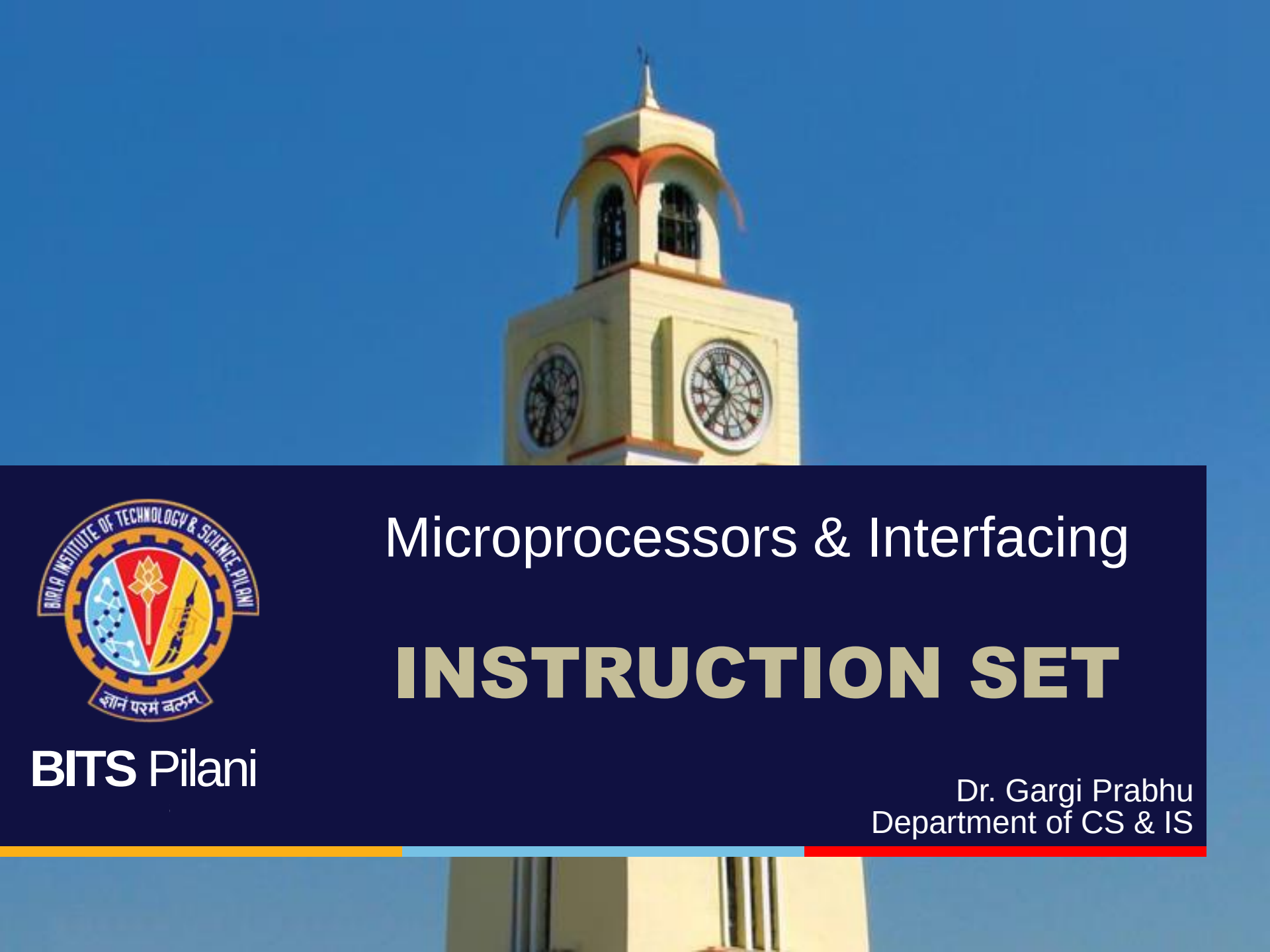

Microprocessors & Interfacing

INSTRUCTION SET

BITS Pilani

Dr. Gargi Prabhu
Department of CS & IS

Number Systems

- Negative numbers are stored as 2's complement format
- Subtraction in 8086 is done using 2's complement

E.g.

MOV CH,22H

SUB CH,44H

```
22H      00100010
44H    + 10111100
         11011110
```

$$Z = 0 \text{ (result not zero)}$$
$$C = 1 \text{ (borrow)}$$
$$A = 1 \text{ (half-borrow)}$$
$$S = 1 \text{ (result negative)}$$
$$P = 1 \text{ (even parity)}$$
$$O = 0 \text{ (no overflow)}$$

Number Systems

- Negative numbers are stored as 2's complement format
- Subtraction in 8086 is done using 2's complement

E.g.

MOV CH,22H

SUB CH,44H

22H 00100010

44H + 10111100

11011110

- AF is set if there is a borrow from bit 3 to bit 4 during the subtraction, the AF flag is set.

ASCII

- American Standard Code for Information Interchange
- Uses a 7-bit binary code to represent characters, including letters (both uppercase and lowercase), numbers, punctuation marks, and control characters.

'A': 65

'a': 97

'0': 48

'9': 57

Space: 32

Newline: 10

Carriage Return: 13

ASCII Arithmetic

- ASCII Arithmetic instructions function with ASCII-coded numbers.
- 0-9 -> 30H to 39H

Four Instructions

- AAA (ASCII adjust after Addition)
- AAD (ASCII adjust before Division)
- AAM (ASCII adjust after Multiplication)
- AAS (ASCII adjust after Subtraction)

E.g.

```
MOV AX,31H
ADD AL, 39H
AAA
ADD AX,3030H
```

Example

E.g.

MOV AX,31H

ADD AL, 39H

AAA

ADD AX,3030H

The AAA instruction clears AH if the result is less than 10, and adds 1 to AH if the result is greater than 10.

```
AX=0031 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0859 ES=0859 SS=0859 CS=0859 IP=0103 NV UP EI NG NZ NA PE NC
0859:0103 0439              ADD     AL,39
-
AX=006A BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0859 ES=0859 SS=0859 CS=0859 IP=0105 NV UP EI PL NZ NA PE NC
0859:0105 37                AAA
-
AX=0100 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0859 ES=0859 SS=0859 CS=0859 IP=0106 NV UP EI PL NZ AC PO CY
0859:0106 053030            ADD     AX,3030
-
AX=3130 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0859 ES=0859 SS=0859 CS=0859 IP=0109 NV UP EI PL NZ NA PE NC
0859:0109 0000              ADD     [BX+SI],AL
-
```

BCD

- Binary Coded Decimal. It is a way of representing decimal numbers in a binary form, where each decimal digit is represented by a fixed number of binary bits.
- In BCD, each decimal digit is encoded using a 4-bit binary representation.

  Decimal – 12

  Binary – 1100

  BCD – 0001 0010
- BCD is often used in digital systems where decimal arithmetic is required, such as in calculators and some embedded systems.
- It allows for easy conversion between binary and decimal representations.

BCD Arithmetic

- DAA (Decimal Adjust after addition)
- DAS (Decimal adjust after subtraction)
- Both the instruction correct the result so that result is also BCD

E.g.

MOV DX,1234H

MOV BX, 3099H

MOV AL,BL

ADD AL,DL

DAA

Basic Logic Instructions

- AND, OR, Exclusive-OR, and NOT
- Logic operations provide binary bit control in low-level software. The logic instructions allow bits to be set, cleared, or complemented.
- When binary data are manipulated in a register or a memory location, the rightmost bit position is always numbered bit 0.
- Bit position numbers increase from bit 0 toward the left, to bit 7 for a byte, and to bit 15 for a word.
- A doubleword (32 bits) uses bit position 31 as its leftmost bit and a quadword (64-bits) uses bit position 63 as it leftmost bit.

AND

- The AND operation performs logical multiplication
- 0 AND anything is always 0, and 1 AND 1 is always 1.

A	B	T
0	0	0
0	1	0
1	0	0
1	1	1

- The AND instruction uses any addressing mode except memory-to-memory and segment register addressing.

AND

Assembly Language	*Operation*
AND AL,BL	AL = AL and BL
AND CX,DX	CX = CX and DX
AND ECX,EDI	ECX = ECX and EDI
AND RDX,RBP	RDX = RDX and RBP (64-bit mode)
AND CL,33H	CL = CL and 33H
AND DI,4FFFH	DI = DI and 4FFFH
AND ESI,34H	ESI = ESI and 34H
AND RAX,1	RAX = RAX and 1 (64-bit mode)
AND AX,[DI]	The word contents of the data segment memory location addressed by DI are ANDed with AX
AND ARRAY[SI],AL	The byte contents of the data segment memory location addressed by ARRAY plus SI are ANDed with AL
AND [EAX],CL	CL is ANDed with the byte contents of the data segment memory location addressed by ECX

Masking

- AND operation clears bits of a binary number , this task is called masking.

```
  x x x x  x x x x    Unknown number
• 0 0 0 0  1 1 1 1    Mask
-------------------
  0 0 0 0  x x x x    Result
```

- The OR instruction uses any of the addressing modes allowed to any other instruction except segment register addressing.

Examples

```
AX=0022 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0859 ES=0859 SS=0859 CS=0859 IP=0103 NV UP EI PL NZ NA PE NC
0859:0103 BB1100            MOV     BX,0011
-
AX=0022 BX=0011 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0859 ES=0859 SS=0859 CS=0859 IP=0106 NV UP EI PL NZ NA PE NC
0859:0106 21D8              AND     AX,BX
-
AX=0000 BX=0011 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0859 ES=0859 SS=0859 CS=0859 IP=0108 NV UP EI PL ZR NA PE NC
0859:0108 3000              XOR     [BX+SI],AL
-
```

Masking Example

- ASCII-coded number can be converted to BCD using masking
- ASCII 30H to 39H -> 0 to 9

```
MOV BX,3135H          ;load ASCII
AND BX,0F0FH          ;mask BX
```

Masking Example

```
MOV BX,3135H          ;load ASCII
AND BX,0F0FH          ;mask BX
```

```
AX=3135 BX=0000 CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0859 ES=0859 SS=0859 CS=0859 IP=0103 NV UP EI NG NZ NA PE NC
0859:0103 BB0F0F            MOV     BX,0F0F
-
AX=3135 BX=0F0F CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0859 ES=0859 SS=0859 CS=0859 IP=0106 NV UP EI NG NZ NA PE NC
0859:0106 21D8              AND     AX,BX
-
AX=0105 BX=0F0F CX=0000 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000
DS=0859 ES=0859 SS=0859 CS=0859 IP=0108 NV UP EI PL NZ NA PE NC
0859:0108 0000              ADD     [BX+SI],AL
```

OR Gate

- Performs logical addition and is often called the Inclusive-OR function.

A	B	T
0	0	0
0	1	1
1	0	1
1	1	1

Masking with OR

	x x x x x x x x	Unknown number
+	0 0 0 0 1 1 1 1	Mask
	x x x x 1 1 1 1	Result

OR Examples

Assembly Language	*Operation*
OR AH,BL	AL = AL or BL
OR SI,DX	SI = SI or DX
OR EAX,EBX	EAX = EAX or EBX
OR R9,R10	R9 = R9 or R10 (64-bit mode)
OR DH,0A3H	DH = DH or 0A3H
OR SP,990DH	SP = SP or 990DH
OR EBP,10	EBP = EBP or 10
OR RBP,1000H	RBP = RBP or 1000H (64-bit mode)
OR DX,[BX]	DX is ORed with the word contents of data segment memory location addressed by BX
OR DATES[DI + 2],AL	The byte contents of the data segment memory location addressed by DI plus 2 are ORed with AL

Example

If we multiply two BCD numbers (5,7) , can you convert its result to ASCII?

Example

If we multiply two BCD numbers (5,7) , can you convert its result to ASCII?

```
MOV   AL,5              ;load data
MOV   BL,7
MUL   BL
AAM                     ;adjust
OR    AX,3030H          ;convert to ASCII
```

Exclusive-OR

- The Exclusive-OR operation excludes this condition; the Inclusive-OR includes it.

A	B	T
0	0	0
0	1	1
1	0	1
1	1	0

- The XOR instruction uses any addressing mode except segment register addressing.

XOR

- The Exclusive-OR instruction is useful if some bits of a register or memory location must be inverted.
- This instruction allows part of a number to be inverted or complemented.

	x x x x x x x x	Unknown number
⊕	0 0 0 0 1 1 1 1	Mask
	x x x x $\bar{x}$ $\bar{x}$ $\bar{x}$ $\bar{x}$	Result

XOR

- Use for the Exclusive-OR instruction is to clear a register to zero.
- E.g. XOR CH,CH
- instruction clears register CH to 00H and requires 2 bytes of memory to store the instruction.
- E.g. MOV CH, 00H instruction also clears CH to 00H, but requires 3 bytes of memory.
- Because of this saving, the XOR instruction is often used to clear a register in place of a move immediate

So far..

- The AND instruction clears (0) bits
- The OR instruction sets (1) bits
- Exclusive-OR instruction inverts bits.

- This is ideal for control system applications in which equipment must be turned on (1), turned off (0), and toggled from on to off or off to on.

Test and Bit Test Instructions

- The TEST instruction performs the AND operation.
- The difference is that the AND instruction changes the destination operand, whereas the TEST instruction does not.
- A TEST only affects the condition of the flag register, which indicates the result of the test.
- The TEST instruction uses the same addressing modes as the AND instruction.
- The TEST instruction functions in the same manner as a CMP instruction.
- The difference is that the TEST instruction normally tests a single bit (or occasionally multiple bits), whereas the CMP instruction tests the entire byte, word, or doubleword.

TEST Example

- TEST instruction is followed by either the JZ (jump if zero) or JNZ (jump if not zero) instruction.
- The destination operand is normally tested against immediate data.
- The value of immediate data is 1 to test the rightmost bit position, 2 to test the next bit, 4 for the next, and so on

```
TEST AL,1          ;test right bit
JNZ  RIGHT         ;if set
TEST AL,128        ;test left bit
JNZ  LEFT          ;if set
```

TEST

- The TEST instruction performs a bitwise AND operation between two operands without storing the result.
- It only updates the flags based on the outcome of the AND operation. The affected flags include:

Zero Flag (ZF): Set if the result of the AND operation is zero.

Parity Flag (PF): Set if the number of set bits in the result is even.

Overflow Flag (OF): Cleared (reset) to zero.

Examples

Assembly Language	*Operation*
TEST DL,DH	DL is ANDed with DH
TEST CX,BX	CX is ANDed with BX
TEST EDX,ECX	EDX is ANDed with ECX
TEST RDX,R15	RDX is ANDed with R15 (64-bit mode)
TEST AH,4	AH is ANDed with 4
TEST EAX,256	EAX is ANDed with 256

NOT & NEG

- Logical inversion, or the one's complement (NOT), and arithmetic sign inversion, or the two's complement (NEG)
- These are two of a few instructions that contain only one operand.
- NOT and NEG can use any addressing mode except segment register addressing

NOT CH	CH is one's complemented
NEG CH	CH is two's complemented
NEG AX	AX is two's complemented
NOT EBX	EBX is one's complemented
NEG ECX	ECX is two's complemented
NOT RAX	RAX is one's complemented (64-bit mode)

Thank You