CSE 4621
Microprocessor and Interfacing

Class-19

String Instructions

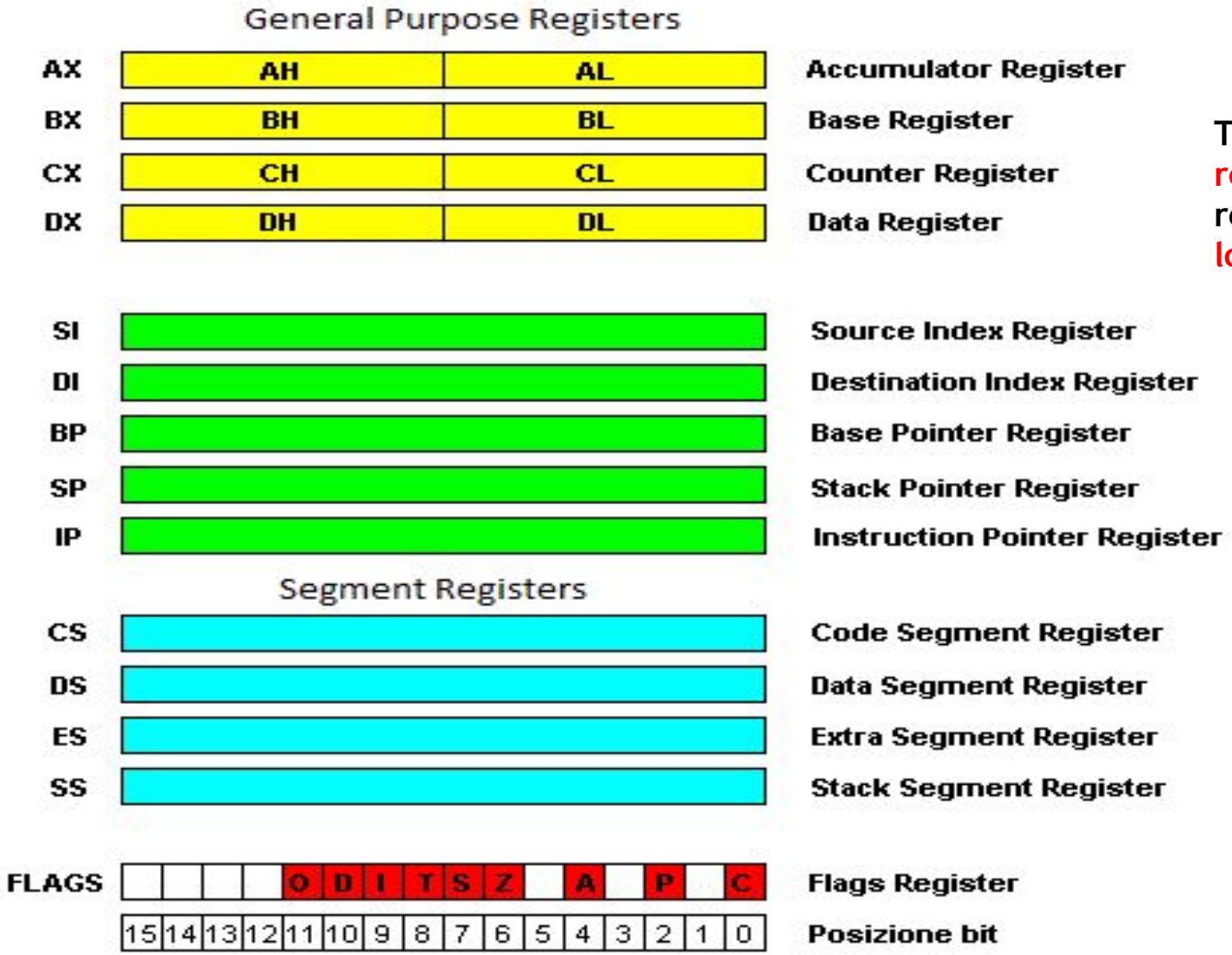
Flash back

Intel 8086 Internal Architecture: Registers

- Information inside microprocessor is stored in register
- Total Fourteen registers: All of these registers are 16-bit long
- Classified based on their functions they perform:
  1. Data Registers: hold data for operation
     - Four (4) general data registers
  2. Address Registers: hold address of data or instruction
     - Segment Register
     - Pointer Register
     - Index Register
  3. A Status Register: keep current status of the processor :FLAGS register
- Temporary register: for holding operands

Intel 8086 Internal Architecture:

Registers



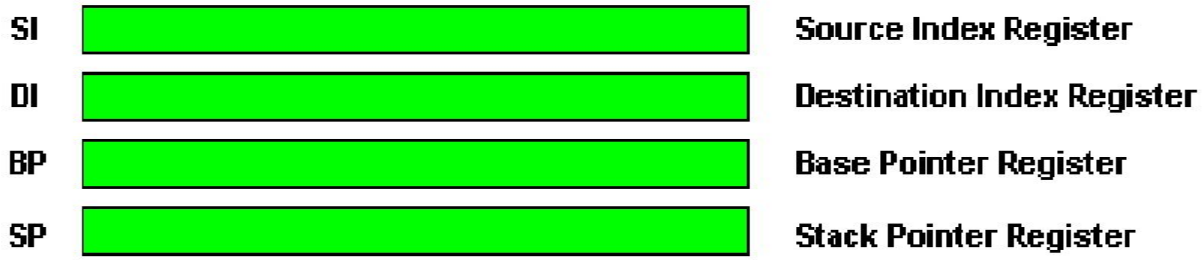
Total Fourteen registers: All of these registers are 16-bit long

Intel 8086 Internal Architecture:
Segment Registers

Memory segment	Segment register	Offset register
Code segment	Code segment Register (CSR)	Instruction Pointer (IP)
Data segment	Data segment Register (DSR)	Source index (SI)/ Destination index (DI)
Stack segment	Stack segment Register (SSR)	Stack Pointer (SP)/ Base Pointer (BP)
Extra segment	Extra segment Register (ESR)	Destination Index(DI)

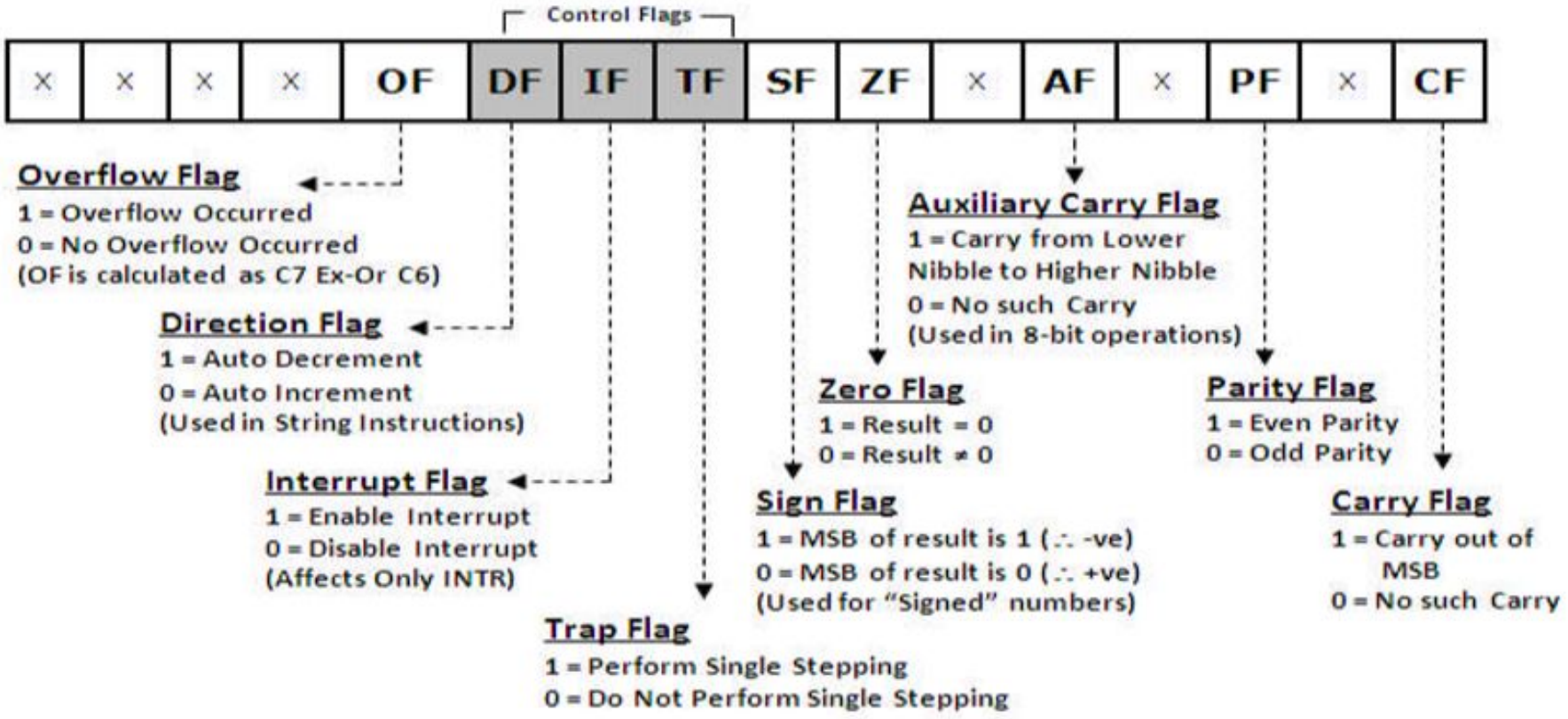
Intel 8086 Internal Architecture:
Pointer and Index Registers

- Pointer Register->points to memory in
  - Stack Segment & Code Segment
- Index Register -> points to memory in Data Segment
- Unlike segment registers, pointer and index registers can be used in arithmetic and other operations.



Intel 8086 Internal Architecture:
Flag Register

In 16 bit flag: 9 active flag



Class-19

String Instructions

String Instructions

- String instructions were designed to operate on large data structures.
- The SI and DI registers are used as pointers to the data structures being accessed or manipulated.
- The operation of the dedicated registers stated above are used to simplify code and minimize its size.

String Instructions

- The registers(DI,SI) are automatically incremented or decremented depending on the value of the direction flag:
  - DF=0, increment SI, DI.
  - DF=1, decrement SI, DI.
- To set or clear the direction flag one should use the following instructions:
  - CLD to clear the DF.
  - STD to set the DF.

String Instructions

- `movs` (move string)
  - Copy a string from one location to another
- `cmps` (compare string)
  - Compare the contents of two strings
- `scas` (scan string)
  - Search a string for one particular value
- `stos` (store string)
  - Store a value in some string position
- `lods` (load string)
  - Copies a value out of some string position

String Instructions

- The REP/REPZ/REPE/REPNZ/REPNE prefixes are used to repeat the operation it precedes.
- String instructions we will discuss:
  - LODS
  - STOS
  - MOVS
  - CMPS
  - SCAS

MOVSB/MOVSW

- Transfers the contents of the the memory byte, word or double word pointed to by SI relative to DS to the memory byte, or word pointed to by DI relative to ES. After the transfer is made, the DI register is automatically updated as follows:
  - DI is incremented if DF=0.
  - DI is decremented if DF=1.

MOVSB/MOVSW

- **Examples:**
  - **MOVSB**

    ES:[DI]=DS:[SI]; DI=DI ± 1;SI=SI ± 1

  - **MOVSW**

    ES:[DI]= DS:[SI]; DI=DI ± 2; SI=SI ± 2

MOVSB/MOVSW

Example

Assume:

Location	Content
Register SI	500H
Register DI	600H
Memory location 500H	'2'
Memory location 600H	'W'

After execution of MOVSB

If DF=0 then:

Location	Content
Register SI	501H
Register DI	601H
Memory location 500H	'2'
Memory location 600H	'2'

Else if DF=1 then:

Location	Content
Register SI	4FFH
Register DI	5FFH
Memory location 500H	'2'
Memory location 600H	'2'

STOSB/STOSW

- Transfers the contents of the AL, AX or EAX registers to the memory byte, word or double word pointed to by DI relative to ES. After the transfer is made, the DI register is automatically updated as follows:
  - DI is incremented if DF=0.
  - DI is decremented if DF=1.

STOSB/STOSW

- Examples:
  - STOSB

    ES:[DI]=AL; DI=DI ± 1
  - STOSW

    ES:[DI]=AX; DI=DI ± 2
  - STOSD

    ES:[DI]=EAX; DI=DI ± 4

STOSB/STOSW

Example

Assume:

Location	Content
Register DI	500H
Memory location 500H	'A'
Register AL	'2'

After execution of STOSB

If DF=0 then:

Location	Content
Register DI	501H
Memory location 500H	'2'
Register AL	'2'

Else if DF=1 then:

Location	Content
Register DI	4FFH
Memory location 500H	'2'
Register AL	'2'

LODSB/LODSW

- Loads the AL, AX registers with the content of the memory byte, word pointed to by SI relative to DS. After the transfer is made, the SI register is automatically updated as follows:
  - SI is incremented if DF=0.
  - SI is decremented if DF=1.

LODSB/LODSW

- Examples:
  - LODSB

    AL=DS:[SI]; SI=SI ± 1
  - LODSW

    AX=DS:[SI]; SI=SI ± 2

LODSB/LODSW

Example

Assume:

Location	Content
Register SI	500H
Memory location 500H	'A'
Register AL	'2'

After execution of LODSB

If DF=0 then:

Location	Content
Register SI	501H
Memory location 500H	'A'
Register AL	'A'

Else if DF=1 then:

Location	Content
Register SI	4FFH
Memory location 500H	'A'
Register AL	'A'

SCASB/SCASW

- Compares the contents of the AL, AX or EAX register with the memory byte, or word pointed to by DI relative to ES and changes the flags accordingly. After the comparison is made, the DI register is automatically updated as follows:
  - DI is incremented if DF=0.
  - DI is decremented if DF=1.

REP/REPZ/REPNZ

- These prefixes cause the string instruction that follows them to be repeated the number of times in the count register ECX or until:
  - ZF=0 in the case of REPZ (repeat while equal).
  - ZF=1 in the case of REPNZ (repeat while not equal).

CMPSB/CMPSW

- Compares the contents of the the memory byte, word or double word pointed to by SI relative to DS to the memory byte, or word pointed to by DI relative to ES and changes the flags accordingly. After the comparison is made, the DI and SI registers are automatically updated as follows:
  - DI and SI are incremented if DF=0.
  - DI and SI are decremented if DF=1.

REP/REPZ/REPNZ

- Use REPNE and SCASB to search for the character ‘f’ in the buffer given below.

- BUFFER DB ‘EE3751’

- MOV AL,’f’
- LEA DI,BUFFER
- MOV ECX,6
- CLD
- REPNE SCASB
- JE FOUND

REP/REPZ/REPNZ

- Use REPNE and SCASB to search for the character ‘3’ in the buffer given below.

- BUFFER DB ‘EE3751’

- MOV AL,’f’
- LEA DI,BUFFER
- MOV ECX,6
- CLD
- REPNE SCASB
- JE FOUND

`rep` prefix

- Normally used with `movs` and with `stos`
- Causes this design to be executed:

```
while count in ECX > 0 loop
    perform primitive instruction;
    decrement ECX by 1;
end while;
```

Additional Repeat Prefixes

- `repe` (equivalent mnemonic `repz`)
  - “repeat while equal” (“repeat while zero”)
- `repne` (same as `repnz`)
  - “repeat while not equal” (“repeat while not zero”)
- Each appropriate for use with `cmps` and `scas` which affect the zero flag ZF

`repe` and `repne` Operation

- Each works the same as `rep`, iterating a primitive instruction while ECX is not zero
- Each also examines ZF after the string instruction is executed
  - `repe` and `repz` continue iterating while ZF=1, as it would be following a comparison where two operands were equal
  - `repne` and `repnz` continue iterating while ZF=0

`cmps`

- Subtracts two string elements and sets flags based on the difference
- If used in a loop, it is appropriate to follow `cmps` by a conditional jump instruction
- `repe` and `repne` prefixes often used with cmps instructions

scas

- Used to scan a string for the presence or absence of a particular string element
  - String which is examined is a destination string – the address of the element being examined is in the destination index register EDI
  - Accumulator contains the element being scanned for

`stos`

- Copies a byte, a word, a doubleword or a quadword from the accumulator to an element of a destination string
- Affects no flag, so only the `rep` prefix is appropriate for use with it
  - When repeated, it copies the same value into consecutive positions of a string

`lods`

- Copies a source string element to the accumulator
- No repeat prefix is useful with `lods`
- `lods` and `stos` are often used together in a loop
  - `lods` at the beginning of a loop to fetch an element
  - `stos` at the end after the element is manipulated

Reference Book

- **Assembly Language Programming and Organization of the IBM PC, Author: Ythe Yu, Charles Marut**
  - **Chapter-11 (except section 11.6.1)**
  - **Chapter-12 (Overview)**
  - **Chapter-13 (Overview and Section 13.1)**
  - **Chapter-14 (Overview)**
  - **Chapter-15 (Overview)**