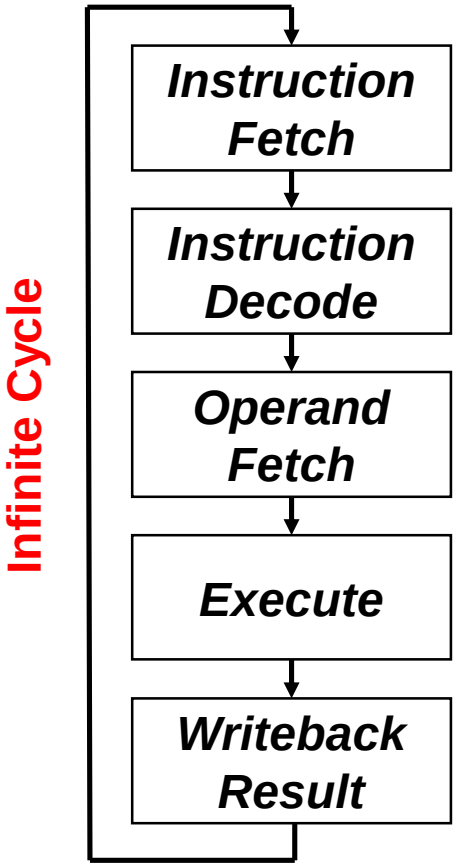
Computer Organization and Assembly Language

iAPX88 Architecture

This lecture will cover

- Instruction Execution Cycle
- Instruction Set/Groups (1.3 BH)
- Basic operand Type
- Intel iAPX88 Architecture (1.5 BH)
  - Registers (1.2, 1.6 BH)
  - Buses and Memory
- First program in Assembly (1.7 BH)
  - Introduction to DosBox, NASM, AFD

Instruction Execute Cycle



Obtain instruction from program storage

Determine required actions and instruction size

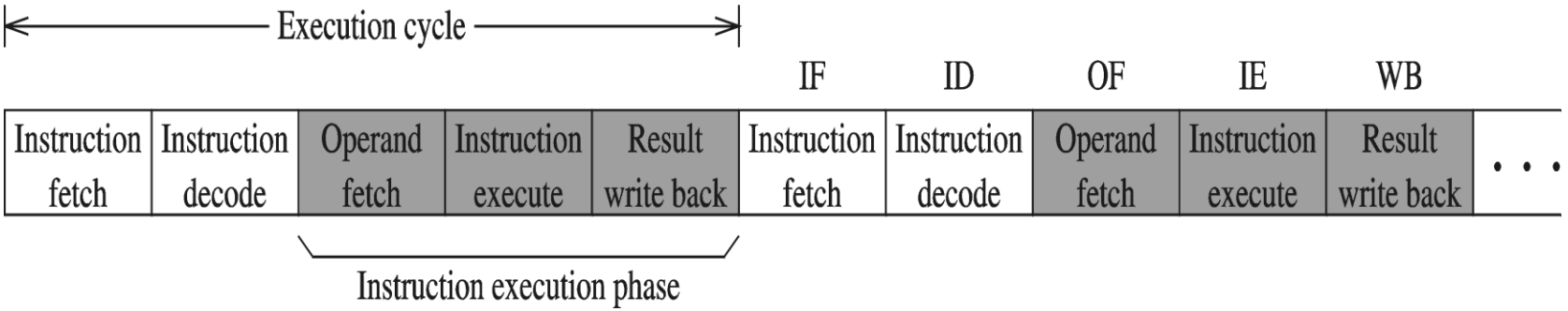
Locate and obtain operand data

Compute result value and status

Deposit results in storage for later use

Instruction Execution Cycle – cont'd

- Instruction Fetch
- Instruction Decode
- Operand Fetch
- Execute
- Result Write back



Instruction Types

- Data Movement Instructions
  - These instructions are used to move data from one place to another.
  - These places can be registers, memory, or even inside peripheral devices. Some examples are:

```
mov ax, bx
lad 1234
```

- Arithmetic and Logic Instructions
  - Arithmetic instructions like addition, subtraction, multiplication, division and Logical instructions like logical and, logical or, logical xor, or complement are part of this group.
  - Some examples are:

```
and ax, 1234
add bx, 0534
add bx, [1200]
```

Instruction Group

- Program Control Instructions
  - These are instructions that control the program execution and flow by playing with the instruction pointer and altering its normal behavior to point to the next instruction.
  - Some examples are:

```
cmp ax, 0
jne 1234
```

- Special Instructions
  - Another group called special instructions works like the special service commandos. They allow changing specific processor behaviors and are used to play with it. They are used rarely but are certainly used in any meaningful program. Some examples are:

```
cli
sti
```

  where `cli` clears the interrupt flag and `sti` sets it

Basic Operand Types

- Registers
  - Uses a named register in the CPU
  - Example

```
mov si,cx
add ax,bx
```

- Immediate operands
  - Constant expressions such as number, character constants or symbols are immediate operands, for example

```
mov ax,10
add ax,5
```

Basic Operand Types

- Direct operands
  - Refers to the content of memory at a location identified by a label in data segment, for example

```
mov ax, [num1]
num1:   dw 5
```

- Direct offset operand
  - Refers to the content of memory at a location identified by a label in data segment by +/- a value from offset, for example

```
mov ax, [num1+1]
num1:   dw 5
        dw 10
```

*Note that declaring data will be covered in coming lectures

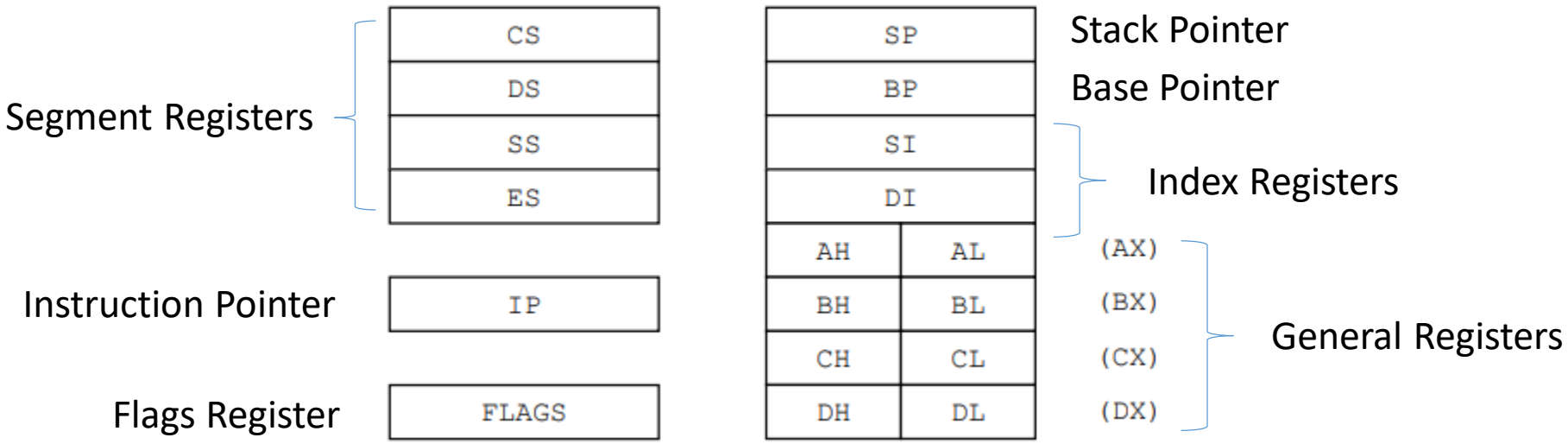
Intel iAPX88 Architecture

Registers

- Types of Registers
  - Accumulator
    - Central register in every processor
    - Traditionally all mathematical and logical operations are performed on the accumulator
    - *n* bit processor has *n* bit accumulator. For example a 16 bit processor has 16 bit accumulator.
  - Pointer, Index, or Base Register
    - Holds the address of data
  - Flag Register
    - This is a special register in every architecture called the flags register or the program status word.
    - The bits of the flags register work independently and individually, and combined its value is meaningless.
    - An example of a bit commonly present in the flags register is the carry flag
    - If a 16 bit number is added to a 16 bit accumulator, and the result is of 17 bits the 17th bit is placed in the carry bit of the flags register.
  - Program Counter or Instruction Pointer
    - The program counter holds the address of the next instruction to be executed

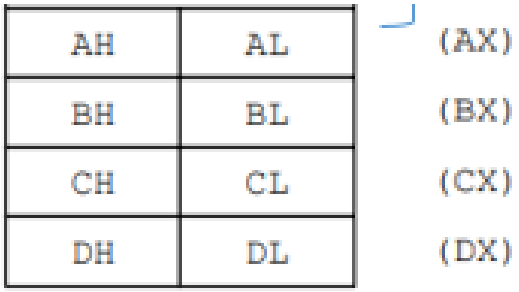
iAPX88 Register Architecture

- 8088 is a 16 bit processor with its accumulator and all registers of 16 bits
- Consists of 14 registers.



General Registers (AX, BX, CX, and DX)

- Used for arithmetic and data movement
- X in their names stand for extended meaning 16bit registers
  - For example AX means we are referring to the extended 16 bit “A” register
- Upper and lower half can be accessed separately,
  - For example
    - AH is for upper 8 bits (A high byte)
    - AL for lower 8 bits (A low byte)
    - AX is used to access whole 16 bits



  - Any changes in AH or AL are reflected in AX.

General Registers (AX, BX, CX, and DX)

- AX, A for accumulator, is the accumulator register as it is favored by CPU for arithmetic operations. Provides slightly more efficiency.
- BX, B for base, it can hold the address. BX can also perform arithmetic or data movement.
- CX, C for counter, acts as counter for repeating or loop instructions. These instructions automatically repeat and decrement CX.
- DX, D for data, has special role in multiply and divide operations, When multiplying, for example, DX holds the high 16 bits of the product.

Index Registers (SI and DI)

- SI and DI stand for source index and destination index respectively
- Hold address of data and used in memory access.
- For flexibility, Intel allows many mathematical and logical operations on these registers as well like the general registers.
- The source and destination are named because of their implied functionality as the source or the destination in a special class of instructions called the string instructions.
- SI and DI are both 16 bits and cannot be used as 8 bit register pairs like AX, BX, CX, and DX.

Instruction Pointer (IP)

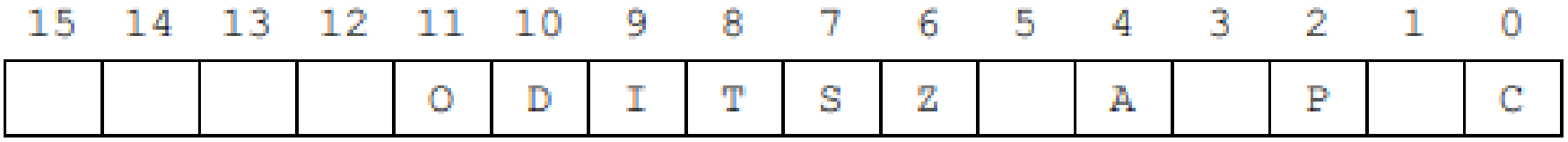
- This is the special register containing the address of the next instruction to be executed.
- No mathematics or memory access can be done through this register.
- It is out of our direct control and is automatically used.
- Playing with it is dangerous and needs special care.
- Program control instructions change the IP register

Stack Pointer (SP) and Base Pointer (BP)

- Stack Pointer (SP) is a memory pointer and is used indirectly by a set of instructions. This register will be explored in the discussion of the system stack.
- Base Pointer (BP) is also a memory pointer containing the address in a special area of memory called the stack and will be explored alongside SP in the discussion of the stack.

Flags Register

- The flags register as previously discussed is not meaningful as a unit rather it is bit wise significant and accordingly each bit is named separately. The bits not named are unused. The Intel FLAGS register has its bits organized as follows:



- The individual flags are explained in the next table.
- Empty positions in figure are undefined.

Flags Register

C	Carry	When two 16bit numbers are added the answer can be 17 bits long or when two 8bit numbers are added the answer can be 9 bits long. This extra bit that won't fit in the target register is placed in the carry flag where it can be used and tested.
P	Parity	Parity is the number of "one" bits in a binary number. Parity is either odd or even. This information is normally used in communications to verify the integrity of data sent from the sender to the receiver.
A	Auxiliary Carry	A number in base 16 is called a hex number and can be represented by 4 bits. The collection of 4 bits is called a nibble. During addition or subtraction if a carry goes from one nibble to the next this flag is set. Carry flag is for the carry from the whole addition while auxiliary carry is the carry from the first nibble to the second.
Z	Zero Flag	The Zero flag is set if the last mathematical or logical instruction has produced a zero in its destination.
S	Sign Flag	A signed number is represented in its two's complement form in the computer. The most significant bit (MSB) of a negative number in this representation is 1 and for a positive number it is zero. The sign bit of the last mathematical or logical operation's destination is copied into the sign flag.

Flags Register

T	Trap Flag	The trap flag has a special role in debugging which will be discussed later.
I	Interrupt Flag	It tells whether the processor can be interrupted from outside or not. Sometimes the programmer doesn't want a particular task to be interrupted so the Interrupt flag can be zeroed for this time. The programmer rather than the processor sets this flag since the programmer knows when interruption is okay and when it is not. Interruption can be disabled or enabled by making this bit zero or one, respectively, using special instructions.
D	Direction Flag	Specifically related to string instructions, this flag tells whether the current operation has to be done from bottom to top of the block (D=0) or from top to bottom of the block (D=1).
O	Overflow Flag	The overflow flag is set during signed arithmetic, e.g. addition or subtraction, when the sign of the destination changes unexpectedly. The actual process sets the overflow flag whenever the carry into the MSB is different from the carry out of the MSB

Segment Registers (CS, DS, SS, and ES)

- The code segment register, data segment register, stack segment register, and the extra segment register are special registers related to the Intel segmented memory model and will be discussed later.

Bus and Memory

- 8088 has 8-bit data bus
- It has 20 bit address bus
- It can access 1MB of memory with this 20 bit address bus
  - 2^{20} = 1MB

First program in Assembly language

- The first program that we will write will only add three numbers.
- The addition operation is performed on data present in registers.
- As addition requires two operands, both the operands should be present in registers (or at least one)
- The next slide shows the code with comments.

Example 1.1

```
001     ; a program to add three numbers using registers
002     [org 0x0100]
003                     mov  ax, 5              ; load first number in ax
004                     mov  bx, 10             ; load second number in bx
005                     add  ax, bx             ; accumulate sum in ax
006                     mov  bx, 15             ; load third number in bx
007                     add  ax, bx             ; accumulate sum in ax
008
009                     mov  ax, 0x4c00         ; terminate program
010                     int  0x21
```

- Things to note:
  - Abbreviations, for example move is written as mov in assembly operation
  - Code will always start with Line 2, and
  - Code will always ends with Line 9 and 10 to terminate program safely.
  - Instructions are written in following format
    - operation destination, source
  - Code is very much like English language for example
    - mov ax,5 means move 5 to ax

Assembling and Debugging code

- NASM (Netwide Assembler) will be used to assemble the code and create executable file with .com extension
- AFD (Advanced Fullscreen Debugger) will be used to debug the code.
- DOSBox will be used to emulate 8088 processor.
- Installation of these tools will be discussed in lab.
- Save the code given in previous slide with .asm extension.
- Following command will be used to assemble the code

```
nasm ex1.asm –o ex1.com –l ex1.lst
```

Listing File

- The listing file created has a lot of information. And is annotated as follows

Hex offset of instruction

instruction (opcode and operands) in hex

disassembled code

```
 1
 2                                          [org 0x0100]
 3 00000000 B80500                                mov  ax, 5
 4 00000003 BB0A00                                mov  bx, 10
 5 00000006 01D8                                  add  ax, bx
 6 00000008 BB0F00                                mov  bx, 15
 7 0000000B 01D8                                  add  ax, bx
 8
 9 0000000D B8004C                                mov  ax, 0x4c00
10 00000010 CD21                                  int  0x21
```

Notes

1: Total size of .com file is 18 bytes (add up bytes for each instruction)

2: operands are stored in little endian format

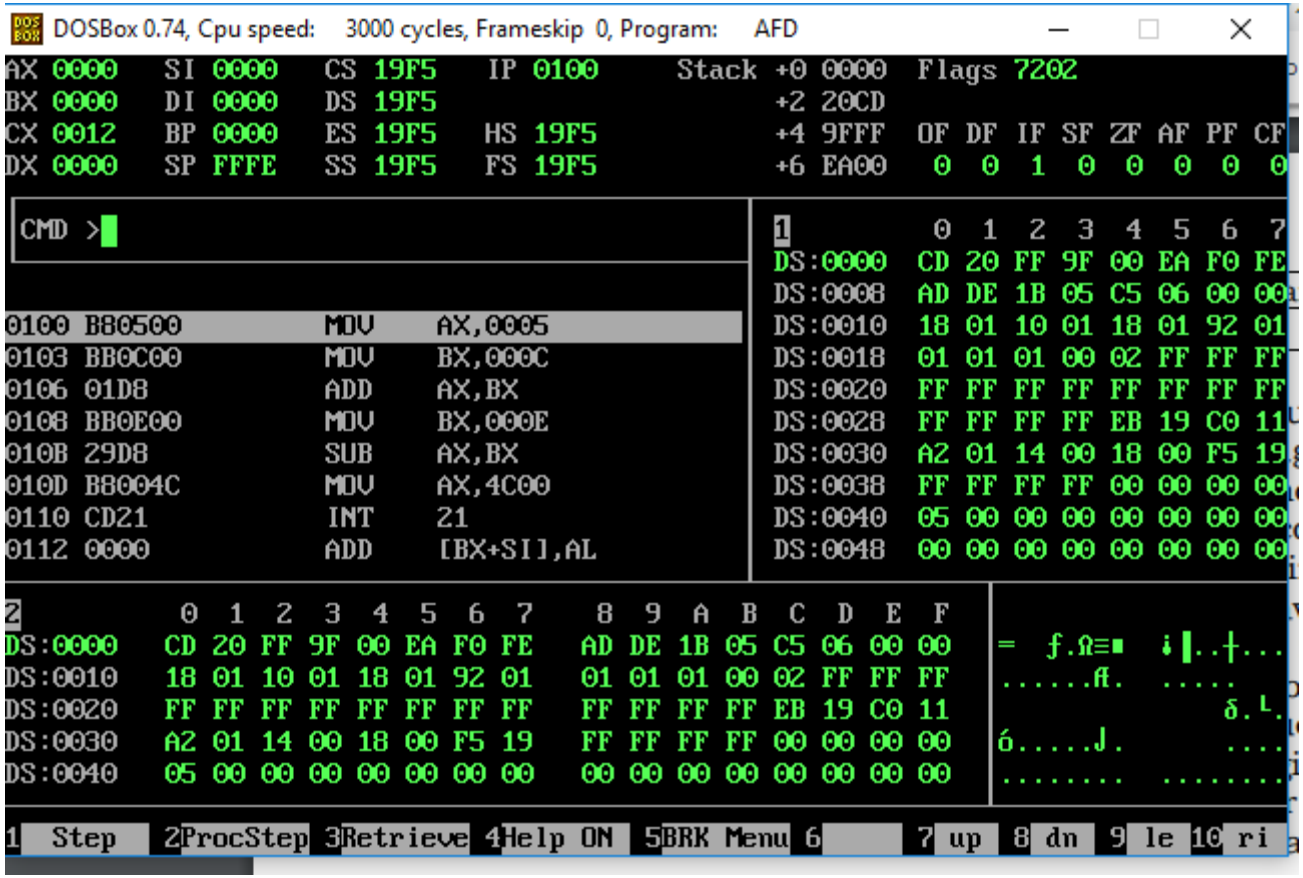
Debugging

- Following command will be used to debug the file

```
afd ex1.com
```

- The debugger shows the values of registers, flags, stack, our code, and one or two areas of the system memory as data.
- Debugger allows us to step our program one instruction at a time and observe its effect on the registers and program data.

Debugging



Things to note:

- Registers
- Memory (m1 and m2 view)
- Offset of 1st instruction
- Instruction Pointer (IP)
- Hit F1 to execute one instruction
- See the change in registers/memory
- Flags