SYSTEM PROGRAMMING

Nibaran Das

Email: nibaran@cse.jdvu.ac.in

- Textbook
  - Leland L. Beck, "System Software: An Introduction to Systems Programming" (3rd Edition), published by Addison Wesley.
  - Srimanta Pal, “system Programming”, Oxford University Press
  - Systems Programming and Operating Systems – D. M. Dhamdhere, TMH
  - Linux Device Drivers – J. Corbet, A. Rubini, G. Kroah-Hartman, O’ Reilly Media

INTRODUCTION

- Software
  - Application software usually used by end-user
    - It is concerned with the solution of some problem, using the computer as a tool, instead of how computers actually work.
  - System software
    - System software consists of a variety of programs that support the operation of a computer (ex: text editor, compiler, debugger)
    - One characteristic in which most system software differ from application software is machine dependency
    - A system software programmer must know the target machine structure

Email: nibaran@cse.jdvu.ac.in

System Software vs Application

- One characteristic in which most system software differs from application software is machine dependency.
- System programs are intended to support the operation and use of the computer itself, rather than any particular application.

Examples of system software

- Text editor, assembler, compiler, loader or linker, debugger, macro processors, operating system, database management systems, software engineering tools, ...

GOAL OF SYSTEM PROGRAMMING?

- a) To understand the basics of system programs like editors, compiler, assembler, linker, loader, interpreter and debugger.
- b) Describe the various concepts of assemblers and macroprocessors.
- c) To understand the various phases of compiler and compare its working with assembler.
- d) To understand how linker and loader create an executable program from an object module created by assembler and compiler.
- e) To know various editors and debugging techniques

Email: nibaran@cse.jdvu.ac.in

SYLLABUS?-WHAT IS THERE…CONTD.

- Introduction to Systems Programming [**1L**]
- Assembler - Functions of an assembler, features (with respect to machine dependence) of an
- assembler, design of assembler – two pass, one pass, concept of overlay
- [**6L**]
- Assembler Design: A case study – Overview of 16 / 32 bit architecture and assembly language
- programming features, Functionalities and design of assembler for such specifications
- [**6L**]
- Macro processor – Functions of macro processor, features of macro processor, design of macro
- processor [**3L**]
- Loaders and Linkers – functions of loader, absolute loader, bootstrap loader, Machine dependent
- and machine independent features of loader, Relocation, Linking, concept and design of
- relative/relocating loader, linking loader, linkage editor, dynamic linking and dynamic loading
- [**6L**]
- Text editor – types of editors, types of files, features, examples [**2L**]
- Cross assembler – requirements, features, example [**2L**]
- Debug – functions of a debugger, hardware support for debugging, example debuggers
- [**2L**]
- Device drivers – concepts, design and developing [**8L**]
- Window manager – Features and facilities, types, examples of X Windows manager, widget
- toolkit [**4L**

Email: nibaran@cse.jdvu.ac.in

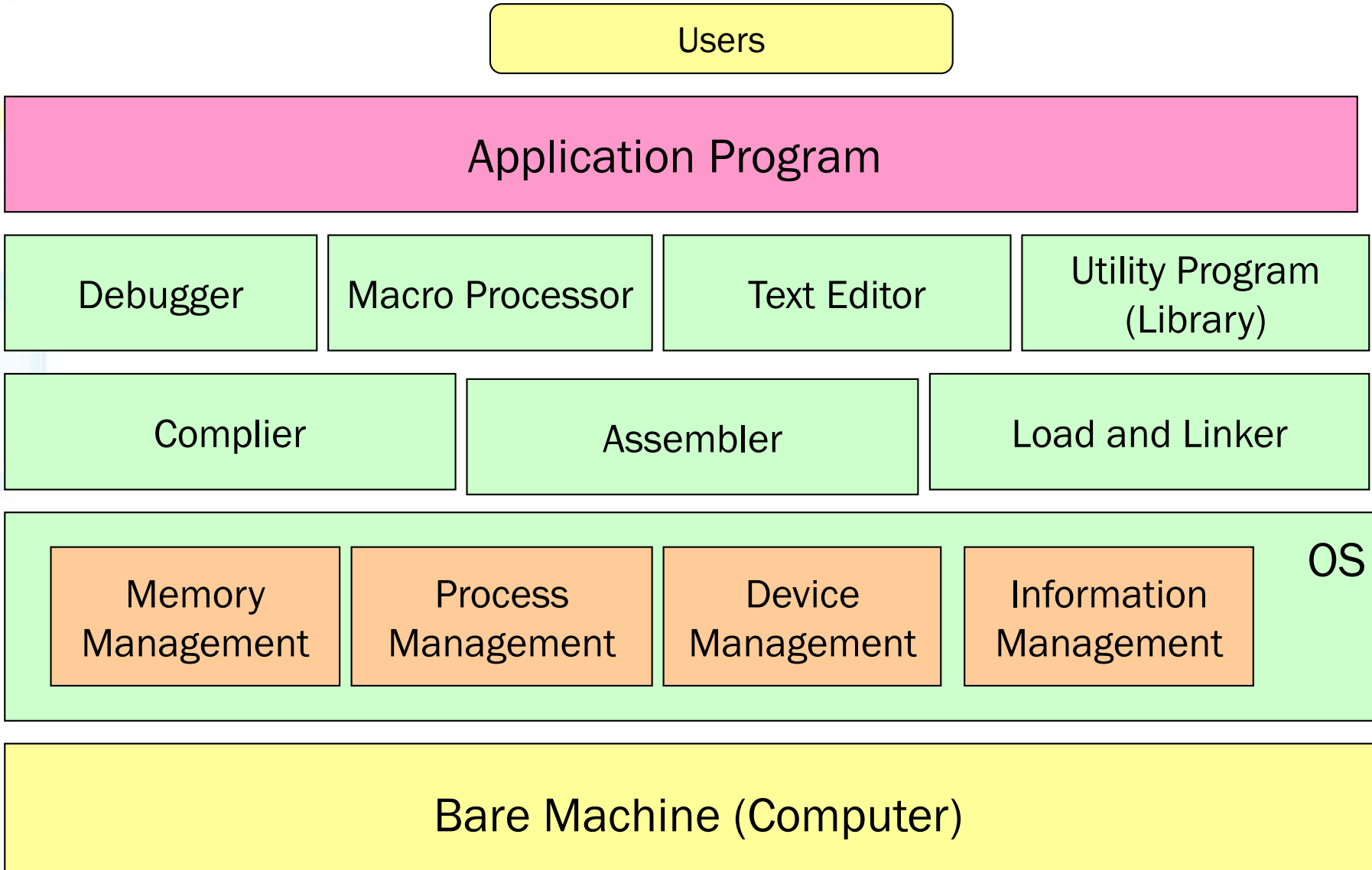
1.1 INTRODUCTION

- System Software consists of a variety of programs that support the operation of a computer.
- The programs implemented in either software and (or) firmware that makes the computer hardware usable.
- The software makes it possible for the users to focus on an application or other problem to be solved, without needing to know the details of how the machine works internally.

SYSTEM SOFTWARE

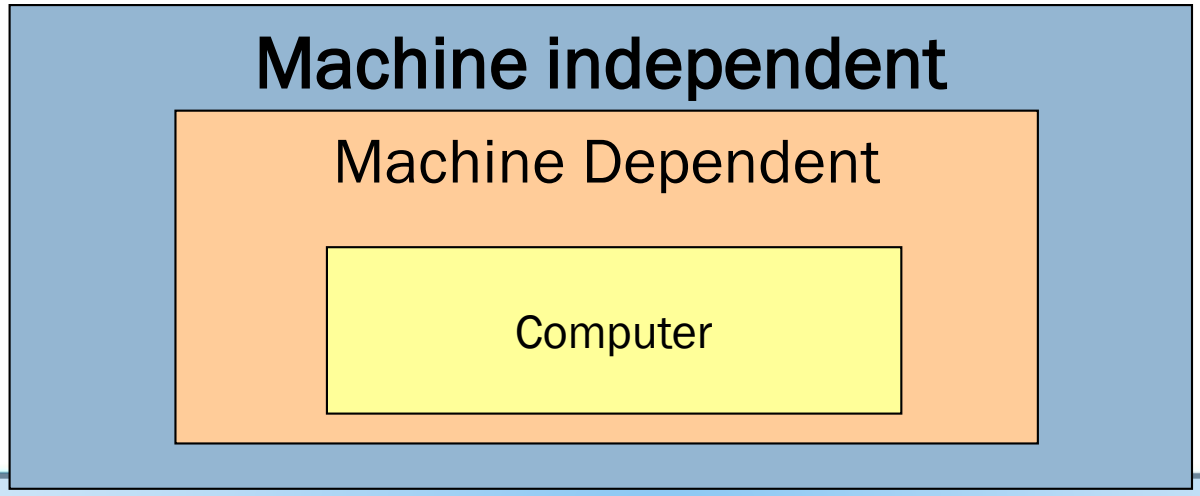
- **Text editor**
  - To create and modify the program
- **Compiler and assembler**
  - You translated these programs into machine language
- **Loader or linker**
  - The resulting machine program was loaded into memory and prepared for execution
- **Debugger**
  - To help detect errors in the program

SYSTEM SOFTWARE CONCEPT



SYSTEM SOFTWARE AND MACHINE ARCHITECTURE

- Machine dependent
- Instruction Set, Instruction Format, Addressing Mode, Assembly language …
- Machine independent General design logic/strategy, Two passes assembler…



SYSTEM SOFTWARE

- The system software includes
  - Assembler
  - Linker
  - Loader
  - Macro processor
  - Text editor
  - Compiler
  - Operating system
  - Debugging system
  - Source Code Control System
  - (optional) Database Management System

Email: nibaran@cse.jdvu.ac.in

WHY ASSEMBLY?????

- It's ugly and you'll never write it most likely
- *BUT*

  key to
  - understanding the compiler and the system interface
  - interface with the operating system , processor, bios
  - clarifies how a program access devices and execute instructions and process data
  - Showing data representation in memory
  - Precisely track the flow of data and execution

WHY ASSEMBLY .....

Added advantages

- Suitable for different boot loader/device drivers programming
- Reverse engineering
- Time and memory critical programming like embedded system, real time system etc.
- Debugging of programs become easy for analyzing and optimizing purpose

ADVANTAGES

- Some advantages to coding Assembly language are
  - Provides more control over handling particular hardware requirement
  - Generates smaller, more compact executable modules
  - More likely results in faster execution
- Common practice is to combine the benefits of both programming levels: code the bulk project in a high level language, and code critical modules those that cause noticeable delays) in assembly language

8086 ARCHITECTURE

Nibaran Das

Email: nibaran@cse.jdvu.ac.in

ARCHITECTURE

- The internal structure of the 8086 consisted of two separate internal processing units.
  - The Bus Interface Unit (BIU)
    - responsible for performing all bus operations, such as instruction fetching, reading and writing operands for memory, and inputting or outputting of data for peripherals

THE EXECUTION UNIT (EU)

- EU
  - responsible for executing instructions.

- Both units operate asynchronously to give the 8086 an overlapping instruction fetch and instruction execution mechanism. In essence, this parallel processing of the BIU and EU eliminates the time needed to fetch many of the instructions. This results in efficient use of the system bus and significantly improved system performance

BUS INTERFACE UNIT

- The bus interface unit is the 8086's interface to the outside world. It provides a full 16-bit bidirectional data bus and 20-bit address. The bus interface unit is responsible for performing all external bus operations. Specifically, it has the following functions:
  - instruction fetch
  - instruction queuing
  - operand fetch and storage
  - address relocation
  - bus control.
- The BIU is also responsible for generating bus control signals such as those for memory read or write and I/O read or write. These signals are needed for control of the circuits in the memory and I/O subsystems.

EXECUTION UNIT

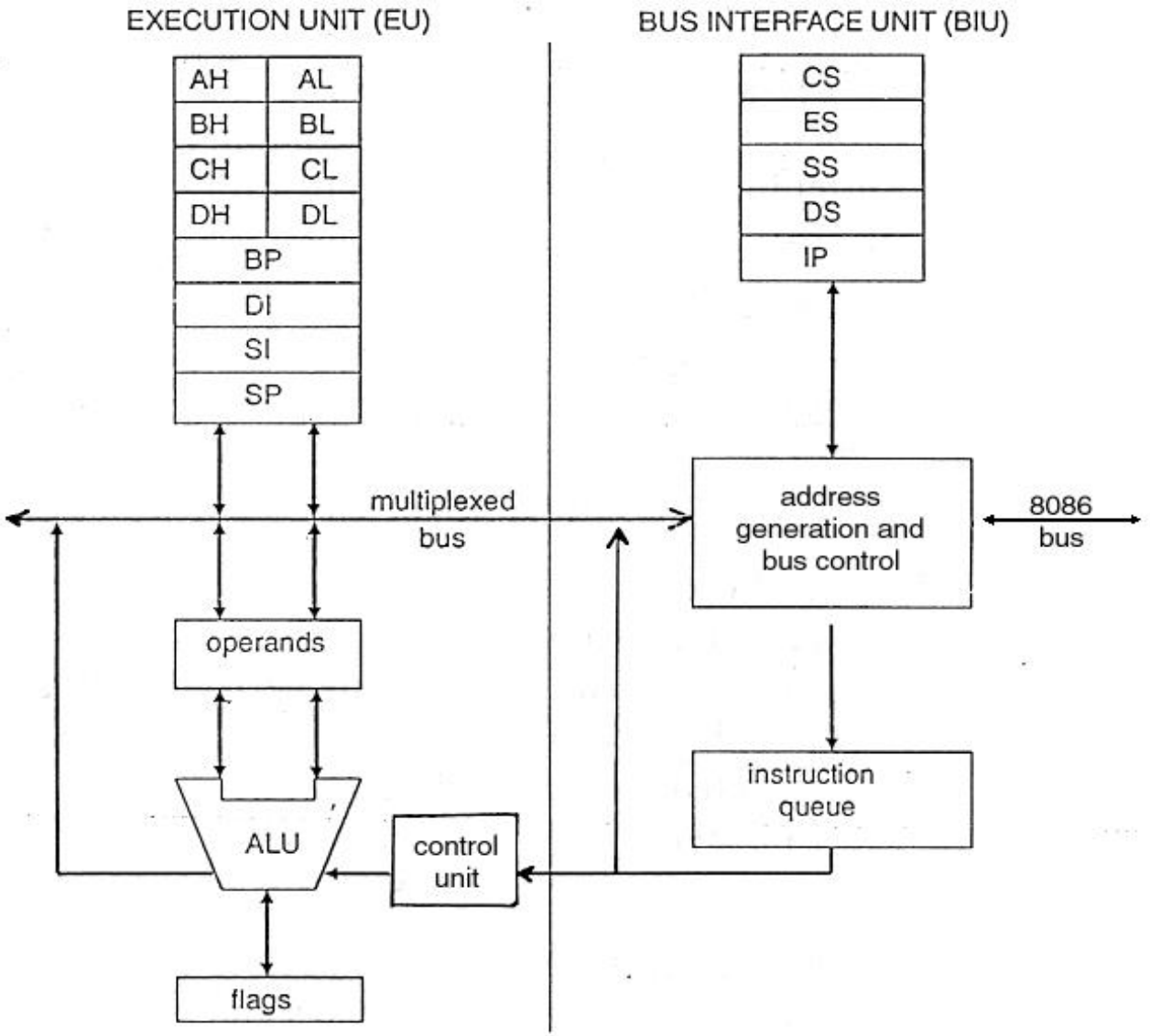
- The execution unit is responsible for *decoding* and *executing* all instructions. It consists of an *ALU*, *status and control flags*, *eight general-purpose registers*, *temporary registers*, and *queue control logic*
- *The EU extracts instructions from the top of the queue in the BIU, decodes them, generates operand addresses if necessary, passes them to the BIU and requests it to perform the read or write bus cycles to memory or I/O, and performs the operation specified by the instruction on the operands.*

EXECUTION UNIT....

- *During execution of the instruction, EU tests the status and control flags and updates them based on the results of executing the instruction. If the queue is empty, the EU waits for the next instruction byte to be fetched and shifted to the top of the queue.*

- *When the EU executes a branch or jump instruction, it transfers control to a location corresponding to another set of sequential instructions. Whenever this happens, the BIU automatically resets the queue and then begins to fetch instructions from this new location to refill the queue.*

INTERNAL BLOCK DIAGRAM OF 8086/8088



SEGMENT REGISTERS

- CS Code Segment
  - 16-bit number that points to the active code-segment
- DS Data Segment
  - 16-bit number that points to the active data-segment
- SS Stack Segment
  - 16-bit number that points to the active stack-segment
- ES Extra Segment
  - 16-bit number that points to the active extra-segment

POINTER REGISTERS

- IP Instruction Pointer
  - 16-bit number that points to the offset of the next instruction
- SP Stack Pointer
  - 16-bit number that points to the offset that the stack is using
- BP Base Pointer
  - used to pass data to and from the stack

GENERAL-PURPOSE REGISTERS

- AX Accumulator Register
  - mostly used for calculations and for input/output
- BX Base Register Only register that can be used as an index
- CX Count Register
  - register used for the loop instruction
- DX Data Register
  - input/output and used by multiply and divide

INDEX REGISTERS

- SI Source Index
  - used by string operations as source
- DI Destination Index
  - used by string operations as destination

FLAGS REGISTER

Abr.	Name	bit no	Description
OF	Overflow Flag	11	indicates an overflow when *set (signed)*
DF	Direction Flag	10	used for string operations to check direction
IF	Interrupt Flag	9	if *set*, interrupt are enabled, else disabled
TF	Trap Flag	8	if *set*, CPU can work in single step mode
SF	Sign Flag	7	if *set*, resulting number of calculation is negative
ZF	Zero Flag	6	if *set*, resulting number of calculation is zero
AF	Auxiliary Carry	4	some sort of second carry flag
PF	Parity Flag	2	indicates even or odd parity
CF	Carry Flag	0	contains the left-most bit after calculations(unsigned)

STEP-BY-STEP INSTRUCTION EXECUTION

- The x86 CPUs do *not* complete execution of an instruction in a single clock cycle. The CPU executes several steps for each instruction. For example, the CPU issues the following commands to execute the mov reg, reg/memory/constant instruction:
  - Fetch the instruction byte from memory.
  - Update the IP register to point at the next byte.
  - Decode the instruction to see what it does.
  - If required, fetch a 16-bit instruction operand from memory.
  - If required, update IP to point beyond the operand.
  - Compute the address of the operand, if required (i.e., bx+xxxx) .
  - Fetch the operand.
  - Store the fetched value into the destination register

8086 BRIEF DESCRIPTION

- 8086 is a 16-bit processor
  - Has 20 address pins (16 multiplexed with data)
  - Can address a max. of 2^{20} = 1 Million locations
  - Address ranges from 00000H to FFFFFH
  - Memory is byte addressable - Every byte has a separate address.

SEGMENT AND ADDRESS REGISTER COMBINATION

- CS:IP
- SS:SP SS:BP
- DS:BX DS:SI
- DS:DI (for other than string operations)
- ES:DI (for string operations)

FLAG REGISTER

- Flag is a bit of special information
- 9 Flags
  - 6 Status Flags
    - Cy, Ac, S, Z, P, Overflow
  - 3 Control flags
    - IE, T, D

ADDRESSING MODES

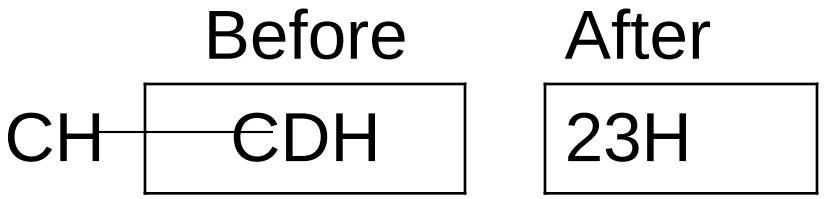
8086 ADDRESSING MODES FOR ACCESSING DATA

- Immediate Addressing mode (for source operand only)
- Register Addressing
- Memory Addressing
- I/O port addressing

IMMEDIATE ADDRESSING MODE

Ex. 1: MOV DX, 1234H

Ex. 2: MOV CH, 23H



- In Immediate Addressing mode the data follows immediately after the opcode

REGISTER ADDRESSING MODE

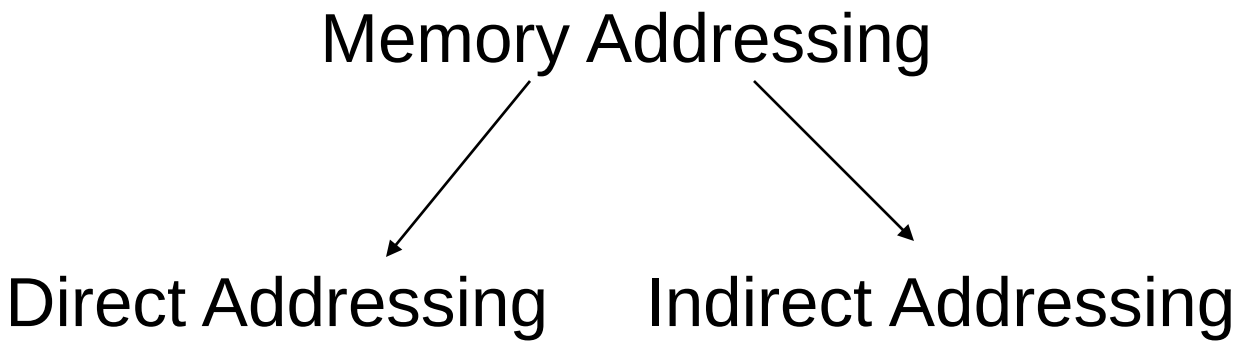
Ex.1: MOV CX, SI

	Before	After
CX	~~1234H~~	5678H
SI	5678H	5678H

Ex.2: MOV DL, AH

	Before	After
DL	~~89H~~	BCH
AH	BCH	BCH

MEMORY ADDRESSING MODE



MEMORY DIRECT ADDRESSING

Ex.1: MOV BX, DS:5634H

	Before	After
BX	ABCDH	8645H
DS:5634H	45H	LS byte
DS:5635H	86H	MS byte

MEMORY DIRECT ADDRESSING CONTD.

Ex.2: MOV CL, DS:5634H

	Before	After
CL	F2H	45H
DS:5634H	45H	
DS:5635H	86H	

MEMORY DIRECT ADDRESSING CONTD.

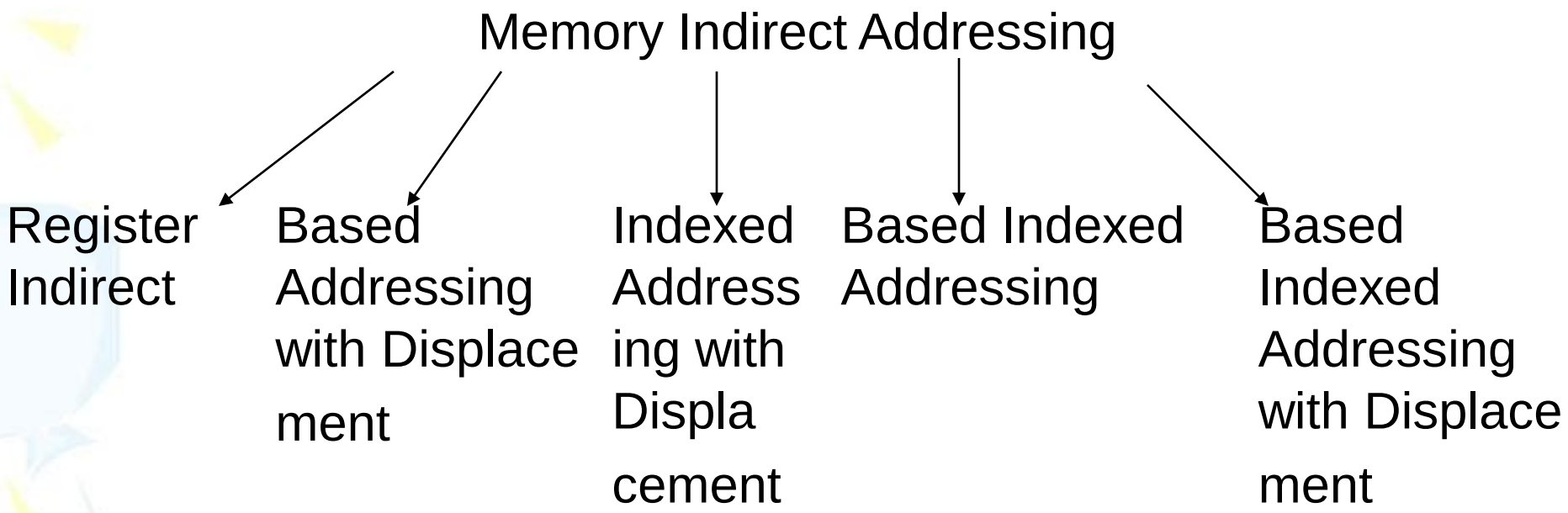
		Before	After
Ex. 3: MOV BH, LOC	BH	C5H	78H

Program

.DATA

LOC DB 78H

MEMORY INDIRECT ADDRESSING



REGISTER INDIRECT ADDRESSING

Ex.1: MOV CL, [SI]

	Before	After
CL	20H	78H
SI	3456H	
DS:3456H	78H	

REGISTER INDIRECT ADDRESSING CONTD.

Ex.2: MOV DX, [BX]

	Before	After
DX	F232H	3567H
BX	A2B2H	
DS:A2B2H	67H	LS byte
DS:A2B3H	35H	MS byte

REGISTER INDIRECT ADDRESSING CONTD

EX.3: MOV AH, [DI]

	Before	After
AH	30H	86H
DI	3400H	
DS:3400H	86H	

Only SI, DI , and BX can be used inside [] from memory addressing point of view. From user point of view [BP] is also possible. Provides 3 ways of addressing a memory operand

BASED ADDRESSING WITH DISPLACEMENT

		Before	After
Ex.1: MOV DH, 2345H[BX]	DH	45H	67H
2345H is 16 bit displacement	BX	4000H	
4000H+2345H = 6345H	DS:6345H	67H	

BASED ADDRESSING WITH DISPLACEMENT CONTD.

Ex.2: MOV AX, 45H[BP]

45H is 8 bit displacement

3000H + 45H = 3045H

It is SS when BP is used

		Before	After	
AX		1000H	CDABH	
BP		3000H		
	SS:3045H	ABH		LS byte
	SS:3046H	CDH		MS byte

BASED ADDRESSING WITH DISPLACEMENT CONTD.

- Base register can only be BX or BP
- Base Addressing with displacement provides 4 ways of addressing an operand in memory

INDEXED ADDRESSING WITH DISPLACEMENT

Ex.1: MOV CL, 2345H[SI]

2345H is 16 bit displacement

6000H+2345H = 8345H

	Before	After
CL	60H	85H
SI	6000H	
DS:8345H	85H	

INDEXED ADDRESSING WITH DISPLACEMENT CONTD.

Ex.2: MOV DX, 37H[DI]

37H is 8 bit displacement

5000H + 37H = 5037H

	Before	After
DX	7000H	B2A2 H
DI	5000H	
DS:5037H	A2H	LS byte
DS:5038H	B2H	MS byte

INDEXED ADDRESSING WITH DISPLACEMENT CONTD.

- Index register can only be SI or DI
- Indexed Addressing with displacement provides 4 ways of addressing an operand in memory

BASED INDEXED ADDRESSING

Ex1:MOV CL,[SI][BX]

	Before	After
CL	40H	67H
SI	2000H	
BX	0300H	
DS:2300H	67H	

2000H+0300H=2300H

BASED INDEXED ADDRESSING CONTD.

Ex2:MOV CX,[BP][DI]

	Before	After
CX	6000H	6385H
BP	3000H	
DI	0020H	

3000H+0020H=3020H

It is SS when BP is used

SS:3020H	85H	LS byte
SS:3021H	63H	MS byte

BASED INDEXED ADDRESSING CONTD

- Based Index Addressing Provides 4 ways of addressing an operand in memory
- One Register must be a Base register and the other must be an Index register
- For ex. MOV CX, [BX][BP] is an invalid instruction

BASED INDEXED ADDRESS - ING WITH DISPLACEMENT

Ex1: MOV DL, 37H[BX+DI]

37H is 8-bit Displacement

2000H+0050H+37H=2087H

	Before	After
DL	40H	12H
BX	2000H	
DI	0050H	
DS:2087H	12H	

BASED INDEXED ADDRESSING WITH DISPLACEMENT CONTD.

Ex2: MOV BX,1234H[SI+BP]

1234 is 16bit Displacement

4000+0020+1234=5254H

It is SS when BP is used

		Before	After
BX		3000H	3665H
SI		4000H	
BP		0020H	
SS:5254H		65H	LS byte
SS:5255H		36H	MS byte

BASED INDEXED ADDRESSING WITH DISPLACEMENT CONTD.

- Provides 8 ways of addressing an operand in memory

MEMORY MODES AS DERIVATIVES OF BASE INDEX ADDRESSING WITH DISP.

Instruction	Base Reg.	Index Reg.	Disp	Addressing Mode
MOV BX, DS:5634H	No	No	Yes	Direct Addressing
MOV CL,[SI]	No	Yes	No	Register Indirect
MOV DX,[BX]	Yes	No	No	
MOV DH, 2345H[BX]	Yes	No	Yes	Base addr. with Disp.
MOV DX, 35H[DI]	No	Yes	Yes	Index addr. with Disp.
MOV CL, 37H[SI+BX]	Yes	Yes	yes	Based Index Addr. with disp
MOV DL, 37H[BX+DI]	Yes	Yes	Yes	Base Index Addr with disp

INPUT/OUTPUT PORT ADDRESSING

I/O port Addressing

- Fixed port addressing or Direct Port addressing
- Variable port addressing or Indirect Port addressing

FIXED PORT ADDRESSING

Ex.1: IN AL, 83H

		Before	After
AL		34H	78H
Input port no. 83H		78H	

FIXED PORT ADDRESSING CONTD.

Ex.2: IN AX, 83H

		Before	After
AX		5634H	F278H
Input port no. 83H		78H	
Input port no. 84H		F2H	

FIXED PORT ADDRESSING CONTD.

Ex.3: OUT 83H, AL

	Before	After
AL	50H	
Output port no. 83H	65H	50H

FIXED PORT ADDRESSING CONTD.

Ex.4: OUT 83H, AX

	Before	After
AX	6050H	
Output port no. 83H	65H	50H
Output port no. 84H	40H	60H

FIXED PORT ADDRESSING CONTD.

- IN and OUT instructions are allowed to use only AL or AX registers
- Port address in the range 00 to FFH is provided in the instruction directly

VARIABLE PORT ADDRESSING

- I/O port address provided in DX register only
- Port addresses range from 0000H to FFFFH
- Data transfer with AL or AX only

VARIABLE PORT ADDRESSING CONTD.

Ex. 1: IN AL, DX

	Before	After
AL	30H	60H
DX	1234H	
Input port no. 1234H	60H	

VARIABLE PORT ADDRESSING CONTD.

Ex. 2: IN AX, DX

	Before	After
AX	3040H	7060H
DX	4000H	
Input port no. 4000H	60H	
Input port no. 4001H	70H	

VARIABLE PORT ADDRESSING CONTD.

Ex. 3: OUT DX, AL

	Before	After
AL	65H	
DX	5000H	
Output port no. 5000H	80H	65H

VARIABLE PORT ADDRESSING CONTD.

Ex. 4: OUT DX, AX

	Before	After
AX	4567H	
DX	5000H	
Output port no. 5000H	25H	67H
Output port no. 5001H	36H	45H

DEBUG

- It is a DOS program allows to view memory to enable programs in memory, and to trace their Execution
- Debug provides a set of commands that perform a number of useful instruction. Among them following 11 are important one

DEBUG CONT.

A	assemble symbolic instruction into machine code
D	display the contents of an area of memory
E	Enter data into memory beginning at specific memory location
G	Run the executable program in memory (G means go)
N	name of the program

DEBUG CONT.

P	proceed or execute a set of related instruction
Q	quit the debug session
R	Display the contents of one or more register
T	Trace the execution of one instruction
U	unassembled (really disassemble) machine code into symbolic code
W	write a program onto disk

MEMORY MODEL

Model	Memory
Tiny	Code and data fit into 64k. The program can made into .com file. This model is also called *Small Impure*
Small	All data file in one 64 k segment. All code fits in one 64 k segment. Maximum program size is 128 k
Medium	All data fits in one 64 k segment but code may greater than 64k
Compact	Data may be greater than64k(*but no single array may be greater*), code must be less than 64 k
Large	Both data and code may be greater than 64k, but no single array may be.
Huge	Data, Code and data array must be greater than 64k

DEBUG PROGRAM

- Write an assembly language program to convert a letter from upper uppercase to lower case

```
MOV AH,01
INT 21
ADD AL,20
MOV DL, AL
MOV AH,02
INT 21
INT 20
```

FILE WRITING IN DEBUG

- N TEST1.COM
- R BX
- 0 // Initialize BX as 0 because debug uses BX:CX for the length of your file. So normally CX value is given .
- R CX
- File length
- W

INTERRUPTS

- An ***interrupt*** is a break in the flow of execution of program
  - the CPU is “interrupted”
- When an interrupt occurs, the CPU deals with the interruption, then carries on where it left off

TYPES OF INTERRUPTS

- Hardware
  - an external signal is applied to the NMI input pin or the INTR input pin
    - NMI - non-maskable interrupt
    - INTR - interrupt
    - used to deal with I/O devices needing attention
- Software
  - execution of the interrupt instruction…INT
  - some error condition is produced by the execution of an instruction
    - such as divide by zero

THE INT INSTRUCTION

- On Execution, an INT interrupts processing and accesses the interrupt services table in low memory to determine the address of the required routine.
- The operation then transfers to DOS or to BIOS for specified action and returns to main program to resume the processing
- *Interrupts require a trail that facilitates exiting a program and, on successful completion , returning to it*.

INT PERFORMS THE FOLLOWING STEPS

- Decrement the stack pointer by 2 and pushes the contents of flags register onto the stack
- Clears the interrupt and trap flags
- Decrements the stack pointer by 2 and pushes the CS register onto the stack
- Decrements the stack pointer by 2 and pushes the instruction pointer onto the stack
- Causes the required operation to be performed

To return from an interrupt, the routine issues an IRET(interrupt return) ,which pops the registers of the stack and returns to the instruction immediately following the INT in the program

COMMON INTERRUPTS

- **INT 21h / AH=1** - read character from standard input, with echo, result is stored in **AL**.
  if there is no character in the keyboard buffer, the function waits until any key is pressed.

  example: mov ah, 1
  int 21h

COMMON INTERRUPTS CONTD.

- **INT 21h / AH=2** - write character to standard output.
  entry: **DL** = character to write, after execution **AL = DL**.

  example: mov ah, 2 mov dl, 'a' int 21h

- **INT 21h / AH=9** - output of a string at **DS:DX**. String must be terminated by '**$**'.

example:

- org 100h mov dx,
- offset msg mov ah,
- 9 int 21h
- ret
- msg db "hello world $"

DIFFERENCES BETWEEN .EXE AND .COM FILE

	.exe	.com
Program Size	Virtually any size no restriction	Restricted to 64 k maximum
PSP block size (immediately load with program during execution)	512 byte	256 byte
segments	1.Stack segment may be defined 2.Usually defines data segment and initialized DS register with the address of that segment.	1. automatically generate Stack segment 2. Do not define data segment
Initialization	May be reinitialized	Always initialized at 100h offset address

PROGRAM

- Write an assembly language program to take a character from the key board and print it
  - MOV AH,01
  - Int 21
  - Mov dl,al
  - Mov ah,02
  - Int 21

8086 INSTRUCTION SET (79 BASIC INSTRUCTIONS)

- AAA AAD AAM AAS ADC ADD AND
- CALL CBW CLC CLD CLI CMC CMP CMPS CMPXCHG
- CWD DAA DAS DEC DIV ESC HLT IDIV
- IMUL IN INC INT INTO IRET/IRETD
- Jxx JCXZ/JECXZ JMP
- LAHF LDS LEA LES LOCK LODS LOOP
- MOV MOVS LOOPE/LOOPZ LOOPNZ/LOOPNE
- MUL NEG NOP NOT OR OUT POP POPF/POPFD PUSH
- RCL RCR PUSHF/PUSHFD
- REP REPE/REPZ RET/RETF REPNE/REPNZ
- ROL ROR SAHF SAL/SHL SAR SBB SCAS SHL SHR
- STC STD STI STOS SUB
- TEST WAIT/FWAIT XCHG XLAT/XLATB XOR

INSTRUCTION TYPES

- Instructions vary from one CPU to another, General groupings possible:
  - Arithmetic/Logic
  - – Add, Subtract, AND, OR, shifts
  - – Performed by ALU
  - Data Movement
  - – Load, Store (to/from registers/memory)
  - Transfer of Control
  - – Jump, Branch, procedure call
  - Test/Compare
  - – Set condition flags
  - Input/Output
  - – In, Out (Only on some CPUs)
  - Others
  - – Halt, NOP

Nibaran Das
nibaran@cse.jdvu.ac.in

INTEGER ADDITION & SUBTRACTION

Instruction	Operation	Notes
ADD dst, src	dst := dst + src	Addition
ADC dst, src	dst := dst + src	Addition with carry flag
SUB dst, src	dst := dst - src	Subtraction
SBB dst, src	dst := dst - src	Subtraction with Borrow
CMP dst, src	dst - src	Compare & Set FLAGS
INC opr	opr := opr + 1	Increment by 1
DEC opr	opr := opr - 1	Decrement by 1
NEG opr	opr := - opr	Negate

- **Operands can be byte, word or doubleword (dword) sized**
- **Arithmetic instructions also set Flag bits, e.g. the Zero Flag (ZF),**
- **the Sign Flag (SF), the Carry Flag (CF), the Overflow Flag (OF)**
- **which can be tested with branching instructions.**

Nibaran Das
nibaran@cse.jdvu.ac.in

INTEGER MULTIPLY & DIVIDE

Instruction	Operation	Notes
IMUL/MUL opr	AX := AL * opr	Word = Byte * Byte
	DX:AX := AX * opr	Doubleword = Word * Word
	EDX:EAX := EAX * opr	Quadword = Dword* Dword
IDIV/MUL opr	AL := AX / opr	Word / Byte
	AH := AX mod opr	
	AX := (DX:AX) / opr	Doubleword / Word
	DX := (DX:AX) mod opr	

Operands must be registers or memory operands

Nibaran Das
nibaran@cse.jdvu.ac.in

ADVANTAGES

- Some advantage s to coding Assembly language are
  - Provides more control over handling particular hardware requirement
  - Generates smaller, more compact executable modules
  - More likely results in faster execution
- Common practice is to combine the benefits of both programming levels: code the bulk project in a high level language, and code critical modules those that cause noticeable delays) in assembly language

DATA ALLOCATION DIRECTIVES

db define byte
dw define word (2 bytes)
dd define double word (4 bytes)
dq define quadword (8 bytes)
dt define tenbytes
equ equate, assign numeric expression to a name

Examples:

```
db 100 dup (?)     define 100 bytes, with no initial values for bytes
db “Hello”         define 5 bytes, ASCII equivalent of “Hello”.
maxint  equ        32767
count   equ        10 * 20        ; calculate a value (200)
```

REVERSE ENGINEERING

- **Reverse engineering** (RE) is the process of discovering the technological principles of a device, object or system through analysis of its structure, function and operation. It often involves taking something (e.g., a mechanical device, electronic component, or software program) apart and analyzing its workings in detail to be used in maintenance, or to try to make a new device or program that does the same thing without utilizing any physical part of the original.

Back

Email: nibaran@cse.jdvu.ac.in