MODULE 2

Principles of pipelining and vector processing - Linear pipelining - Classification of pipeline processors - General pipelines - Instruction and Arithmetic pipelines –Design of Pipelined instruction unit-Principles of Designing Pipeline Processors- Instruction prefetch and branch handling- Data Buffering and Busing Structure-Internal forwarding and register tagging-Hazard detection and Resolution Dynamic pipelines and Reconfigurability ,Architecture of Cray-1.

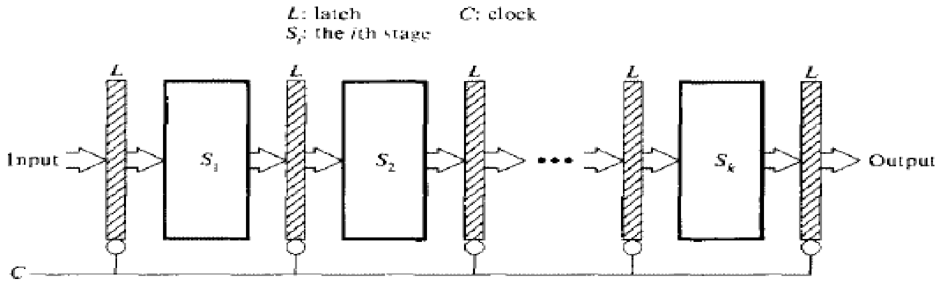
,

We have already seen those parallel computers are classified into pipeline computers, array processors and multiprocessor systems. This module deals with pipeline computers. Pipelining offers an economical way to realize temporal parallelism in digital computers. To achieve pipelining, one must subdivide the input task into a sequence of subtasks, each of which can be executed by a specialized hardware stage that operates concurrently with other stages in the pipeline.

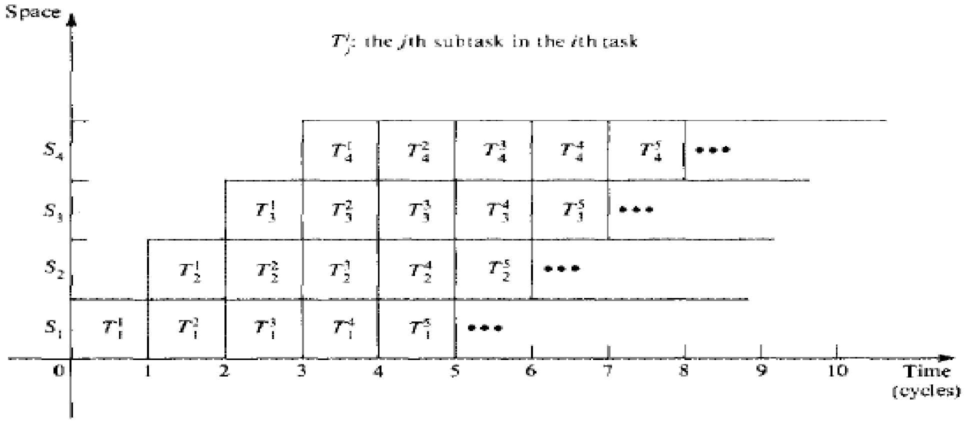
The concept of pipeline processing in a computer is similar to assembly lines in an industrial plant.To achieve pipelining ,we must subdivide the input task into a sequence of subtasks each of which can be executed by a specialized hardware stages that operates concurrently with other stages in the pipeline.That is, a pipeline computer performs overlapped computations

2.1 PRINCIPLES OF LINEAR PIPELINING

A basic linear pipeline processor is depicted in figure. The pipeline consists of a cascade of processing stages. The stages are pure combinational circuits performing arithmetic or logic operations over the data stream flowing through the pipe. The stages are separated by high speed interface latches. The latches are fast registers for holding the intermediate results between the stages. Information flows between adjacent stages are under the control of a common clock applied to all the latches simultaneously.



(a) Basic structure of a linear pipeline processor



(b) The space-time diagram depicting the overlapped operations

Clock period:-

The logic circuitry in each stage si has a time delay denoted bt τ_i.
Let τ_i be the time delay of each interface latch. The clock period of a linear pipeline is defined by
τ=max { τ_i.}^{k}1 + τ_i= τ_{m+} τ_i.

The reciprocal of the clock period is called the frequency f=1/ τ of a pipeline processor.

Speedup:-

We define the speedup of a K stage linear pipeline processor over an equivalent nonpipeline processor as

$$S_k = \frac{T_1}{T_k} = \frac{n.k}{k+(n-1)}$$

Maximum speed up that a linear pipeline can provide is k, where k is the number of stages in the pipe.This maximum speed up is never fully achievable because of data dependencies between instructions. Interrupts, program branches, and other factors to be revealed in later sections.

To understand the operational principles of pipeline computation, we illustrate the design of a pipeline floating point adder in figure. This pipeline is linearly constructed with four functional stges.The inputs to this pipeline are two normalized floating point numbers:

$A=a \times 2^p$
$B=b \times 2^q$

Where a and b are two fractions and p and q are their exponents, respectively.Our purpose is to compute the sum

$C=A+B=c \times 2^r=d \times 2^s$

Where r=max (p.q) and .5<d<1.operations performed in the 4 pipeline stages are specified below:

1. Compare the two exponents p and q to reveal the larger exponent r=max (p, q) and to determine their difference t=|p-q|
2. Shift right the fraction associated with the smaller exponent by t bits to equalize the two exponents before fraction addition.
3. Add the preshifted fraction with the other fraction to produce the intermediate sum fraction c, where 0<c<1

4 Count the number of leading zeros, say u,in fraction c and shift left c by u bits to produce the normalized fraction sum $d=c \times 2^u$,with a leading bit 1.Update the large exponent s by subtracting s=r-u to produce the o/p exponent.

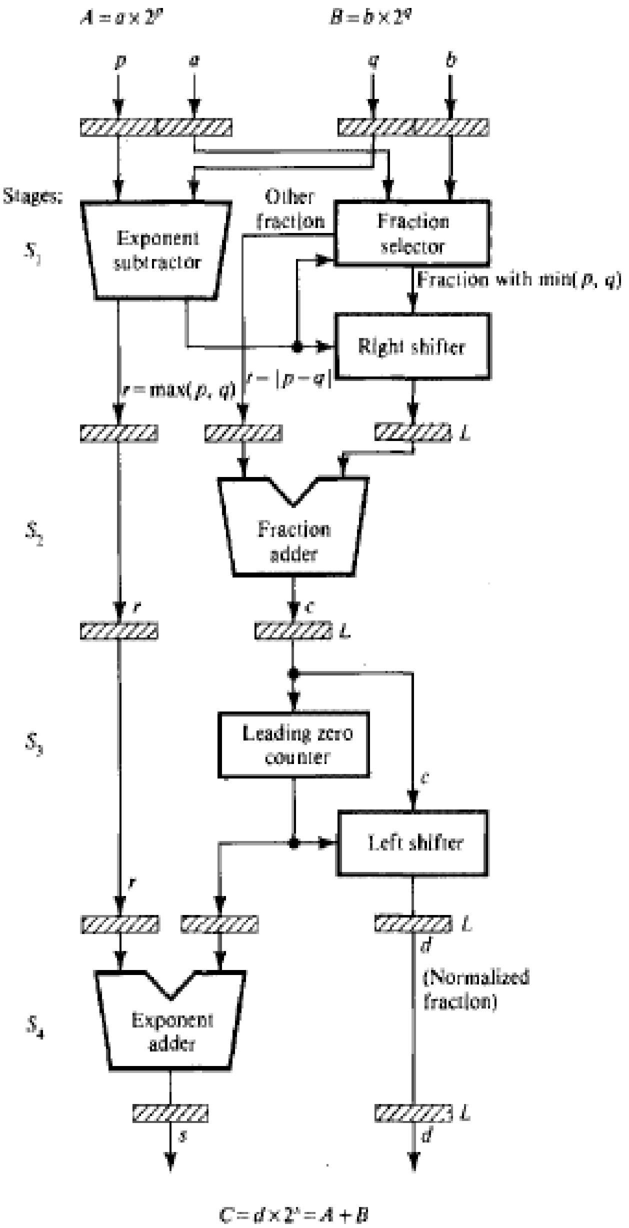


Figure 3.2 A pipelined floating-point adder with four processing stages.

Efficiency:-

The efficiency of a linear pipeline is measured by the percentage of busy time space spans over the total time space span, which equals the sum of all busy and idle time space spans.

Throughput :-

The number of tasks that can be completed by a pipeline per unit time is called its throughput. This rate reflects the computing power of a pipeline.

2.3. CLASSIFICATION OF PIPELINE PROCESSORS

According to the levels of processing, Handler has proposed the following classification scheme for pipeline processors, They are Arithmetic pielining, Instruction pipelining, Processor pipelining.

Arithmetic pipelining

The arithmetic logic units of a computer can be segmentized for pipeline operations in various data formats. Well known arithmetic pipe line examples are the upto 14 pipeline stages used in the Cray-1 and the upto 26 stages per pipe in the Cyber-205.

Instruction pipelining

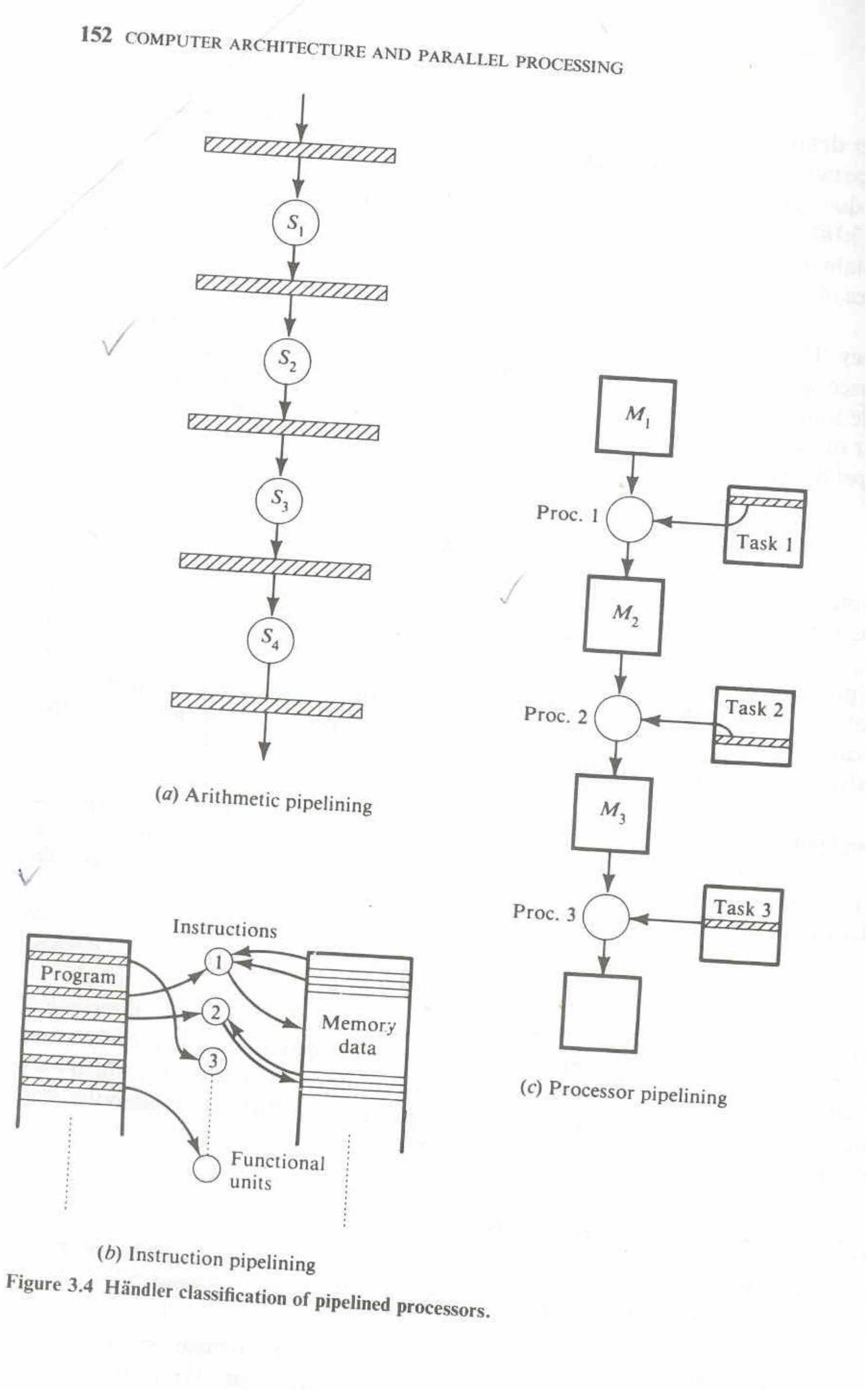
The execution of a stream of instructions can be pipelined by overlapping the execution of the current instruction with the fetch, decode and operand fetch of subsequent instructions.

Processor pipelining

This refers to the pipeline processing of the same data stream by a cascade of processors, each of which processes a specific task. The data stream passes the first processor with the result stored in memory block which is also accessible by the second processor. The second processor then passes the refined results to the third and so on.

.

152 COMPUTER ARCHITECTURE AND PARALLEL PROCESSING



(a) Arithmetic pipelining

(c) Processor pipelining

(b) Instruction pipelining

Figure 3.4 Händler classification of pipelined processors.

Based on configuration they are classified into following pipelines.

5.4) **<u>Static Vs dynamic pipelines</u>**

A static pipeline may assume only one functional configuration at a time.Staitic pipelines can be either unifunctional or multifunctional. A dynamic pipeline processor permits several functional configurations to exist simultaneously. In this sense a dynamic pipeline must be multifunctional. On the other hand a unifunctional pipe must be static.

5.5) **Unifunction Vs multifunction pipelines**

A pipeline unit with a fixed and dedicated function is called unifunctional.A multifunction pipe may perform different functions, either at different times or at the same timely interconnecting different subsets of stages in the pipeline.

5.6) **Scalar Vs vector pipelines**

Scalar pipeline process es a sequence of scalar operands under the control of a DO loop. Instructions in a small DO loop are often prefetched into the instruction buffer. The required scalar operands for repeated scalar instructions are moved into a data cache in order to continuously supply the pipeline with operands.

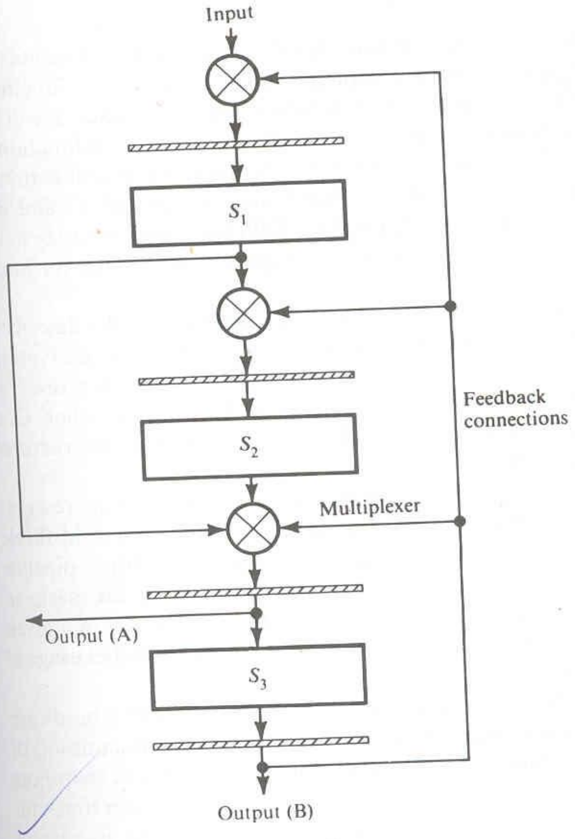
Vector pipelines are specially designed to handle vector instructions over vector operands. Computers having vector instructions are often called vector processors.

5.7) **General pipelines**

In some computations like linear recurrence the outputs of the pipeline are fed back as future inputs. In other words the inputs may depend on previous outputs. Pipelines with feedback may have a nonlinear flow of data. The timing of the feedback inputs becomes crucial to the nonlinear data flow.

Consider a sample pipeline that has a structure with both feed forward or feedback connections, in addition to the cascaded connection in linear pipeline. We use a two dimensional chart known as reservation table which is borrowed from the Gantt charts used in operation research to show how successive pipeline stages are utilized for a specific function evaluation in succeeive pipeline cycles.

The two reservation tables shown in figure correspond to the two functions of the sample pipeline. The rows correspond to pipeline stages and the columns to clock time units. The total number of clock units in the table is called the evaluation time for the given function.



(*a*) A sample pipeline

Time

	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7
S_1	A			A			A	
S_2		A						A
S_3			A		A	A		

(*b*) Reservation table for function A

	t_0	t_1	t_2	t_3	t_4	t_5	t_6
S_1	B				B		
S_2			B			B	
S_3		B		B			B

(*c*) Reservation table for function B

Figure 3.5 A sample pipeline and reservation tables for two of its functions.

Before studying various pipeline design techniques and examining vector processing requirements ,we need to understand how instructions can be overlapped ,executed and how repeated arithmetic computations can be done with pipelining. Instruction pipelining is illustrated with the designs in the IBM 360/91.Arithmetic pipelining will be studied in detail with multiple number addition, floating point addition, multiplication, and division.

Design of pipelined instruction units

The IBM 360/91 incorporates a high degree of pipelining in both instruction preprocessing and instruction execution. It is a 32 bit machine specially designed for scientific computations in either fixed point or floating point data formats.

A block diagram of the CPU in the IBM 360\91 is depicted in figure. It consists of 4 major parts. The main storage control unit, the instruction unit, the fixed point execution unit, and the floating point execution unit. The instruction unit is pipelined with a clock period of 60 ns.The cpu is designed to issue instructions atva burst rate of one instruction per clock cycle, and the performance of two execution units should support this rate. The storage control unit supervises information exchange between the cpu and main memory major functions of I unit, including instruction fetch, decode, and delivery to appropriate E unit, operand address calculation and operand fetch. The two E units are responsible for the fixed point and floating point arithmetic logic operations needed in the execution phase.

Concurrent arithmetic executions are facilitated in the model 91 by using two separate units for fixed point execution and floating point execution.
This permits instructions of two classes to be executed in parallel. As long as no cross unit data dependencies exist, the execution does not necessarily flow in the sequence in which the instructions are programmed. Within the floating point E unit are an add unit and multiply/divide unit which can operate in parallel.

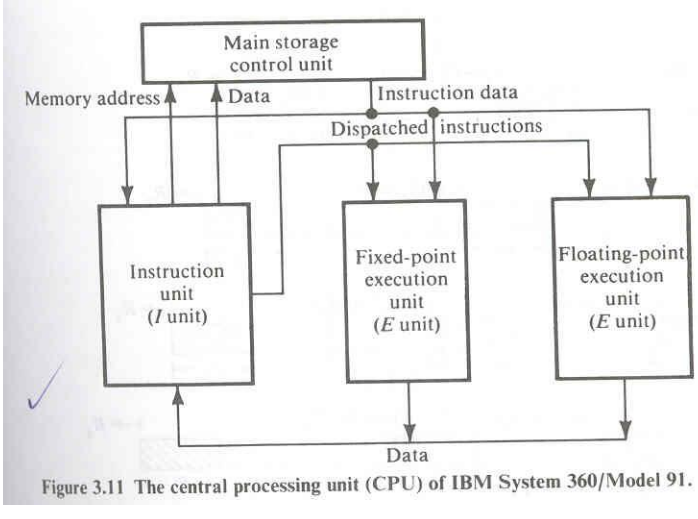
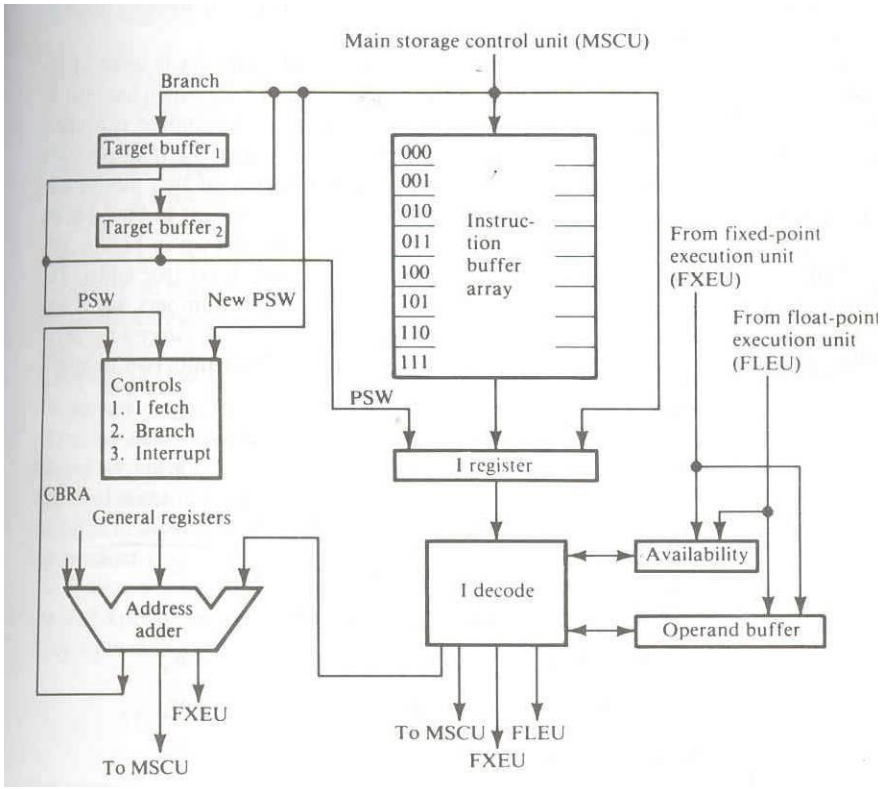

Main storage control unit
Memory address
Data
Instruction data
Dispatched instructions
Instruction unit (I unit)
Fixed-point execution unit (E unit)
Floating-point execution unit (E unit)
Data

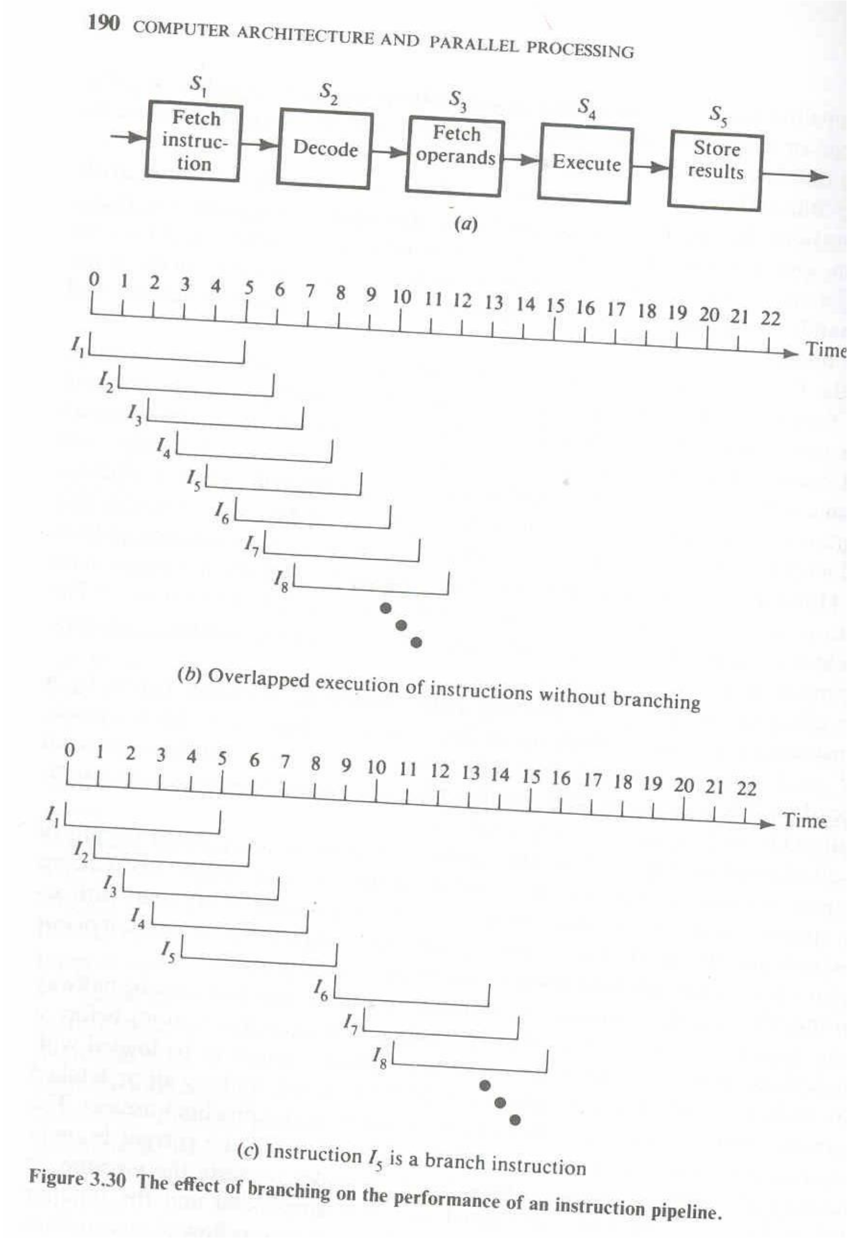
Figure 3.11 The central processing unit (CPU) of IBM System 360/Model 91.

The I unit in model 91 is specially designed to support the pipeline operations buffer is used to prefetch up to eight double words of instructions. A special controller is designed to handle instruction fetch, branch and interrupt conditions. There are two target buffers for branch handling. Sequential instruction fetch branch and interrupt handling are all built in hardware features. After decoding the I unit will dispatch the instructions to the fixed point E unit, the floating point E unit. For memory reference instructions the operand address is generated by an address adder. This adder is also used for branch address generation.



Instruction Prefetch and Branch handling

From the viewpoint of overlapped instruction execution sequencing for pipelined processing, the instruction mixes in typical computer programs can be classified into four types, as shown in Table. The arithmetic load operations constitute 60 percent of a typical computer program. These are mainly data manipulation operations which require one or two operand fetches. The execution of different arithmetic operations requires a different number of pipeline cycles. The store type operation does not require a fetch operand, but memory access is needed to store the data. The branch type operations correspond to an unconditional jump. There are two paths for a conditional operation .The yes path requires the calculation of the new address being branched to, whereas the no path proceeds to the next sequential instruction in the program.



190 COMPUTER ARCHITECTURE AND PARALLEL PROCESSING

(*a*)

(*b*) Overlapped execution of instructions without branching

(*c*) Instruction I_5 is a branch instruction

Figure 3.30 The effect of branching on the performance of an instruction pipeline.

Some functions like interrupt and branch produce damaging effect on the performance of pipeline computers. When instruction I is being executed the occurrence of interrupt postpone the execution of instruction I+1 until the interrupting request has been serviced.Gernerally there are two types of interrupts .Precise interrupts are caused by illegal operation codes found in instructions, which can be detected during the decoding stage. The other type, imprecise interrupts, is caused by defaults from storage, address, and execution functions.

For the Cray-1 computer the interrupt system is built around an exchange package. To change tasks, it is necessary to save the current processor state and to load a new processor state. The Cray-1 does this semi automatically when an interrupt occurs or when a program encounters an exit instruction. Under such circumstances the cray-1 saves the eight scalar registers, the eight address registers, the program counter, and the monitor flags. These are packed into 16 words are swapped with a block whose address is specified by a hardware exchange address register. However the exchange package does not contain all the hardware state information, so software interrupt handlers must save the rest of states. The rest includes 512 words of vector registers, 128 words of intermediate registers, a vector mask, and a real-time clock.

The effect of branching on pipeline performance is described below by a linear instruction pipeline consisting of five segments: instruction fetch, decode, operand fetch, execute, and store results. Possible memory conflicts between overlapped fetches are ignored and a sufficiently large cache memory is used in the following analysis.

As illustrated in figure the instruction pipeline executes a stream of instructions continuously in an overlapped fashion if branch type instructions do not appear. Under such circumstances once the pipeline is filled up with sequential instructions the pipeline completes the execution of one instruction per a fixed latency.

On the other hand a branch instruction entering the pipeline may be halfway down the pipe before a branch decision is made. This will cause the program counter to be loaded with the new address to which the program should be directed, making all prefetched instructions useless. The next instruction cannot be initiated until the completion of the current branch instruction cycle. The overlapped action is suspended and the pipeline must be drained at the end of the branch cycle. The continuous flow of instructions into the pipeline is thus temporarily interrupted because of the presence of a branch instruction.

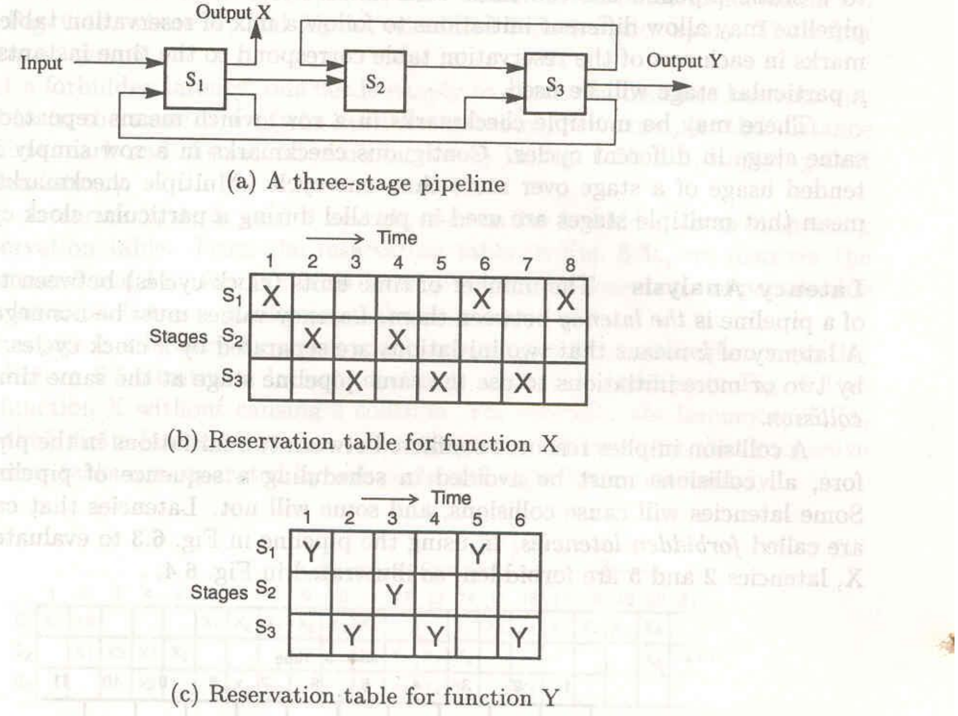
Dynamic pipelines

How to avoid collision in dynamic pipeline

Reservation table:

The reservation table is used to analyze how specific pipeline stages are utilized to evaluate a particular function. Reservation table for a dynamic pipeline becomes more interesting because a nonlinear pattern is followed.

Two reservation tables are given in figure.



(a) A three-stage pipeline

Time →

Stages	1	2	3	4	5	6	7	8
S_1	X					X		X
S_2		X		X				
S_3			X		X		X	

(b) Reservation table for function X

Time →

Stages	1	2	3	4	5	6
S_1	Y				Y	
S_2			Y			
S_3		Y		Y		Y

(c) Reservation table for function Y

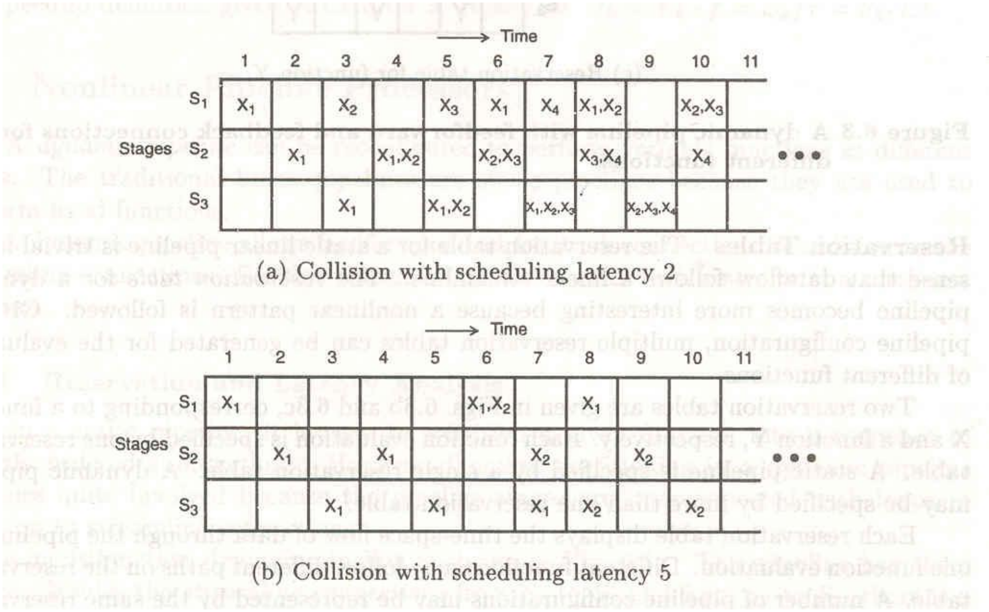
3 A dynamic pipeline with feedforward and feedback connections for two different functions.

Corresponding to a function X and Y.Each reservation table displays time space flow of data through pipeline for one function evaluation.

Latency analysis:

The number of time units between two initiations of a pipeline is the latency between them. A collision implies resource conflicts between two initiations in the pipeline. Therefore all collisions must be avoided in scheduling a sequence of pipeline initiations.

Some latency will cause collisions some will not. Latencies that cause collisions are called forbidden latencies. In using the the pipeline in above figure to evaluate the function X, latencies 2 and 5 are forbidden. as illustrated in fig:



Time →

Stages	1	2	3	4	5	6	7	8	9	10	11	
S_1	X_1		X_2		X_3	X_1	X_4	X_1,X_2		X_2,X_3		
S_2		X_1		X_1,X_2		X_2,X_3		X_3,X_4		X_4		• • •
S_3			X_1		X_1,X_2		X_1,X_2,X_3		X_2,X_3,X_4			

(a) Collision with scheduling latency 2

Time →

Stages	1	2	3	4	5	6	7	8	9	10	11	
S_1	X_1					X_1,X_2		X_1				
S_2		X_1		X_1			X_2		X_2			• • •
S_3			X_1		X_1		X_1	X_2		X_2		

(b) Collision with scheduling latency 5

The i^{th} initiations are denoted as Xi in fig. With latency 2,initiations X1 and X2 collide in stage 2 at time 4.At time 7,these initiations collide in stage 3.Similarly,other collisions are shown in times 5,6,8,…..etc.The collision patterns for latency 5 are shown In fig where X1 and X2 are scheduled 5 clock cycles apart. Their first collision occurs at time 6.

A latency cycle is a latency sequence which repeats the same subsequence indefinitely. Consider a latency cycle C.The set G_c of all possible time intervals between initiations derived from cycle C is called initiation interval set.

For Example G_c ={4,8,12,…}for c=(4)

Let G_c(mod p) be the set formed by taking mod p equivalents of all elements of set G_c.

G_c(mod 20)={4,8,12,16}

The compliment of set G_c equals Z- $G_c$where Z is the set of positive integers.

Compliment of G_c(mod p)=z(mod p)- G_c(mod p)

A latency cycle with period p and an initiation interval set Gc is allowable in a pipeline with forbidden latency set F if and only if

$F(\text{mod } p) \cap G_c(\text{mod } p)=\varnothing$

This means that there will be no collision if none of the initiation intervals equals a forbidden latency.

Collision free scheduling

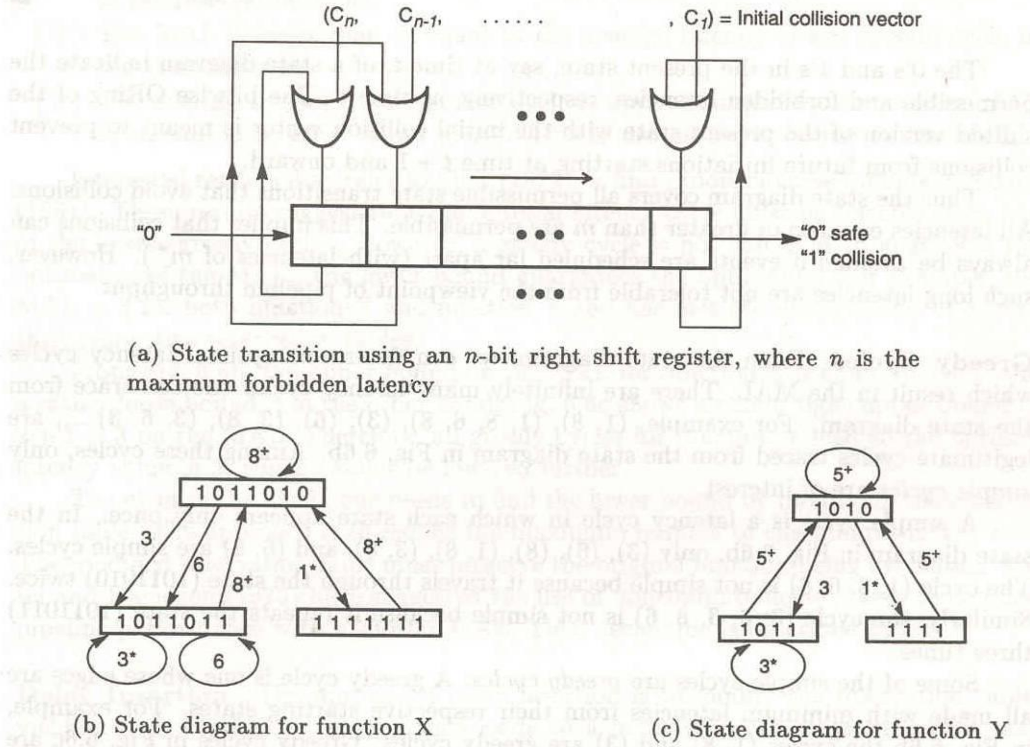
Collision vectors

The combined set of permissible and forbidden latencies can be displayed by a collision vector. Which is an m bit binary vector.$C=\{C_mC_{m-1}\ldots.C_2,C_1\}$.The value of C_i is 1 if latency causes a collision and zero if latency is permissible.

State diagrams

From the collision vector one can construct a state diagram.

The following figure explains Collision vector and state diagram.



(a) State transition using an n-bit right shift register, where n is the maximum forbidden latency

(b) State diagram for function X

(c) State diagram for function Y

Figure 6.6 Two state diagrams obtained from the two reservation tables in Fig. 6.3, respectively.

Greedy cycle

From the state diagram we can determine optimal latency cycle which result in the MAL(Minimum average latency).There are infinitely many latency cycles one can trace from the state diagram. For example(1,8),(1,8,6,8),(3),(6),(3,8),(3,6,3)…., are cycles traced from above state diagram. **A simple cycle** is a latency cycle in which each state appears only once. A greedy cycle is one whose edges are all made with minimum latencies from their respective starting states.

Pipeline Schedule optimization

An optimization technique based on the MAL is given below.

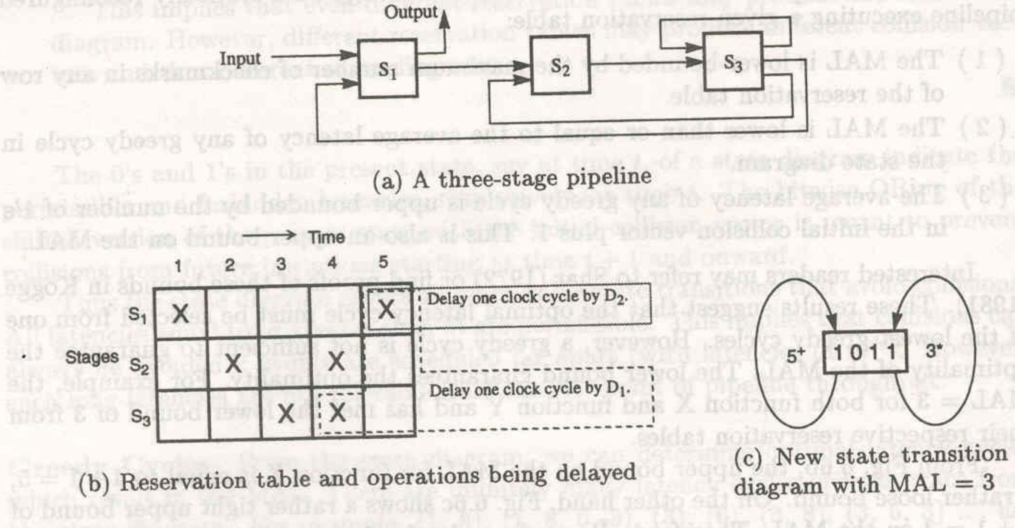
Bounds on Mal:

1. The MAL is lower bounded by maximum number of check marks in any row of the reservation table.

2. The Mal is lower than or equal to average latency of any greedy cyclein the state diagram.

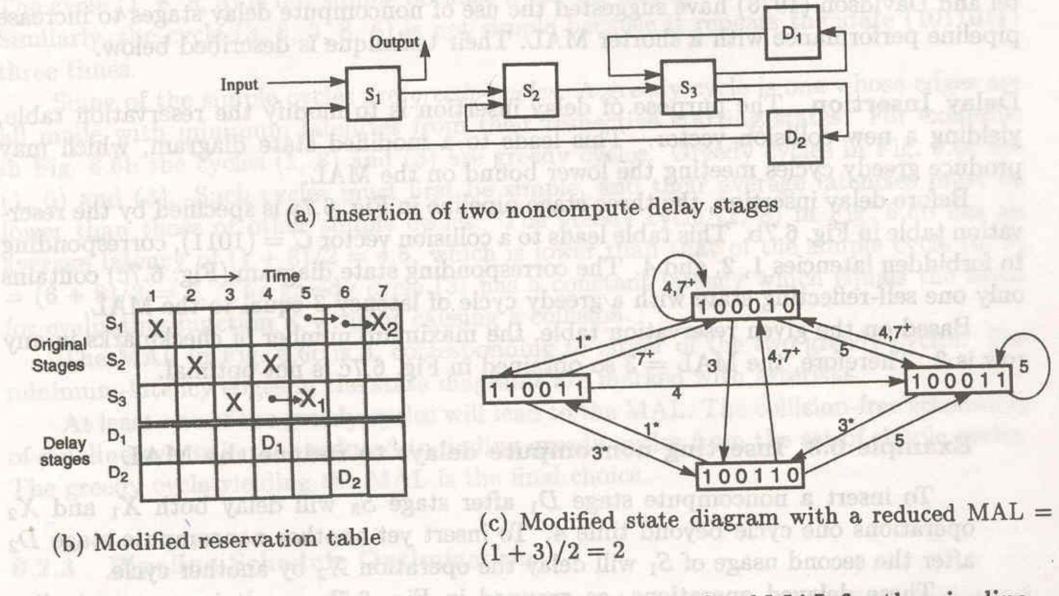
3. The average latency of any greedy cycleis upper bounded by the number of 1’s in initial collision vector plus one.

The following figure will explain how to insert delays to optimize pipeline schedule .


Output
Input
S_1
S_2
S_3
(a) A three-stage pipeline
Time
Stages
Delay one clock cycle by D_2.
Delay one clock cycle by D_1.
(b) Reservation table and operations being delayed
1011
5+
3*
(c) New state transition diagram with MAL = 3

	1	2	3	4	5
S_1	X				X
S_2		X		X	
S_3			X	X	

Figure 6.7 A pipeline with a minimum average latency of 3.


Output
Input
S_1
S_2
S_3
D_1
D_2
(a) Insertion of two noncompute delay stages
Time
Original Stages
Delay stages
(b) Modified reservation table
4,7+
100010
1*
7+
5
4,7+
3
4,7+
110011
4
100011
5
1*
3*
5
3*
100110
(c) Modified state diagram with a reduced MAL = (1 + 3)/2 = 2

	1	2	3	4	5	6	7
S_1	X					X_2	
S_2		X		X			
S_3			X		X_1		
D_1				D_1			
D_2						D_2	

Figure 6.8 Insertion of two delay stages to obtain an optimal MAL for the pipeline in Fig. 6.7.

The task arrivals in a pipeline processor may be periodic for a program with inner loops. If we assume that each task can only occupy one stage at a time, no parallel computation can be done within a single task. Such an assumption stems from the practical difficulties encountered in implementing a priority scheme involving parallel computations of a task. Once some buffers are provided internally the task scheduling problem can be greatly simplified. Whenever two or more tasks trying to use the same stage only one of the tasks allow using the stage, while the rest wait in the buffers according to some priority schemes.

There are two different implementations of internal buffers in a pipeline. The first use one buffer for each stage. And the second uses one buffer per computation step. For one buffer per stage two priority schemes FIFO global can be used. In the FIFI global scheme a task has priority overall tasks initiated later. In the LIFO global scheme a task has priority over all tasks initiated earlier. For one buffer per computation step multiple buffers may be used in each segment with the following priorites:MPF:most processed first;LPF:least processed first;LWRF:least work remaining first; and MWERF:most work remaining first.

A dynamic pipeline would allow several configurations to be simultaneously present. For example a dynamic pipeline arithmetic unit could perform addition and multiplication at the same time. Tremendous control overhead and increased interconnection complexity would be expected. None of the existing pipeline processors has achieved this dynamic capability. Most commercial pipelines are static. In T1-ASC the desired control allows different instructions to assume different data paths through the arithmetic pipeline at different times. All path control information is stored in a read only memory., which can be accessed at the initiation of an instruction.

Architecture of Cray-1.

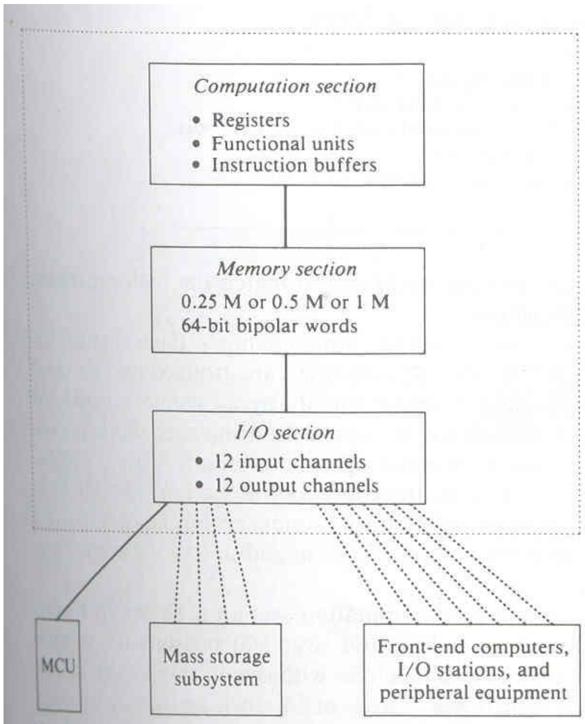


Figure 4.17 Front-end system interface and Cray-1 connections.

The Cray-1 has been available as the first modern vector processor since 1976.The architecture of Cray-1 consists of a number of working registers ,large instruction buffers and data buffers and 12 functional pipeline units. The clock rate in the Cray-1 is 12.5 ns.The Cray -1 is not a standalone computer. A front end host computer is required to serve as the system manager. A Data General Eclipse computer has been used as the front end, which is connected to the Cray-1 CPU via I/O channels. Figure shows the front end system interface and the Cray-1 memory and functional sections. The CPU contains a computation section, a memory section and an I/O section. Twenty four I/O channels are connected to the front end computer, the I/O stations ,peripheral equipment the mass storage subsystem and a maintenance control unit .The front end system will collect the data, present it to Cray-1 for distribution to slower deices.

The memory section in Cray-1 computer is organized in 8 or 16 banks with 72 modules per bank. Bipolar RAMs are used in the main memory with at most one million words of 72 bits each. Each memory module contributes 1 bit of a 72 bit word, out of which 8 bits re parity checks for single error correction and double error correction. The actual data word has only 64 bits.

The I/O section contains 12 input and 12 output channels. Each channel has a maximum transfer rate of 80 M bytes/s.The channels are grouped into six input groups and and are served equally by all memory banks. At most one 64 bit word can be transferred per channel during each clock period.

A functional block diagram of the computation section is shown in figure. It contains 64 x 4 instruction buffers and over 800 registers for various purposes. The 12 functional units are all pipelines with one to seven clock delays except for the reciprocal units, which has a delay of 14 clock periods. Arithmetic operations include 24 bit integer and 64 bit floating point computations. Large number of high speed registers contribute to the vector ans scalar processing capability of Cray-1.Without these many registers the functional units cannot operate with a clock rate of 12 ns.The three primary type of registers are address(A) scalar(S) and vector(V) .

There are eight address registers with 24 bits each used for memory addressing, indexing, shift counting, loop control an I/O channel addressing. There are 64 bit S registers serving as the storage of source and destination operands for the execution of scalar arithmetic and logic instructions. There are eight V registers, each has 64 component registers.ASll instructions either 16 or 32 bit long are first loaded from memory into one of 4 instruction buffers, each having 64 sixteen bit registers. The P register is a 22 bit program counter indicating the next parcel of program code to enter the next instruction parcel register in a linear program sequence.