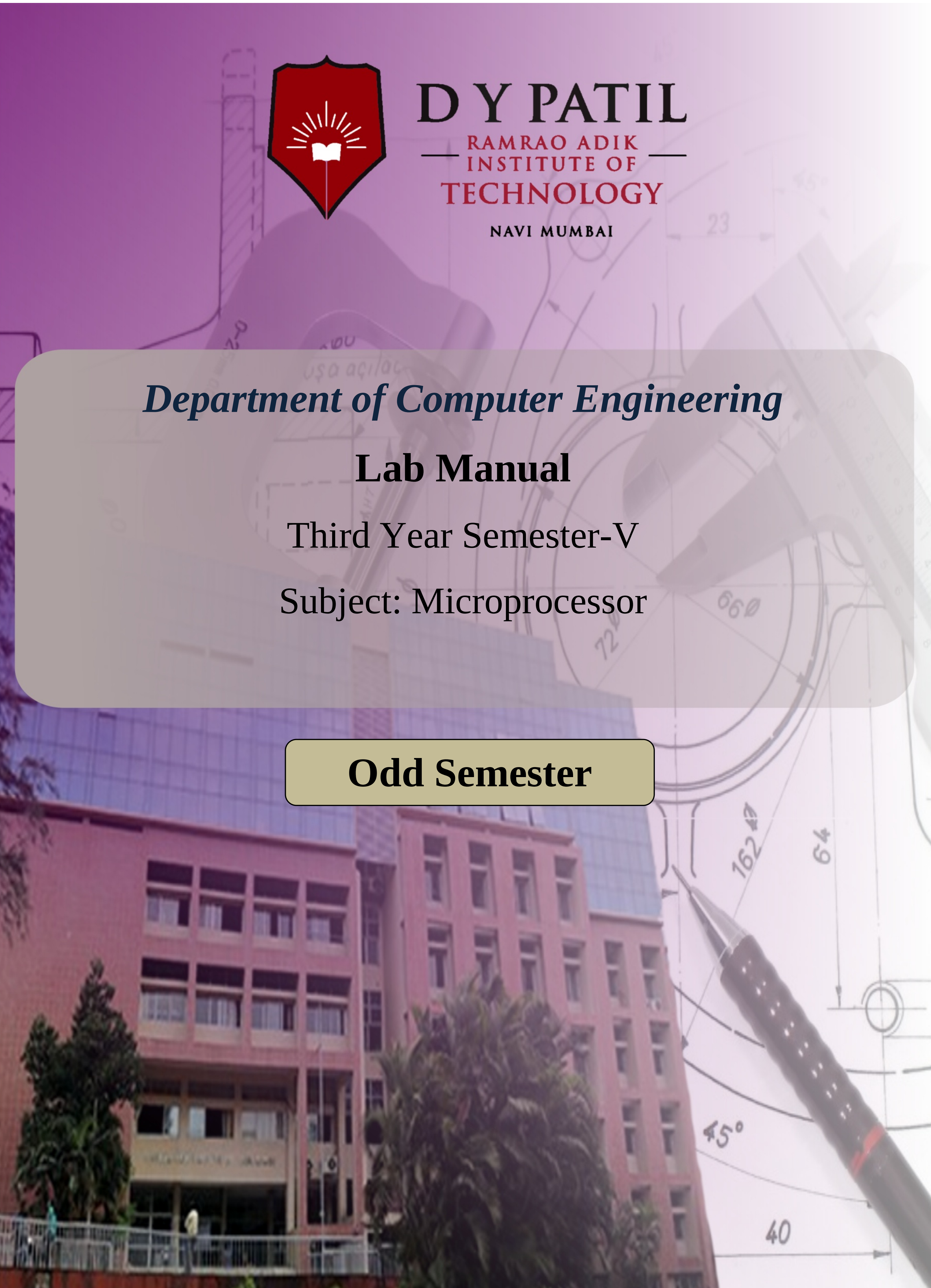


D Y PATIL
RAMRAO ADIK INSTITUTE OF TECHNOLOGY
NAVI MUMBAI

Department of Computer Engineering

Lab Manual

Third Year Semester-V

Subject: Microprocessor

Odd Semester

Institutional Vision, Mission and Quality Policy

Our Vision

To foster and permeate higher and quality education with value added engineering, technology programs, providing all facilities in terms of technology and platforms for all round development with societal awareness and nurture the youth with international competencies and exemplary level of employability even under highly competitive environment so that they are innovative adaptable and capable of handling problems faced by our country and world at large.

RAIT's firm belief in new form of engineering education that lays equal stress on academics and leadership building extracurricular skills has been a major contribution to the success of RAIT as one of the most reputed institution of higher learning. The challenges faced by our country and world in the 21 Century needs a whole new range of thought and action leaders, which a conventional educational system in engineering disciplines are ill equipped to produce. Our reputation in providing good engineering education with additional life skills ensure that high grade and highly motivated students join us. Our laboratories and practical sessions reflect the latest that is being followed in the Industry. The project works and summer projects make our students adept at handling the real life problems and be Industry ready. Our students are well placed in the Industry and their performance makes reputed companies visit us with renewed demands and vigour.

Our Mission

The Institution is committed to mobilize the resources and equip itself with men and materials of excellence thereby ensuring that the Institution becomes pivotal center of service to Industry, academia, and society with the latest technology. RAIT engages different platforms such as technology enhancing Student Technical Societies, Cultural platforms, Sports excellence centers, Entrepreneurial Development Center and Societal Interaction Cell. To develop the college to become an autonomous Institution & deemed university at the earliest with facilities for advanced research and development programs on par with international standards. To invite international and reputed national Institutions and Universities to collaborate with our institution on the issues of common interest of teaching and learning sophistication.

RAIT's Mission is to produce engineering and technology professionals who are innovative and inspiring thought leaders, adept at solving problems faced by our nation and world by providing quality education.

The Institute is working closely with all stake holders like industry, academia to foster knowledge generation, acquisition, dissemination using best available resources to address the great challenges being faced by our country and World. RAIT is fully dedicated to provide its students skills that make them leaders and solution providers and are Industry ready when they graduate from the Institution.

We at RAIT assure our main stakeholders of students 100% quality for the programmes we deliver. This quality assurance stems from the teaching and learning processes we have at work at our campus and the teachers who are handpicked from reputed institutions IIT/NIT/MU, etc. and they inspire the students to be innovative in thinking and practical in approach. We have installed internal procedures to better skills set of instructors by sending them to training courses, workshops, seminars and conferences. We have also a full fledged course curriculum and deliveries planned in advance for a structured semester long programme. We have well developed feedback system employers, alumni, students and parents from to fine tune Learning and Teaching processes. These tools help us to ensure same quality of teaching independent of any individual instructor. Each classroom is equipped with Internet and other digital learning resources.

The effective learning process in the campus comprises a clean and stimulating classroom environment and availability of lecture notes and digital resources prepared by instructor from the comfort of home. In addition student is provided with good number of assignments that would trigger his thinking process. The testing process involves an objective test paper that would gauge the understanding of concepts by the students. The quality assurance process also ensures that the learning process is effective. The summer internships and project work based training ensure learning process to include practical and industry relevant aspects. Various technical events, seminars and conferences make the student learning complete.

Our Quality Policy

ज्ञानधीनं जगत् सर्वम।

Knowledge is supreme.

Our Quality Policy

It is our earnest endeavour to produce high quality engineering professionals who are innovative and inspiring, thought and action leaders, competent to solve problems faced by society, nation and world at large by striving towards very high standards in learning, teaching and training methodologies.

Our Motto: If it is not of quality, it is NOT RAIT!

Departmental Vision, Mission

Vision

To impart higher and quality education in computer science with value added engineering and technology programs to prepare technically sound, ethically strong engineers with social awareness. To extend the facilities, to meet the fast changing requirements and nurture the youths with international competencies and exemplary level of employability and research under highly competitive environments.

Mission

To mobilize the resources and equip the institution with men and materials of excellence to provide knowledge and develop technologies in the thrust areas of computer science and Engineering. To provide the diverse platforms of sports, technical, cocurricular and extracurricular activities for the overall development of student with ethical attitude. To prepare the students to sustain the impact of computer education for social needs encompassing industry, educational institutions and public service. To collaborate with IITs, reputed universities and industries for the technical and overall upliftment of students for continuing learning and entrepreneurship.

Departmental Program Educational Objectives (PEOs)

1. **Learn and Integrate**

   To provide Computer Engineering students with a strong foundation in the mathematical, scientific and engineering fundamentals necessary to formulate, solve and analyze engineering problems and to prepare them for graduate studies.

2. **Think and Create**

   To develop an ability to analyze the requirements of the software and hardware, understand the technical specifications, create a model, design, implement and verify a computing system to meet specified requirements while considering real-world constraints to solve real world problems.

3. **Broad Base**

   To provide broad education necessary to understand the science of computer engineering and the impact of it in a global and social context.

4. **Techno-leader**

   To provide exposure to emerging cutting edge technologies, adequate training & opportunities to work as teams on multidisciplinary projects with effective communication skills and leadership qualities.

5. **Practice citizenship**

   To provide knowledge of professional and ethical responsibility and to contribute to society through active engagement with professional societies, schools, civic organizations or other community activities.

6. **Clarify Purpose and Perspective**

   To provide strong in-depth education through electives and to promote student awareness on the life-long learning to adapt to innovation and change, and to be successful in their professional work or graduate studies.

Departmental Program Outcomes (POs)

Pa. **Foundation of computing -** An ability to apply knowledge of computing, applied mathematics, and fundamental engineering concepts appropriate to the discipline.

Pb. **Experiments & Data Analysis -** An ability to understand, identify, analyze and design the problem, implement and validate the solution including both hardware and software.

Pc. **Current Computing Techniques** – An ability to use current techniques, skills, and tools necessary for computing practice .

Pd. **Teamwork** – An ability to have leadership and management skills to accomplish a common goal.

Pe. **Engineering Problems -** an ability to identify, formulates, and s olve engineering problems.

Pf. **Professional Ethics** – An understanding of professional, ethical, legal, security and social issues and responsibilities.

Pg. **Communication** – An ability to communicate effectively with a range of audiences in both verbal and written form.

Ph. **Impact of Technology** – An ability to analyze the local and global impact of computing on individuals, organizations, and society.

Pi. **Life-long learning** – An ability to recognize the need for, and an ability to engage in life-long learning.

Pj. **Contemporary Issues** – An ability to exploit gained skills and knowledge of contemporary issues.

Pk. **Professional Development** – Recognition of the need for and an ability to engage in continuing professional development and higher studies.

Pl. **Employment -** An ability to get an employment to the international repute industries through the training programs, internships, projects, workshops and seminars.

Index

List of Experiments

Sr. No.	Experiments Name
1	To study about instruction set of 8086 Microprocessor.
2	Write an assembly language program to accept and display “Hello World” on screen using DOS / BIOS.
3	Write an assembly language program to implement basic arithmetic operations on two 8 / 16 bit numbers.
4	Write an assembly language program to transfer data block using string instructions and without using string instructions.
5	Write an assembly language program to find the number / string is palindrome or not.
6	Write an assembly language program to sort elements in ascending /descending order.
7	Write an assembly language program to find the factorial of a number using procedure.
8	Write a program to separate even or odd numbers from array using mixed language programming.
9	Write a program to search number in an array using mixed language programming.
10	To study the 5-stage scalar pipeline (Non Linear Pipeline).
11	To Study The Effect Of Branch Operation On Linear Pipeline.

Course Objectives & Course Outcome, Experiment Plan

Course Objectives:

1.	To understand basic architecture of 16-bit and 32-bit microprocessors.
2.	To understand interfacing of 16-bit microprocessor with memory and peripheral
3.	To understand techniques for faster execution of instructions and improve speed of operation and performance of microprocessors
4.	To understand RISC and CISC based microprocessors.
5.	To understand concept of multi core processors

Course Outcomes:

CO1	Write programs to run on 8086 microprocessor based systems
CO2	Describe the action of the X86 assembly language instructions
CO3	Understand the design system using memory chips and peripheral chips for 16 bit 8086 microprocessor
CO4	Understand and devise techniques for faster execution of instructions, improve the speed of operations and enhance performance of microprocessors
CO5	Distinguish between RISC and CISC processors
CO6	Understand multi core processor and its advantages

Module No.	Week No.	Experiments Name	Course Outcome	Weightage
1	W1	To study about instruction set of 8086 Microprocessor.	CO1	3
2	W2	Write an assembly language program to accept and display “hello world” on screen using DOS / BIOS.	CO1	3
3	W3	Write an assembly language program to implement basic arithmetic operations on two 8 / 16 bit numbers.	CO1	4
4	W4	Write an assembly language program to transfer data block using string instructions and without using string instructions.	CO2	5
5	W5	Write an assembly language program to find the number / string is palindrome or not.	CO2	5
6	W6	Write an assembly language program to sort elements in ascending /descending order.	CO3	5
7	W7	Write an assembly language program to find the factorial of a number using procedure.	CO3	5
8	W8	Write a program to separate even or odd numbers from array using mixed language programming.	CO4	5
9	W9	Write a program to search number in an array using mixed language programming.	CO4	5
10	W10	Write a program to perform for a 5-stage scalar pipeline (Non Linear Pipeline).	CO5	5
11	W11	To Study The Effect Of Branch Operation On Linear Pipeline.	CO5	5

Mapping Course Outcomes (CO) - Program Outcomes (PO)

Subject Weight	Course Outcomes	Contribution to Program outcomes											
		P_a	P_b	P_c	P_d	P_e	P_f	P_g	P_h	P_i	P_j	P_k	P_l
PR 40%	**CO1:** Write programs to run on 8086 microprocessor based systems	3	2	2	1					1			
	CO2: Describe the action of the X86 assembly language instructions	3	2	1		1		1	1	1			
	CO3: Understand the design system using memory chips and peripheral chips for 16 bit 8086 microprocessor	2	3		3							1	1
	CO4: Understand and devise techniques for faster execution of instructions, improve the speed of operations and enhance performance of microprocessors		3		1	1			2	2	1		
	CO5: Distinguish between RISC and CISC processors	1	3			2	1		2				1
	CO6: Understand multi core processor and its advantages.	1	2				1	1			1	2	2

Study and Evaluation Scheme

Course Code	Course Name	Teaching Scheme			Credits Assigned			
CPC501	Microprocessor	Theory	Practical	Tutorial	Theory	Practical	Tutorial	Total
		4	2	-	4	1	-	5

Course Code	Course Name	Examination Scheme		
CPC501	Microprocessor	Term Work	Oral	Total
		25	25	50

Term Work:

1. Term work assessment must be based on the overall performance of the student with every experiment graded from time to time. The grades should be converted into marks as per the Credit and Grading System manual and should be added and averaged.
2. The final certification and acceptance of term work ensures satisfactory performance of laboratory work and minimum passing marks in term work.

Practical & Oral:

1. Practical examination will be conducted based on above mentioned experiment list.

Microprocessor

Experiment No. : 1

To study about instruction set of 8086 Microprocessor

Experiment No. 1

1. **Aim:** To study about instruction set of 8086 Microprocessor
2. **Objectives:**
   - To understand the basic concept in Microprocessor
   - To know the instruction set of 8086 Microprocessor
3. **Outcomes:** The learner will be able to
   - apply knowledge of computing, applied mathematics, and fundamental engineering concepts appropriate to the discipline
   - Familiar with 8086 instruction set.
4. **Hardware/Software Required :** TASM 5.0
5. **Theory: 8086 INSTRUCTION SET**

The mnemonics assigned to the instructions are designed to indicate the function of the instruction. The instructions fall into the following functional categories:

1. Data Transfer Group:

The data transfer instructions move data between registers or between memory and registers.

MOV	Move
MVI	Move Immediate
LDA	Load Accumulator Directly from Memory
STA	Store Accumulator Directly in Memory
LHLD	Load H & L Registers Directly from Memory
SHLD	Store H & L Registers Directly in Memory

An 'X' in the name of a data transfer instruction implies that it deals with a register pair (16- bits);

LXI	Load Register Pair with Immediate data
LDAX	Load Accumulator from Address in Register Pair
STAX	Store Accumulator in Address in Register Pair
XCHG	Exchange H & L with D & E
XTHL	Exchange Top of Stack with H & L

2. Arithmetic Group:

The arithmetic instructions add, subtract, increment, or decrement data in registers or memory.

ADD	Add to Accumulator
ADI	Add Immediate Data to Accumulator
ADC	Add to Accumulator Using Carry Flag
ACI	Add immediate data to Accumulator Using Carry
SUB	Subtract from Accumulator
SUI	Subtract Immediate Data from Accumulator
SBB	Subtract from Accumulator Using Borrow (Carry) Flag
	Subtract Immediate from Accumulator Using Borrow (Carry) Flag
INR	Increment Specified Byte by One
DCR	Decrement Specified Byte by One
INX	Increment Register Pair by One
DCX	Decrement Register Pair by One
DAD	Double Register Add; Add Content of Register Pair to H & L Register Pair

3. Logical Group:
This group performs logical (Boolean) operations on data in registers and memory and on condition flags.

The logical AND, OR, and Exclusive OR instructions enable you to set specific bits in the accumulator ON or OFF.

ANA	Logical AND with Accumulator
ANI	Logical AND with Accumulator Using Immediate Data
ORA	Logical OR with Accumulator
OR	Logical OR with Accumulator Using Immediate Data
XRA	Exclusive Logical OR with Accumulator
XRI	Exclusive OR Using Immediate Data

The Compare instructions compare the content of an 8-bit value with the contents of the accumulator;

CMP	Compare
CPI	Compare Using Immediate Data

The rotate instructions shift the contents of the accumulator one bit position to the left or right:

RLC	Rotate Accumulator Left
RRC	Rotate Accumulator Right
RAL	Rotate Left Through Carry
RAR	Rotate Right Through Carry

Complement and carry flag instructions:

CMA	Complement Accumulator
CMC	Complement Carry Flag
STC	Set Carry Flag

4. Branch Group:

The branching instructions alter normal sequential program flow, either unconditionally or conditionally. The unconditional branching instructions are as follows:

JMP	Jump
CALL	Call
RET	Return

Conditional branching instructions examine the status of one of four condition flags to determine whether the specified branch is to be executed. The conditions that may be specified are as follows:

NZ	Not Zero (Z = 0)
Z	Zero (Z = 1)
NC	No Carry (C = 0)
C	Carry (C = 1)
PO	Parity Odd (P = 0)
PE	Parity Even (P = 1)
P	Plus (S = 0)
M	Minus (S = 1)

Thus, the conditional branching instructions are specified as follows:

Jumps	Calls	Returns	
C	CC	RC	(Carry)
INC	CNC	RNC	(No Carry)
JZ	CZ	RZ	(Zero)
JNZ	CNZ	RNZ	(Not Zero)
JP	CP	RP	(Plus)
JM	CM	RM	(Minus)
JPE	CPE	RPE	(Parity Even)
JP0	CPO	RPO	(Parity Odd)

Two other instructions can affect a branch by replacing the contents or the program counter:

PCHL	Move H & L to Program Counter
RST	Special Restart Instruction Used with Interrupts

Stack I/O, and Machine Control Instructions:
The following instructions affect the Stack and/or Stack Pointer:

PUSH	Push Two bytes of Data onto the Stack
POP	Pop Two Bytes of Data off the Stack
XTHL	Exchange Top of Stack with H & L
SPHL	Move content of H & L to Stack Pointer

The I/0 instructions are as follows:

IN	Initiate Input Operation
OUT	Initiate Output Operation

The Machine Control instructions are as follows:

EI	Enable Interrupt System
DI	Disable Interrupt System
HLT	Halt
NOP	No Operation

Procedure for execute the Assembly program using TASM

- Assembly language programs are converted into executable machine code by a utility program referred to as an **assembler**, the conversion process being referred to as assembly or assembling the program.

- **Assembly language** (sometimes abbreviated as **ASM**, usually as the file extension for a text file which is used as a code for a program written in Assembly language, or in the names of assemblers, like FASM, MASM, NASM and TASM) is a low-level programming language for computers, microprocessors, microcontrollers, and other programmable devices in which each statement corresponds to a single machine language instruction. An assembly language is specific to a certain computer architecture, in contrast to most high-level programming languages, which generally are portable to multiple systems.

What is TASM assembler?

The **Turbo Assembler** (TASM) is an x86 assembler that uses the Intel syntax for MS-DOS and Microsoft Windows. Beginning with TASM 8.0 there are two versions of the assembler - one for 16-bit and 32-bit assembly sources, and another (**ML64**) for 64-bit sources only.

Assembling and Running Assembly Language Programs

An assembly language program must be assembled and linked before it can be executed. The assembler produces an object file (extension .OBJ). This file is taken by the linker and an executable program (extension .EXE) is produced, assuming there were no errors in the program. We use the MASM assembler and the LINK linker. These are available on NAL under Programming:“TASM files v 5.0”.When saving the file with Notepad, you MUST save it with the “File Type” set to “All Files”. You should now select the MS-DOS Prompt (Command PROMPT) from the Start button menu (sometimes under Programs option)

To do assembly language programming TASM assembler you can follow these steps:

Following are steps to execute a assembly program in tasm assembler.

1. Save .asmextention file by writing code in text editor.

2. Open dos prompt

3. Go the target file by prompt

4. Write tasm filename.asm and press enter

5. Write tlink filename.obj and press enter

6. Write debug filename.exe and press enter

7. Cursor will be displayed. Press –t for single step debugging mode otherwise –g for direct compilation.

Tasm folder must contains

Tasm exe

Tlink exe

Td.exe

Following are steps to execute a mixed language program in TURBOC.

1. Write a program in turbo c editor and save it as .cpp file

2. Go the compile

3. Run the compile file

Note: GUI based Emulator 8086 can also be used to write and execute assembly programs.

In the event of errors, you must edit your program and correct the errors. Then you repeat the above steps to assemble and link your program, before running it.

Similarly, if you modify your program, you must assemble and link it before running it again.

6. **Conclusion**: We learn the instruction set of 8086 microprocessor like data transfer instruction, logical instruction, arithmetic instruction and branching instruction.

7. Viva Questions:

- Give the feature of 8086 microprocessor.
- What is an instruction?
- What are different types of instructions? Give examples

8. References:

1. Microprocessor and Interfacing: Douglas Hall, Tata McGraw Hill

2. Microcomputer Systems: 8086/8088 family Architecture, Programming and Design: Liu & Gibson, PHI Publication

Microprocessor

Experiment No. 2

Program to accept and display “Hello World” on screen using DOS / BIOS

Experiment No. 2

1. **Aim:** Write an assembly language program to accept and display "Hello World" on screen using DOS / BIOS

2. **Objectives:**
   - To understand techniques for execution of instructions
   - To know interrupt of 8086 Microprocessor

3. **Outcomes:** The learner will be able to
   - Familiar with 8086 instruction set and the execution of instructions.
   - understand, identify, analyze and design the problem, implement and validate the solution including both hardware and software.

4. **Hardware/Software Required :** TASM 5.0
5. **Theory:** For 8086 two types interrupt can be generated software and hardware

The DOS (Disk Operating System) provides a large number of procedures to access devices, files and memory. These procedures can be called in any user program using software interrupts "**INT n**" instruction.

The steps involved in accessing DOS services are :

1. Load a DOS function number in AH register. If there is a sub-function then it is loaded in AL register.
2. Load the other registers as indicated in the DOS service formats.
3. Prepare buffers, ASCIIZ (ASCII string terminated by zero) and control blocks if necessary.
4. Set the location of the disk area if necessary.
5. Invoke DOS service INT 21H which will return the required parameters in the specified register.

Many software programs written for 8086 computers are designed to run under the MS-DOS operation system. Included as part of this operating system are the DOS functions and BIOS calls. These are subprograms, callable from applications software that can be used to access the hardware of the PC. The intention is to save the programmer from having to "reinvent the wheel" with each new applications program. In addition, by providing a standard set of input/output routines, these subprograms ensure software compatibility between computers with different hardware configurations.

The BIOS routines are the most primitive in a computer as they "talk" directly to the system hardware. Accordingly, the BIOS is hardware specific that is, it must know the exact port address and control bit configurations for each I/O.

Algorithm:

1. Declare Hello world as string

2. Initialize to program's Data Segment
2. Use DOS Function 09 H to Display a string and interrupt Int 21h
3. Execute DOS Function.

6. Conclusion. Thus understand the use of interrupt of 8086 in simple hello word string.

7. Viva Questions:

- What is an interrupt?
- What are different types of interrupt in 8086?
- How many segment registers are there in 8086?
- What is data segment?

8. References:

1. Microprocessor and Interfacing: Douglas Hall, Tata McGraw Hill

2. Microcomputer Systems: 8086/8088 family Architecture, Programming and Design: Liu & Gibson, PHI Publication

Microprocessor

Experiment No. : 3

Program to implement basic arithmetic operations on two 8 / 16 bit numbers

Experiment No. 3

1. **Aim:** Write an assembly language program to implement basic arithmetic operations on two 8 / 16 bit numbers
2. **Objectives:**
   - To understand the basic arithmetic operation in Microprocessor
   - To understand techniques for faster execution of instructions and improve speed of operation and performance of microprocessors
3. **Outcomes:** The learner will be able to

- Familiar with 8086 arithmetic operations instruction
- Understand, identify, analyze the problem ,implement and validate the solution including both hardware and software

4. **Hardware/Software Required :** TASM 5.0
5. **Theory:** Following instruction may be used for implementation and its formats are as follows

ADD **Add two numbers**

Syntax: Add dest, src
dest: register or memory
src: register, memory, or immediate
Action: dest = dest + src
Flags Affected: OF, SF, ZF, AF, PF, CF
Notes: Works for both signed and unsigned numbers.

For Example :	AX = 1234 H		1234 H
	BX = 0100 H	+	0100 H
			1334 H

SUB **Subtract two numbers**

Syntax: Sub dest, src
dest: regsiter or memory
src: register, memory, or immediate
Action: dest = dest – src
Flags Affected: OF, SF, ZF, AF, PF, CF
Notes: Works for both signed and unsigned numbers.

MUL **Unsigned multiply**

Syntax: Mul op8
mul op16 op8: 8-bit
register or memory
op16: 16-bit register or memory
Action: If operand is op8, unsigned AX = AL * op8
If operand is op16, unsigned DX::AX = AX * op16
Flags Affected: OF, SF=?, ZF=?, AF=?, PF=?, CF

DIV **Unsigned divide**
Syntax: Div op8
Div op16
op8: 8-bit register or memory
op16: 16-bit register or memory
Action: If operand is op8, unsigned AL = AX / op8 and AH = AX % op8
If operand is op16, unsigned AX = DX::AX / op16 and DX = DX::AX % op16
Flags Affected: OF=?, SF=?, ZF=?, AF=?, PF=?, CF=?
Notes: Performs both division and modulus operations in one instruction.

IDIV **Signed divide**
Syntax: Idiv op8
Idiv op16
op8: 8-bit register or memory
op16: 16-bit register or memory
Action: If operand is op8, signed AL = AX / op8 and AH = AX % op8
If operand is op16, signed AX = DX::AX / op16 and DX = DX::AX % op16
Flags Affected: OF=?, SF=?, ZF=?, AF=?, PF=?, CF=?
Notes: Performs both division and modulus operations in one instruction.

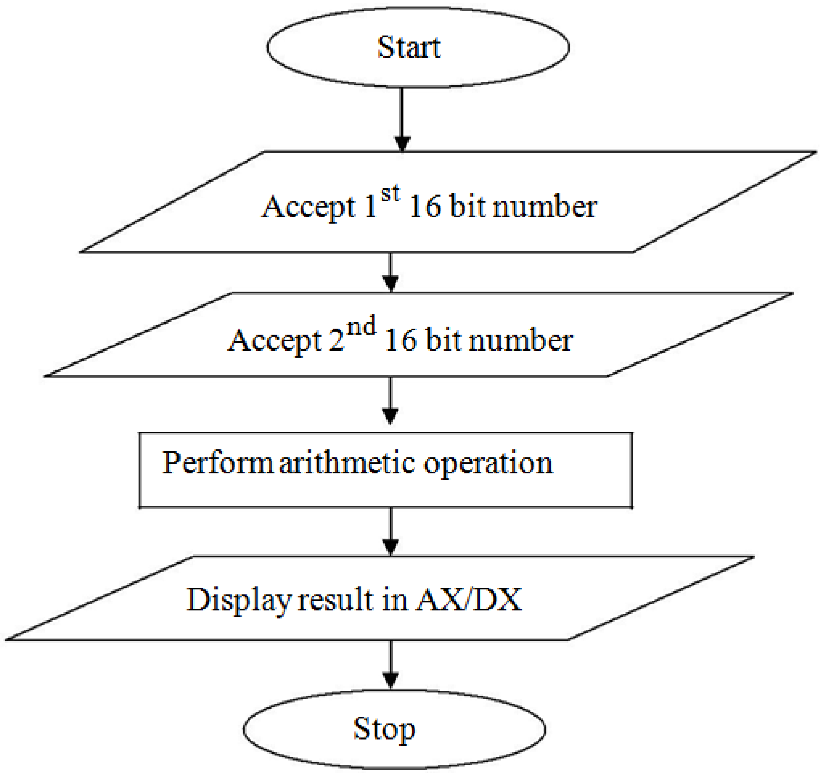
IMUL **Signed multiply**
Syntax: Imul op8
Imul op16
op8: 8-bit register or memory
op16: 16-bit register or memory
Action: If operand is op8, signed AX = AL * op8
If operand is op16, signed DX::AX = AX * op16
Flags Affected: OF, SF=?, ZF=?, AF=?, PF=?, CF

Algorithm :

1. Initialize the data segment.
2. Get the first number in AX register.
3. Get the second number in BX register.
4. Perform arithmetic operation on two numbers.
5. Display the AX/DX result.

6. Stop.

Flow Chart:



6. **Conclusion:** We have used the arithmetic instructions like Add, Sub, Mul and Div in assembly language.

7. **QUIZ / Viva Questions:**

- What is an interrupt?
- What are different types of interrupt in 8086?
- How many segment registers are there in 8086?
- What is data segment?

8. References:

1. Microprocessor and Interfacing: Douglas Hall, Tata McGraw Hill

2. Microcomputer Systems: 8086/8088 family Architecture, Programming and Design: Liu & Gibson, PHI Publication

Microprocessor

Experiment No. : 4

Program to transfer data block using string instructions and without using string instructions

Experiment No. 4

1. **Aim:** Write an assembly language program to transfer data block using string instructions and without using string instructions
2. **Objectives:**
    - To understand the string instruction in Microprocessor
    - To understand techniques for faster execution of instructions and improve speed of operation and performance of microprocessors

3. **Outcomes:** The learner will be able to
    - Work with 8086 string instruction operation.
    - apply knowledge of computing, applied mathematics, and fundamental engineering concepts appropriate to the discipline.

4. **Hardware/Software Required :** TASM 5.0

5. **Theory:** Consider that a block of data of N bytes is present at source location. Now this block of N bytes is to be moved from source location to a destination location.

    Let the number of bytes N = 10.

    We will have to initialize this as count in the CX register.

    We know that source address is in the SI register and destination address is in the DI register.

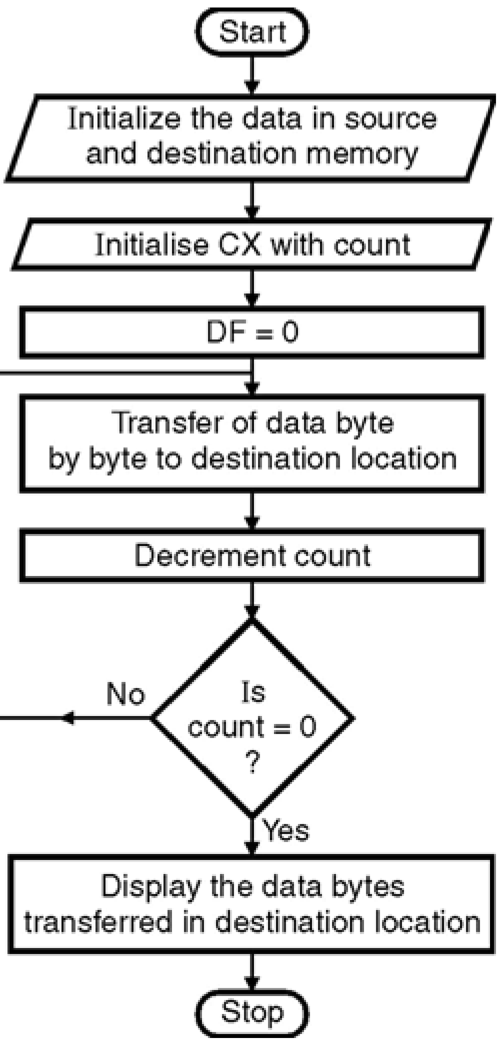
    Clear the direction flag.

    Using the string instruction move the data from source location to the destination location. It is assumed that data is moved within the same segment. Hence the DS and ES are initialized to the same segment value.

Algorithm :

1. Initialize the data in the source memory and destination memory.
2. Initialize SI and DI with source and destination address.
3. Initialize CX register with the count.
4. Initialize the direction flag to zero.
5. Transfer the data block byte by byte to destination.
6. Decrement CX.
7. Check for count in CX, if not zero go tostep 5 else to step 8.
8. Stop.

Flowchart :



Start
Initialize the data in source and destination memory
Initialise CX with count
DF = 0
Transfer of data byte by byte to destination location
Decrement count
Is count = 0 ?
No
Yes
Display the data bytes transferred in destination location
Stop

6. **Conclusion:** Using the string instruction the data move from source location to the destination location, data is moved within the same segment. Hence the DS and ES are initialized to the same segment value.

7. **Viva Questions:**
   - What is the use of string instruction?
   - Explain use of DS and ES

8. References:

1. Microprocessor and Interfacing: Douglas Hall, Tata McGraw Hill

2. Microcomputer Systems: 8086/8088 family Architecture, Programming and Design: Liu & Gibson, PHI Publication

Experiment No. : 5

Program to find the number / string is palindrome or not.

Experiment No. 5

1. **Aim:** Write an assembly language program to find the number / string is palindrome or not.
2. **Objectives:**
    - To understand the string operation in Microprocessor
    - To understand techniques for faster execution of instructions and improve speed of operation and performance of microprocessors
3. **Outcomes:** The learner will be able to
    - Work with 8086 string operation instruction
    - Identify, formulate, and solve engineering problems.
4. **Hardware/Software Required :** TASM 5.0
5. **Theory:** This program is used to check whether the given string is palindrome or not. Here palindrome means we compare the character sequence from left to right and right to left in which they sound same.

Algorithm :

```
1.    Start.
2.    Initialize pointer at the start of string.
3.    Initialize pointer at the end of string.
4.    Initialize counter = length/2.
5.    Decrement counter.
6.    All Character equal if yes, display the message.
7.    Is count = 0 if yes display the message string is palindrome else go
      to step 5.
8.    Stop.
```

6. **Conclusion:** we compare the character sequence from left to right and right to left in which they sound same.
7. **Viva Questions:**
    - Which instructions are use to find out palindrome?
    - What is palindrome and explain how it works in 8086 processor?

8.References:

1. Microprocessor and Interfacing: Douglas Hall, Tata McGraw Hill

2. Microcomputer Systems: 8086/8088 family Architecture, Programming and Design: Liu & Gibson, PHI Publication

Experiment No. : 6

Program to sort elements in ascending /descending order.

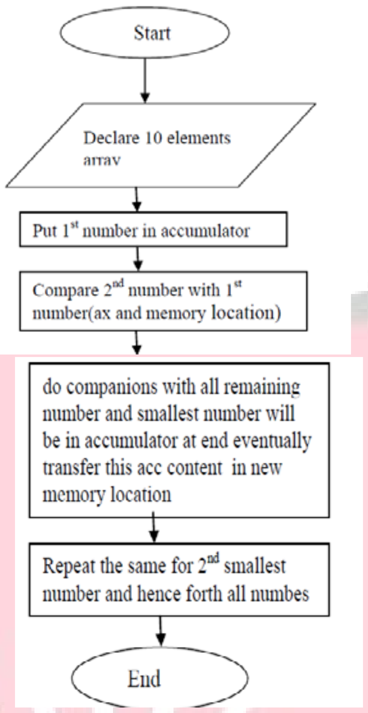
Experiment No. 6

1. **Aim:** Write an assembly language program to sort elements in ascending /descending order
2. **Objectives:**
   - To understand the basic CMP operation in Microprocessor
   - To understand techniques for faster execution of instructions and improve speed of operation and performance of microprocessors
3. **Outcomes:** The learner will be able to
   - Familiar with 8086 CMP operations instruction
   - understand, identify, analyze the problem, implement and validate the solution including both hardware and software
4. **Hardware/Software Required :** TASM 5.0
5. **Theory:** Bubble sort is basic technique to sort the numbers in ascending or descending order but writing code in assembly language is bit challenging. Sorting numbers stored in memory can be performed by transferring every number into register and comparing second number in memory using CMP instruction. This comparison can be done for all number in pair.

Algorithm :

1. Declare 10 elements array.
2. Put 1st number in accumulator i.e AX.
3. Compare 2nd number with 1st number(ax and memory location) using CMP INSTRUCTION
4. Do comparisons with all remaining number and smallest number will be in accumulator USING LOOP AND JZ/JS INSTRUCTION.
5. Transfer this accumulator content in new memory location USING MOV.
6. Repeat the same for 2nd smallest number and hence forth all number USING INCSI AND DI.
7. Stop.

Flowchart :



Start
Declare 10 elements array
Put 1st number in accumulator
Compare 2nd number with 1st number(ax and memory location)
do companions with all remaining number and smallest number will be in accumulator at end eventually transfer this acc content in new memory location
Repeat the same for 2nd smallest number and hence forth all numbes
End

6. **Conclusion :**We learn basic technique to sort the numbers in ascending or descending order but writing code in assembly language

7. **Viva Questions:**
   - Explain the CMP instruction
   - What is accumulator?
   - Explain JZ/JS INSTRUCTIO

8. **References:**

1. Microprocessor and Interfacing: Douglas Hall, Tata McGraw Hill

2. Microcomputer Systems: 8086/8088 family Architecture, Programming and Design: Liu & Gibson, PHI Publication

Experiment No. : 7

Program to find the factorial of a number using procedure.

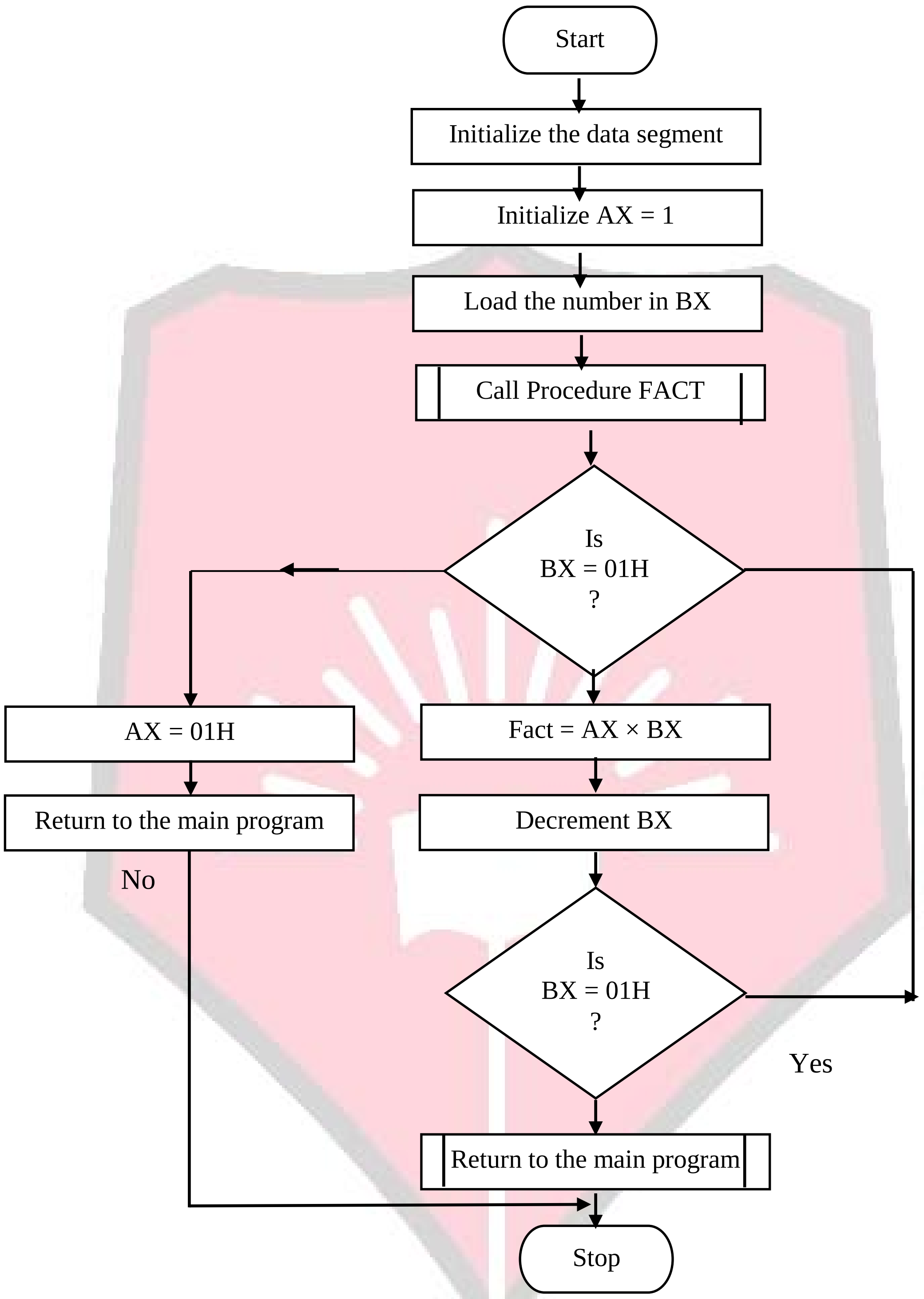
Experiment No. 7

1. **Aim:** Write an assembly language program to find the factorial of a number using procedure
2. **Objectives:**
    - To understand the procedure operation in Microprocessor
    - To understand techniques for faster execution of instructions and improve speed of operation and performance of microprocessors
3. **Outcomes:** The learner will be able to
    - Work with 8086 procedure operation.
    - Identify, formulate, and solve engineering problems
4. **Hardware/Software Required :** TASM 5.0
5. **Theory:** To compute the factorial of a number means to multiply a number n with (n-1) (n-2) …× 2 × 1.
   Example : to compute 5! = 5×4×3×2×1=120.
   We will initialize AX=1 and load the number whose factorial is to be computed in BX. Call procedure fact, which will calculate the factorial of the number.

Algorithm :

1. Initialize the Data Segment.
2. Initialize AX = 1.
3. Load the number in BX.
4. Call procedure fact.
5. Compare BX with 1, if not go to step 7.
6. AX = 1 and return back to the calling program.
7. AX = AX × BX.
8. Decrement BX.
9. Compare BX with 1, if not go to step 7.
10. Return back to calling program.
11. Stop.

Flowchart :



6. **Conclusion :** We have use the procedure in 8086 microprocessor to find the factorial.

7. **Viva Questions:**
   - What is procedure in 8086 microprocessor?
   - Explain the instructions used in finding factorial.

8. References:

1. Microprocessor and Interfacing: Douglas Hall, Tata McGraw Hill

2. Microcomputer Systems: 8086/8088 family Architecture, Programming and Design: Liu & Gibson, PHI Publication

Experiment No. : 8

Program to separate even or odd numbers from array using mixed language programming.

Experiment No. 8

1. **Aim:** Write a program to separate even or odd numbers from array using mixed language programming.
2. **Objectives:**
   - To understand the mixed language programming
   - To understand new techniques to solve the problem by Inline assembler
3. **Outcomes:** The learner will be able to
   - Familiar with mixed language programming
   - Recognize the need for, and ability to engage in life-long learning.

4. **Hardware/Software Required :** Inline assembler
5. **Theory:** Mixed-language programming allows you to combine the unique strengths of C++ with your assembly-language routines. There are some cases where you want to achieve things using inline assembly, such as improving speed, reducing memory needs and getting more efficiency. However, inline assembler is not as powerful as TASM, it does not support macros or directives.
   You can write small assembly language routines within your C or C++ code. These routines are compiled using the inline or embedded assembler of the TURBOC compiler. However, there are a number of restrictions to the assembly language code you can write if you are using the inline or embedded assembler. These restrictions are such as instruction set and number of registers can be used.In c++ , every integer will take 2 bytes therefore for accessing any element using from array assembly pointer should increment and decrement by 2.

Algorithm :

1. Accept array of elements using cin or scanf functions.
2. For every element, check modulus of 2 After bring element into accumulator register(DIV instruction can be used).
3. If remainder is zero then it is even otherwise it odd number put into respective array.
4. Display respective even and odd arrays.

6. **Conclusion :** Mixed-language programming allows us to combine the unique strengths of C++ with our assembly-language routines and improving speed, reducing memory needs and getting more efficiency.

7. **Viva Questions:**
   - What is mixed language programming?
   - What is Inline assembler?

8. References:

1. Microprocessor and Interfacing: Douglas Hall, Tata McGraw Hill
2. Microcomputer Systems: 8086/8088 family Architecture, Programming and Design: Liu & Gibson, PHI Publication
3. Assembly Language for x86 Processors by KIP R. IRVINE

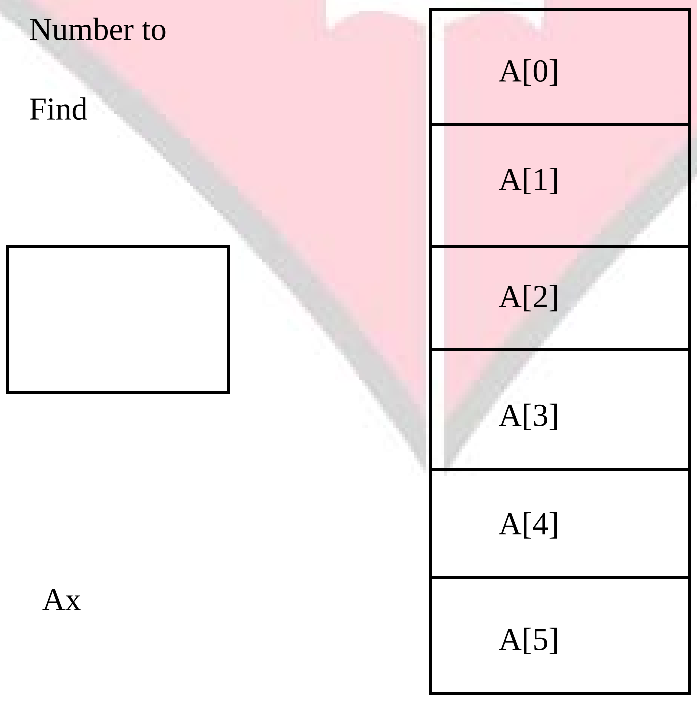
Experiment No. : 9

Program to search number in an array using mixed language programming.

Experiment No. : 9

1. **Aim:** To search number in an array using mixed language programming.
2. **Objectives:**
   - To understand mixed language programming.
   - To understand new techniques to solve the problem by Inline assembler

3. **Outcomes:** The learner will be able to
   - Familiar with mixed language programming.
   - an ability to recognize the need for and an ability to engage in life-long learning
4. **Hardware/Software Required :** Inline assembler
5. **Theory:** Mixed-language programming is the process of building programs in which the source code is written in two or more languages. Although mixed-language programming presents some additional challenges, it is worthwhile because it enables you to call existing code that may be written in another language**.** Search for given array can be performed with simple c++ code but using mixed language program can be quite time consuming since searching array which is defined in high level language such as c++ or java has more memory allocation than assembly language. Binary search or sequential search can be applied to achieve this objective. Here in this case number which is to be found can be stored into register AX and then either simple CMP OR SCAS can be used to search a number by comparing array with number. Following figure depicts searching number.



Algorithm:

1. Declare array of 10 elements in c++ .
2. Accept number which is to be searched.
3. Compare number either by CMP or SCAS.
4. Use interrupt to display found message if it is found.
5. Else exit the loop.

6. **Conclusion:** Search for given array can be performed with simple c++ code but using mixed language program can be quite time consuming since searching array which is defined in high level language such as c++ or java has more memory allocation than assembly language.

7. **Viva Questions:**
   - Explain the CMP and SCAS
   - Explain the difference between mixed language program and high level language

8. References:

1. Microprocessor and Interfacing: Douglas Hall, Tata McGraw Hill
2. Microcomputer Systems: 8086/8088 family Architecture, Programming and Design: Liu & Gibson, PHI Publication
3. Assembly Language for x86 Processors by KIP R. IRVINE

Experiment No. : 10

To study a 5-stage scalar pipeline (Non Linear Pipeline).

Experiment No. : 10

1. **Aim:** To study 5-stage scalar pipeline (Non Linear Pipeline).

2. **Objectives:**
    - To study 5-stage scalar pipeline (Non Linear Pipeline).
    - To understand concept of multi core processors

3. **Outcomes:** The learner will be able to
    - Understand the 5-stage scalar pipeline(Non Linear Pipeline)
    - An ability to understand,identify,analyze the system
4. **Hardware/Software Required:**

5. **Theory:** Pipelining:

- It is a technique of decomposing a sequential process into sub-operations which can be executed in a special dedicated segment that operates concurrently with all other segments

- It improves processor performance by overlapping the execution of multiple instructions

Example for pipeline in computer

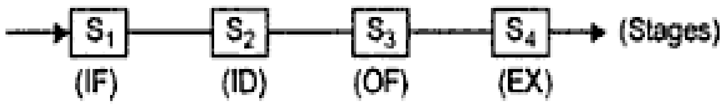
- Consider that the process of execution of an instruction involves four major steps:

1. **Instruction Fetch (IF):** from main memory

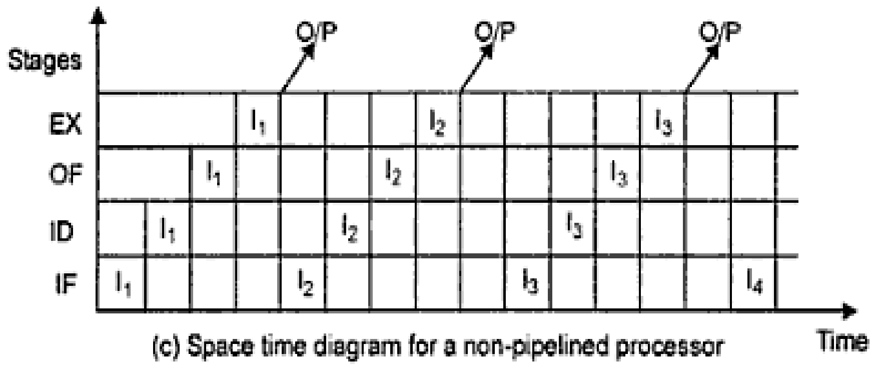
2. **Instruction Decoding (ID):** which identifies the operation to be performed

3. **Operand Fetch(OF):** if needed in execution

4. **Execution(EX):** of the decoded arithmetic/logic operation

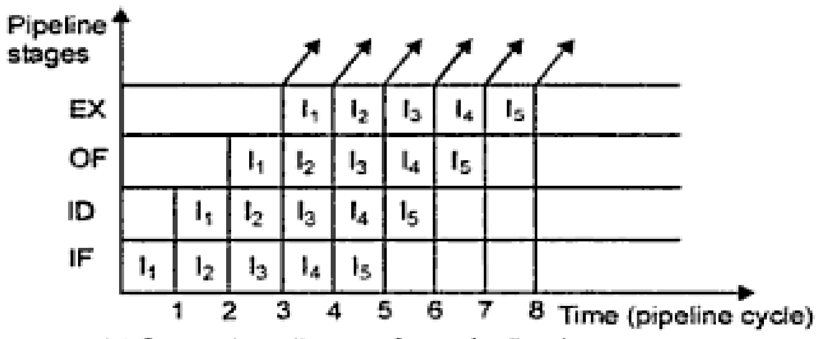


- In a **non-pipelined computer**, these four steps must be completed before the next instruction can be issued



(c) Space time diagram for a non-pipelined processor

- In a pipelined computer, successive stages are executed in an overlapped fashion



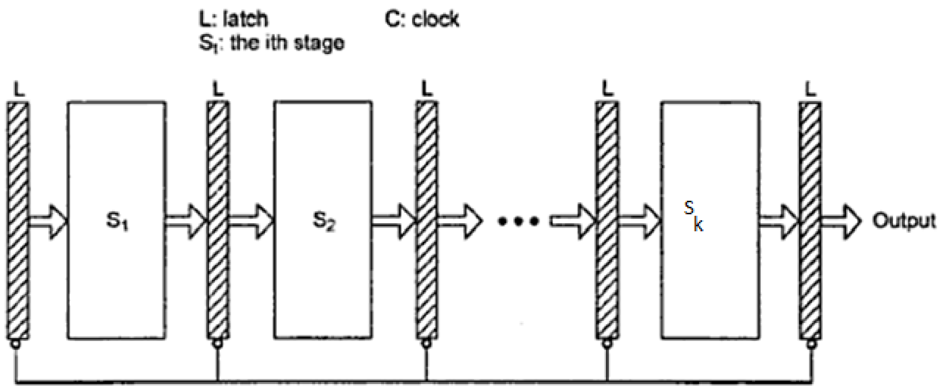
(b) Space time diagram for a pipelined processor

- Theoretically a **k-stage** linear pipeline could be **k-times faster.**
- But this ideal speedup cannot be achieved due to factors like :
  - **Data dependency**
  - **Branch and Interrupts**

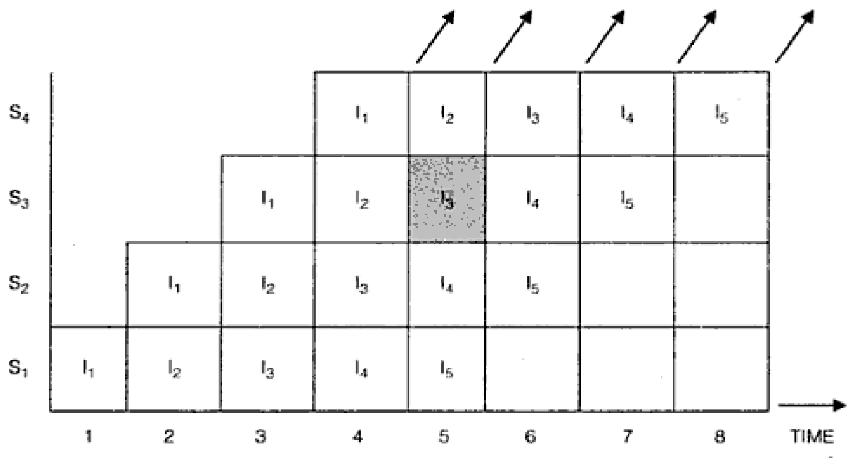
Principles of Linear Pipelining

- In pipelining, we divide a task into set of subtasks.
- The **precedence relation** of a set of subtasks {T_1, T_2,…, T_k} for a given task T implies that the task **T_j cannot start until** some earlier task **T_i finishes.**
- The **interdependencies** of all subtasks **form the precedence graph.**
- With a linear precedence relation, **task T_j cannot start** until earlier subtasks **{ T_i} for all (i < j) finish.**
- A **linear pipeline** can process subtasks with a linear precedence graph.

Basic Linear Pipeline



- L: latches, interface between different stages of pipeline
- S1, S2, etc. : pipeline stages
- It consists of cascade of processing stages.
- **Stages : Pure combinational circuits performing arithmetic or logic operations over the data** flowing through the pipe.
- Stages are separated by **high speed interface latches**.
- **Latches :** Fast Registers holding intermediate results between stages
- **Information Flow** are under the **control of common clock** applied to all latches
- The flow of data in a linear pipeline having four stages for the evaluation of a function on five inputs is as shown below:



- The vertical axis represents four stages
- The horizontal axis represents time in units of clock period of the pipeline.

Clock Period (τ) for the pipeline

- Let τ_i be the time delay of the circuitry S_i and t_1 be time delay of latch.
- Then the **clock period** of a linear pipeline is defined by

$$\tau = \max_{i=1}^{k}\{\tau_i\} + t_1 = t_m + t_1$$

The reciprocal of clock period is called clock frequency ($f = 1/\tau$) of a pipeline processor

Performance of a linear pipeline

- Consider a linear pipeline with **k stages.**
- Let **T be the clock period** and the pipeline is initially empty.
- Starting at any time, let us feed **n inputs** and wait till the results come out of the pipeline.
- **First input takes k periods** and the remaining (**n-1) inputs come one after the another** in successive clock periods.
- Thus the **computation time** for the pipeline T_P is

$$\mathbf{T_P = kT+(n-1)T = [k+(n-1)]T}$$

- For example if the linear pipeline have four stages with five inputs.
- Tp = [k+(n-1)]T = [4+4]T = 8T

Performance Parameters

- The various performance parameters of pipeline are :

1. Speed-up
2. Throughput
3. Efficiency

Speedup

Speedup is defined as

$$\text{Speedup} = \frac{\text{Time taken for a given computation by a non-pipelined functional unit}}{\text{Time taken for the same computation by a pipelined version}}$$

Assume a function of k stages of equal complexity which takes the same amount of time T.

Non-pipelined function will take kT time for one input.

Then Speedup = nkT/(k+n-1)T = nk/(k+n-1)

The maximum value of speedup is

$$\lim_{n \to \infty} [\text{Speedup}] = k$$

Efficiency

It is an indicator of how efficiently the resources of the pipeline are used.

If a stage is available during a clock period, then its availability becomes the unit of resource.

Efficiency can be defined as

$$\text{Efficiency} = \frac{\text{Number of stage time units actually used during computation}}{\text{Total number of stage time units available during that computation}}$$

No. of stage time units = *nk* (there are n inputs and each input uses k stages.)

Total no. of stage-time units available = k[k + (n-1)]

It is the product of no. of stages in the pipeline (k) and no. of clock periods taken for computation(k+(n-1)).

Thus efficiency is expressed as follows:

$$\text{Efficiency} = \frac{\text{nk}}{k[\text{k} + \text{n} - 1]} = \frac{n}{k + n - 1}$$

The maximum value of efficiency is 1.

Throughput

It is the average number of results computed per unit time.

For n inputs, a k-staged pipeline takes [k+(n-1)]T time units

Then,

Throughput = n / [k+n-1] T = nf / [k+n-1]

where f is the clock frequency

The maximum value of throughput is

$$\lim_{n \to \infty} [\text{Throughput}] = f$$

Throughput = Efficiency x Frequency

6. **Conclusion :** Pipe line improves processor performance by overlapping the execution of multiple instructions

7. **Viva Questions:**
   - What is pipelining?
   - How pipeline improves the performance of 8086 processor

8. References:

1. Microprocessor and Interfacing: Douglas Hall, Tata McGraw Hill
2. Microcomputer Systems: 8086/8088 family Architecture, Programming and Design: Liu & Gibson, PHI Publication

Experiment No. : 11

To Study the effect of Branch Operation on Linear Pipeline.

Experiment No. : 11

1. **Aim:** To Study the effect of Branch Operation on Linear Pipeline.
2. **Objectives:**
    - To study the effect of Branch operation on linear pipeline
    - To understand techniques for faster execution of instructions and improve speed of operation and performance of microprocessors

3. **Outcomes:** The learner will be able to
    - Understand branch operation on linear pipeline
    - Understand, identify, analyze and design problem**.**
4. **Hardware/Software Required:**
5. **Theory:** Branch prediction is one of the ancient performance improving techniques which still finds relevance into modern architectures. While the simple prediction techniques provide fast lookup and power efficiency they suffer from high mis prediction rate. On the other hand, complex branch predictions – either neural based or variants of two level branch prediction – provide better prediction accuracy but consume more power and complexity increases exponentially. In addition to this, in complex prediction techniques the time taken to predict the branches is itself very high – ranging from 2 to 5 cycles – which is comparable to the execution time of actual branches.

    Branch prediction is essentially an optimization (minimization) problem where the emphasis is on to achieve lowest possible miss rate, low power consumption and low complexity with minimum resources. There really are three different kinds of branches:

    Forward conditional branches - based on a run-time condition, the PC (Program Counter) is changed to point to an address forward in the instruction stream. Backward conditional branches - the PC is changed to point backward in the instruction stream. The branch is based on some condition, such as branching backwards to the beginning of a program loop when a test at the end of the loop states the loop should be executed again.

    Unconditional branches - this includes jumps, procedure calls and returns that have no specific condition. For example, an unconditional jump instruction might be coded in assembly language as simply "jmp", and the instruction stream must immediately be directed to the target location pointed to by the jump instruction, whereas a conditional jump that might be coded as "jmpne" would redirect the instruction stream only if the result of a comparison of two values in a previous "compare" instructions shows the values to not be equal. (The segmented addressing scheme used by the x86 architecture adds extra complexity, since jumps can be either "near" (within a segment) or "far" (outside the segment).

    Each type has different effects on branch prediction algorithms.) A Closer Look At Branch Prediction Static Branch Prediction predicts always the same direction for the same branch during the whole program execution. It comprises hardware-fixed prediction and compiler-directed prediction. Simple hardware-fixed direction mechanisms can be: Predict always not taken Predict always taken Backward branch predict taken, forward branch predict not taken Sometimes a bit in the branch opcode allows the compiler to decide the prediction direction.

Static Branch Prediction Dynamic Branch Prediction: the hardware influences the prediction while execution proceeds. Prediction is decided on the computation history of the program. During the start-up phase of the program execution, where a static branch prediction might be effective, the history information is gathered and dynamic branch prediction gets effective. In general, dynamic branch prediction gives better results than static branch prediction, but at the cost of increased hardware complexity.

6. **Conclusion:** Branch prediction is one of the ancient performance improving techniques which still finds relevance into modern architectures.

7. **Viva Questions:**
   - What is pipelining?
   - How branch prediction improves the performance of 8086 processor

8. References:

1. Microprocessor and Interfacing: Douglas Hall, Tata McGraw Hill
2. Microcomputer Systems: 8086/8088 family Architecture, Programming and Design: Liu & Gibson, PHI Publication