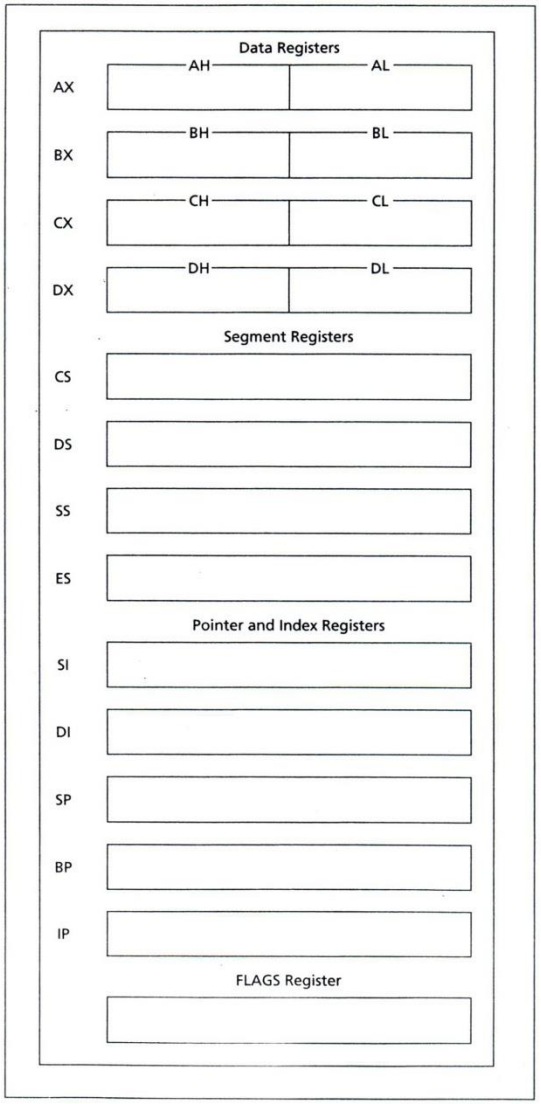
Outline

- Organization of the 8086/8088 Microprocessors
- Computer Registers
- Assembly Language Syntax
- Program Data
- Variables
- Named Constants

The 8086 and 8088 Microprocessors

- Intel introduced the 8086 in 1978 as its first 16-bit microprocessor.
- A 16-bit processor can operate on 16 bits of data at a time.
- The 8088 has an 8-bit data bus.
- Because the 8086 and 8088 essentially have the same internal structure, the name “8086” applies to both 8086 and 8088.

8086 Registers



Data Registers

- General-Purpose Registers
- Data registers are available to programmers for general data manipulation.
- Even though a processor can operate on data stored in memory, the same instruction is faster (requires fewer clock cycles) if the data are stored in registers.
- The high and low bytes of data registers can be accessed separately.

AX (Accumulator Register)

- AX is the preferred register to use in arithmetic, logic, and data transfer instructions because its use generates the shortest machine code.
- In multiplication and division operations, one of the numbers involved must be in AX or AL.
- Input and output operations also require the use of AL and AX.

BX (Base Register)

- BX also serves as an address register.

CX (Count Register)

- Program loop constructions are facilitated by the use of CX, which serves as a loop counter.
- CL is used as a count in instructions that shift and rotate bits.

DX (Data Register)

- DX is used in multiplication and division.
- DX is also used in I/O operations.

Segment Registers

- Address Registers
- Segment registers store addresses of instructions and data in memory.
- These values are used by a processor to access memory location.
- The idea of memory segments is a direct consequence of using a 20-bit address in a 16-bit processor.

Physical Address

- 8086 assigns a 20-bit physical address to its memory.
- It is possible to address 2^{20} = 1,048,576 bytes (one megabyte) of memory.

Physical Address

0000 0000 0000 0000 0000	00000h
0000 0000 0000 0000 0001	00001h
0000 0000 0000 0000 0010	00002h
0000 0000 0000 0000 0011	00003h
…	
1111 1111 1111 1111 1110	FFFFEh
1111 1111 1111 1111 1111	FFFFFh

Segment

- A memory segment is a block of 2^{16} (or 64 K) consecutive memory bytes.
- Each segment is identified by a segment number, starting with 0.
- A segment number is 16 bits, so the highest segment number is FFFFh.

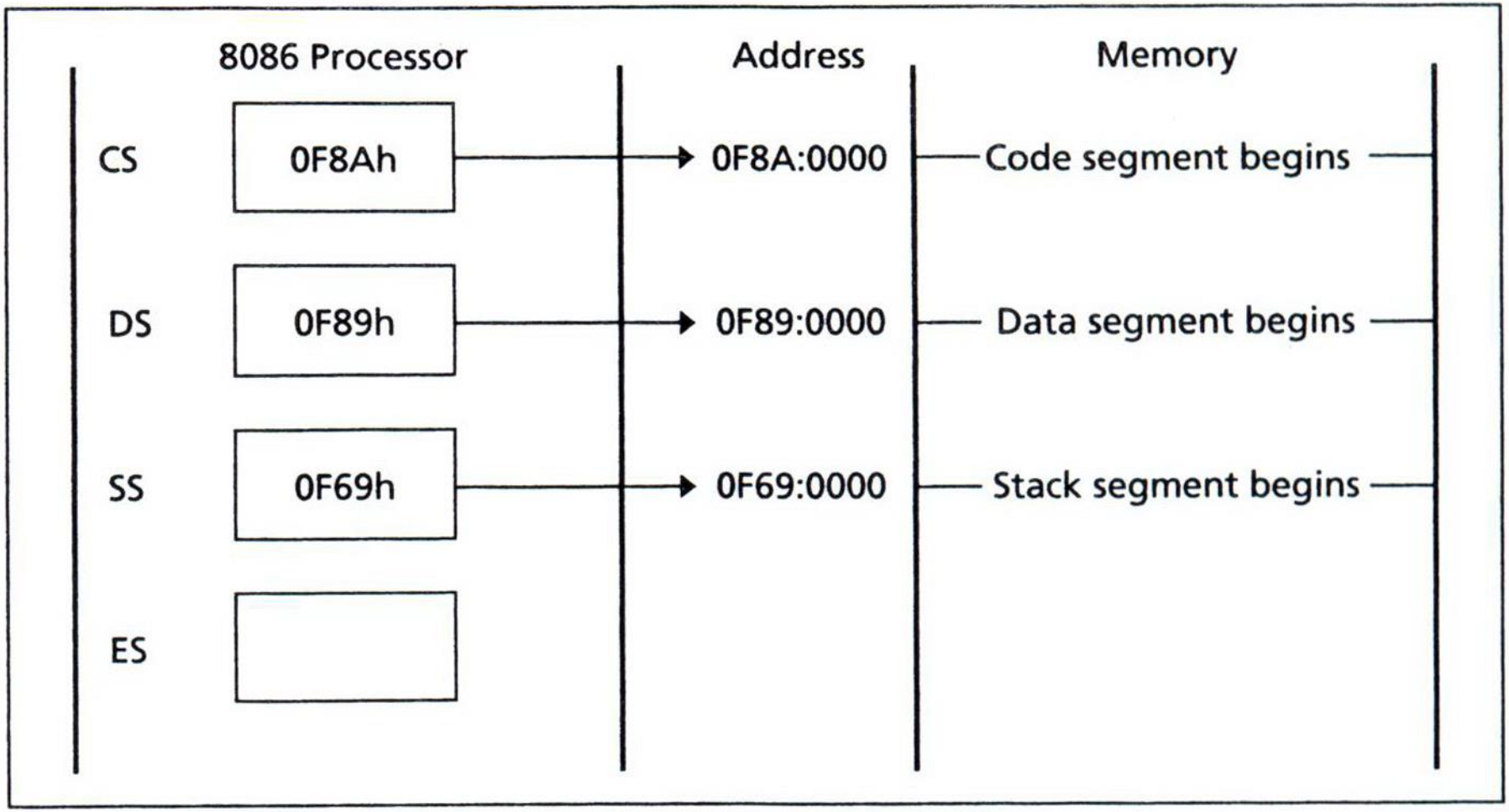
Offset

- Within a segment, a memory location is specified by giving an offset.
- An offset is the number of bytes from the beginning of the segment.
- With a 64-KB segment, the offset can be given as a 16-bit number.
- The first byte in a segment has offset 0.
- The last offset in a segment is FFFFh.

Segment:Offset Address

- Logical Address
- To obtain a 20-bit physical address, the 8086 microprocessors first shifts the segment address 4 bits to the left (this is equivalent to multiplying by 10h), and then adds the offset.
- **physical address = segment × 10h + offset**
- A4FB:4872 = A4FB0h + 4872h = A9822h

Segment Registers



Pointer and Index Registers

- SP, BP, SI, and DI normally point to (contain the offset addresses of) memory locations.
- Unlike segment registers, pointer and index registers can be used in arithmetic and other operations.

SP (Stack Pointer)

- SP is used in conjunction with SS for accessing the stack segment.

BP (Base Pointer)

- BP is used primarily to access data on the stack.
- Unlike SP, we can also use BP to access data in the other segments.

SI (Source Index)

- SI is used to point to memory locations in the data segment addressed by DS.
- By incrementing the contents of SI, we can easily access consecutive memory locations.

DI (Destination Index)

- DI performs the same functions as SI.

IP (Instruction Pointer)

- To access instructions, 8086 uses CS and IP.
- CS contains the segment number of the next instruction, and IP contains the offset.
- IP is updated each time an instruction is executed so that it will point to the next instruction.
- Unlike other registers, IP cannot be directly manipulated by an instruction; that is, an instruction may not contain IP as its operand.

FLAGS Register

- The purpose of FLAGS is to indicate the status of a microprocessor

Status Flags

- Status flags reflect the result of an instruction executed by a processor.
- When a subtraction operation results in a 0, **ZF (zero flag)** is set to 1 (true).
- A subsequent instruction can examine ZF and branch to some code that handles a zero result.

Control Flags

- Control flags enable or disable certain operations of a processor.
- If **IP (interrupt flags)** is cleared (set to 0), inputs from a keyboard are ignored by a processor.

Statements

- **name operation operand(s) comment**
- The **instruction** is translated into machine code by the assembler.
- START: MOV CX, 5 ; initialize counter
- The **assembler directive** instructs the assembler to perform some specific tasks.
- MAIN PROC

Numbers

Number	Type
11011	decimal
11011B	binary
64223	decimal
-21843D	decimal
1,234	illegal
1B4DH	hex
1B4D	illegal hex number
FFFFH	illegal hex number
0FFFFH	hex

Data-Defining Pseudo-ops

Pseudo-op	**Stands for**
DB	define byte
DW	define word
DD	define doubleword
DQ	define quadword
DT	define tenbytes

Byte Variables

- **name DB initial_value**
- ALPHA DB 4
- BYT DB ?
- -128 to 127 for signed interpretation
- 0 to 255 for unsigned interpretation

Word Variables

- **name DW initial_value**
- WRD DW -2
- -32768 to 32767 for signed interpretation
- 0 to 65535 for unsigned interpretation

Arrays

- B_ARRAY DB 10H, 20H, 30H

Symbol	**Address**	**Contents**
B_ARRAY	0200h	10h
B_ARRAY+1	0201h	20h
B_ARRAY+2	0202h	30h

Arrays

- W_ARRAY DW 1000,40,29887,329

Symbol	**Address**	**Contents**
W_ARRAY	0300h	1000d
W_ARRAY+2	0302h	40d
W_ARRAY+4	0304h	29887d
W_ARRAY+6	0306h	329d

High and Low Bytes of a Word

- WORD1 DW 1234H
- The low byte of WORD1 contains 34h, and the high byte contains 12h.
- The low byte has symbolic address WORD1, and the high byte has symbolic address WORD1+1.

Character Strings

- LETTERS DB ‘ABC’
- LETTER DB 41H, 42H, 43H
- MSG DB ‘HELLO’, 0AH, 0DH, ‘$’
- MSG DB 48H, 45H, 4CH, 4CH, 4FH, 0AH, 0DH, 24H

EQU

- The EQU (equates) pseudo-op is used to assign a name to a constant.

name	**EQU**	**constant**
LF	EQU	0AH
MOV	DL, 0AH	
MOV	DL, LF	
PROMPT	EQU	‘TYPE YOUR NAME’
MSG	DB	‘TYPE YOUR NAME’
MSG	DB	PROMPT