of a previously executed branch instruction and the target instruction for that branch.
- It also stores the next few instructions after the branch target instruction.
- The advantage of this scheme is that branch instructions that have occurred previously are readily available in the pipeline without interruption.

Loop buffer:

- A variation of the BTB is the loop buffer. This is a small very high speed register file maintained by the instruction fetch segment of the pipeline.
- When a program loop is detected in the program, it is stored in the loop buffer in its entirety, including all branches.

Branch Prediction:

- A pipeline with branch prediction uses some additional logic to guess the outcome of a conditional branch instruction before it is executed.
- The pipeline then begins pre-fetching the instruction stream from the predicted path.
- A correct prediction eliminates the wasted time caused by branch penalties

Delayed branch:

- A procedure employed in most RISC processors is the delayed branch.
- In this procedure, the compiler detects the branch instructions and rearranges the machine language code sequence by inserting useful instructions that keep the pipeline operating without interruptions.

17. Explain (i) Vector Processing (ii) Vector Operations.

Vector Processing

- There is a class of computational problems that are beyond the capabilities of a conventional computer.
- These problems are characterized by the fact that they require a vast number of computations that will take a conventional computer days or even weeks to complete.
- In many science and engineering applications, the problems can be formulated in terms of vectors and matrices that lend themselves to vector processing.

Applications of Vector processing

- Long-range weather forecasting

- Petroleum explorations
- Seismic data analysis
- Medical diagnosis
- Aerodynamics and space flight simulations
- Artificial intelligence and expert systems
- Mapping the human genome
- Image processing

Vector Operations

- Many scientific problems require arithmetic operations on large arrays of numbers.
- These numbers are usually formulated as vectors and matrices of floating-point numbers.
- A vector is an ordered set of a one-dimensional array of data items.
- A vector V of length n is represented as a row vector by V = [V1 V2 V3 … Vn] that may be represented as a column vector if the data items are listed in a column.
- A conventional sequential computer is capable of processing operands one at a time.
- Consequently, operations on vectors must be broken down into single computations with subscripted variables.
- The element of vector V is written as V(I) and the index I refers to a memory address or register where the number is stored.

18. What is an array processor? Explain the different types of array processor.

- An array processor is a processor that performs computations on large arrays of data.
- The term is used to refer to two different types of processors, attached array processor and SIMD array processor.
- An attached array processor is an auxiliary processor attached to a general-purpose computer.
- It is intended to improve the performance of the host computer in specific numerical computation tasks.
- An SIMD array processor is a processor that has a single-instruction multiple-data organization.
- It manipulates vector instructions by means of multiple functional units responding to a common instruction.

Attached Array Processor

- An attached array processor is designed as a peripheral for a conventional host computer, and its purpose is to enhance the performance of the computer by providing vector processing for complex scientific applications.
- It achieves high performance by means of parallel processing with multiple functional units.

- It includes an arithmetic unit containing one or more pipelined floating-point adders and multipliers.
- The array processor can be programmed by the user to accommodate a variety of complex arithmetic problems.
- Figure 6.11 shows the interconnection of an attached array processor to host computer.

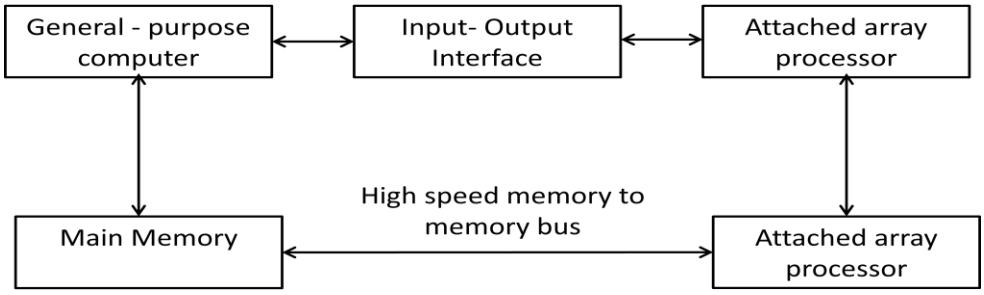


Figure 6.11: Attached array processor with host computer.

- The host computer is a general-purpose commercial computer and the attached processor is a back-end machine driven by the host computer.
- The array processor is connected through an input-output controller to the computer and the computer treats it like an external interface.
- The data for the attached processor are transferred from main memory to a local memory through a high-speed bus.
- The general-purpose computer without the attached processor serves the users that need conventional data processing.
- The system with the attached processor satisfies the needs for complex arithmetic applications.

SIMD Array Processor

- An SIMD array processor is a computer with multiple processing units operating in parallel.
- The processing units are synchronized to perform the same operation under the control of a common control unit, thus providing a single instruction stream, multiple data stream (SIMD) organization.
- A general block diagram of an array processor is shown in Figure 6.12

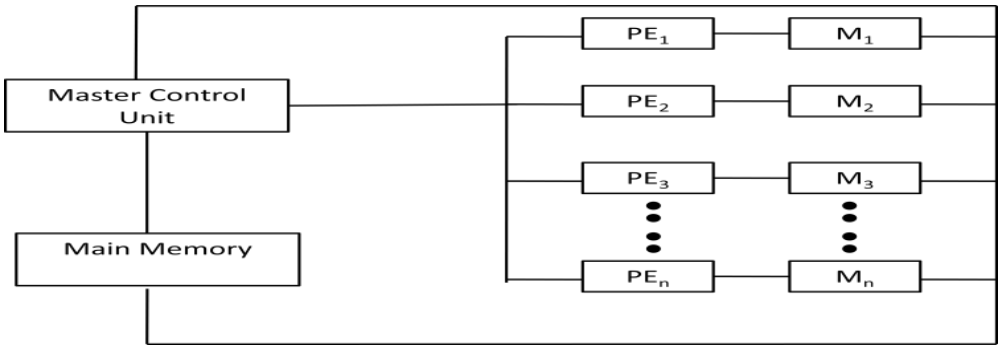


Figure 6.12: SIMD array processor organization

- It contains a set of identical processing elements (PEs), each having a local memory M.
- Each processor element includes an ALU, a floating-point arithmetic unit and working registers.
- The master control unit controls the operations in the processor elements.
- The main memory is used for storage of the program.
- The function of the master control unit is to decode the instructions and determine how the instruction is to be executed.
- Scalar and program control instructions are directly executed within the master control unit.
- Vector instructions are broadcast to all PEs simultaneously.
- Vector operands are distributed to the local memories prior to the parallel execution of the instruction.
- Masking schemes are used to control the status of each PE during the execution of vector instructions.
- Each PE has a flag that is set when the PE is active and reset when the PE is inactive.
- This ensures that only that PE'S that needs to participate is active during the execution of the instruction.
- SIMD processors are highly specialized computers.
- They are suited primarily for numerical problems that can be expressed in vector matrix form.
- However, they are not very efficient in other types of computations or in dealing with conventional data-processing programs.