SOFTWARE ARCHITECTURE OF THE 8088 AND 8086 MICROPROCESSORS

2.1 Microarchitecture of the 8088/8086 Microprocessor
2.2 Software Model of the 8088/8086 Microprocessor
2.3 Memory Address Space and Data Organization
2.4 Data Types
2.5 Segment Registers and Memory Segmentation
2.6 Dedicated, Reserved, and General-Used Memory
2.7 Instruction Pointer
SOFTWARE ARCHITECTURE OF THE
8088 AND 8086 MICROPROCESSORS

2.8 Data Registers
2.9 Pointer and Index Register
2.10 Status Register
2.11 Generating a Memory Address
2.12 The Stack
2.13 Input/Output Address Space

2.1 Microarchitecture of the 8088/8086 Microprocessor

- 8088/8086 both employ parallel processing
- 8088/8086 contain two processing unit – the bus interface unit (BIU) and execution unit (EU)
- The bus interface unit is the path that 8088/8086 connects to external devices.
- The system bus includes an 8-bit bidirectional data bus for 8088 (16 bits for the 8086), a 20-bit address bus, and the signal needed to control transfers over the bus.
2.1 Microarchitecture of the 8088/8086 Microprocessor

- Components in BIU
  - Segment register
  - The instruction pointer
  - Address generation adder
  - Bus control logic
  - Instruction queue

- Components in EU
  - Arithmetic logic unit, ALU
  - Status and control flags
  - General-purpose registers
  - Temporary-operand registers

Pipeline architecture of the 8086/8088 microprocessors
2.1 Microarchitecture of the 8088/8086 Microprocessor

EU and BIU of the 8086/8088 microprocessors

2.2 Software Model of the 8088/8086 Microprocessor

- 8088 microprocessor includes 13 16-bit internal registers.
  - The instruction pointer, IP
  - Four data registers, AX, BX, CX, DX
  - Two pointer register, BP, SP
  - Two index register, SI, DI
  - Four segment registers, CS, DS, SS, ES
- The status register, SR, with nine of its bits implement for status and control flags.
- The memory address space is 1 Mbytes and the I/O address space is 64 Kbytes in length.
2.2 Software Model of the 8088/8086 Microprocessor

The software model of the 8088/8086 microprocessors is shown in the diagram provided.

2.3 Memory Address Space and Data Organization

- The 8088 microcomputer supports 1 Mbytes of external memory.
- The memory of an 8088-based microcomputer is organized as 8-bit bytes, not as 16-bit words.

<table>
<thead>
<tr>
<th>Memory address space of the 8088/8086 microcomputer</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFFFFF</td>
</tr>
<tr>
<td>FFFFE</td>
</tr>
<tr>
<td>FFFFD</td>
</tr>
<tr>
<td>FFFFC</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
2.3 Memory Address Space and Data Organization

- Lower address byte and higher address byte

<table>
<thead>
<tr>
<th>Address</th>
<th>Memory (binary)</th>
<th>Memory (hexadecimal)</th>
<th>Address</th>
<th>Memory (binary)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00725&lt;sub&gt;16&lt;/sub&gt;</td>
<td>0101 0101</td>
<td>5 5</td>
<td>0072C&lt;sub&gt;16&lt;/sub&gt;</td>
<td>11111101</td>
</tr>
<tr>
<td>00724&lt;sub&gt;16&lt;/sub&gt;</td>
<td>0000 0010</td>
<td>0 2</td>
<td>0072B&lt;sub&gt;16&lt;/sub&gt;</td>
<td>10101010</td>
</tr>
</tbody>
</table>

The two bytes represent the word 0101010100000010<sub>2</sub> = 5502<sub>16</sub>

---

EXAMPLE

What is the data word shown in the previous figure? Express the result in hexadecimal form. Is it stored at an even- or odd-addressed word boundary? Is it an aligned or misaligned word of data?

Solution:

11111101<sub>2</sub> = FD<sub>16</sub> = FDH
10101010<sub>2</sub> = AA<sub>16</sub> = AAH

Together the two bytes give the word 1111110110101010<sub>2</sub> = FDAA<sub>16</sub> = FDAAH

Expressing the address of the least significant byte in binary form gives 0072BH = 0072B<sub>16</sub> = 0000000001110010101011<sub>2</sub>

Therefore, it is misaligned word of data.
2.3 Memory Address Space and Data Organization

- Even- or odd-addressed word
  If the least significant bit of the address is 0, the word is said to be held at an even-addressed boundary.
- Aligned word or misaligned word

<table>
<thead>
<tr>
<th>Address</th>
<th>Physical memory</th>
<th>Aligned words</th>
</tr>
</thead>
<tbody>
<tr>
<td>00008H</td>
<td>Byte 8</td>
<td>Word 1</td>
</tr>
<tr>
<td>00007H</td>
<td>Byte 7</td>
<td>Word 1</td>
</tr>
<tr>
<td>00006H</td>
<td>Byte 6</td>
<td>Word 1</td>
</tr>
<tr>
<td>00005H</td>
<td>Byte 5</td>
<td>Word 1</td>
</tr>
<tr>
<td>00004H</td>
<td>Byte 4</td>
<td>Word 1</td>
</tr>
<tr>
<td>00003H</td>
<td>Byte 3</td>
<td>Word 1</td>
</tr>
<tr>
<td>00002H</td>
<td>Byte 2</td>
<td>Word 1</td>
</tr>
<tr>
<td>00001H</td>
<td>Byte 1</td>
<td>Word 1</td>
</tr>
<tr>
<td>00000H</td>
<td>Byte 0</td>
<td>Word 1</td>
</tr>
</tbody>
</table>

2.3 Memory Address Space and Data Organization

- A double word corresponds to four consecutive bytes of data stored in memory.
2.3 Memory Address Space and Data Organization

A pointer is a double word. The higher address word represents the **segment base address** while the lower address word represents the **offset**.

<table>
<thead>
<tr>
<th>Address (binary)</th>
<th>Memory (hexadecimal)</th>
<th>Address (hexadecimal)</th>
<th>Memory (hexadecimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00007₁₆</td>
<td>0011 1011</td>
<td>00008₁₆</td>
<td>A0</td>
</tr>
<tr>
<td>00006₁₆</td>
<td>0100 1100</td>
<td>0000A₁₆</td>
<td>00</td>
</tr>
<tr>
<td>00005₁₆</td>
<td>0000 0000</td>
<td>00009₁₆</td>
<td>55</td>
</tr>
<tr>
<td>00004₁₆</td>
<td>0110 0101</td>
<td>00008₁₆</td>
<td>FF</td>
</tr>
</tbody>
</table>

Example: Segment base address = 3B4C₁₆ = 0011101101001100₂
Offset value = 85₁₆ = 0000000001100101₂

---

**EXAMPLE**

How should the pointer with segment base address equal to A000₁₆ and offset address 55FF₁₆ be stored at an even-address boundary starting at 00008₁₆? Is the double word aligned or misaligned?

**Solution:**

Storage of the two-word pointer requires four consecutive byte locations in memory, starting at address 00008₁₆. The least-significant byte of the offset is stored at address 00008₁₆ and is shown as FF₁₆ in the previous figure. The most significant byte of the offset, 55₁₆, is stored at address 00009₁₆. These two bytes are followed by the least significant byte of the segment base address, 00₁₆, at address 0000A₁₆, and its most significant byte, A0₁₆, at address 0000B₁₆. Since the double word is stored in memory starting at address 00008₁₆, it is aligned.
2.4 Data Types

- Integer data type
  - Unsigned or signed integer
  - Byte-wide or word-wide integer

MSB  |  LSB
---   |  ---
D_7   |  D_0

MSB  |  LSB
---   |  ---
D_{15} |  D_0

Unsigned byte and unsigned word integer

2.4 Data Types

- The most significant bit of a signed integer is a sign bit. A zero in this bit position identifies a positive number.
- The range of a signed byte integer is +127 ~ -128. The range of a signed word integer is +32767 ~ -32768.
- The 8088 always expresses negative numbers in 2’s-complement.
2.4 Data Types

EXAMPLE

A signed word integer equals FEFF_{16}. What decimal number does it represent?

Solution:

FEFF_{16} = 111111101111111_{2}

The most significant bit is 1, the number is negative and is in 2’s complement form.

Converting to its binary equivalent by subtracting 1 from the least significant bit and then complement all bits give

FEFF_{16} = -0000000100000001_{2}

= -257

2.4 Data Types

- The 8088 can also process data that is coded as binary-coded decimal (BCD) numbers.
- BCD data can be stored in packed or unpacked forms.

<table>
<thead>
<tr>
<th>Decimal</th>
<th>BCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
</tr>
</tbody>
</table>

BCD Numbers

Packed BCD digit

MSB

LSB

BCD Digit 0

BCD Digit 1

Unpacked BCD digit

MSB

LSB

BCD Digit 0

BCD Digit 1

D3

D0

D3

D0

D3

D0

D3

D0

D3

D0

D3

D0
2.4 Data Types

EXAMPLE

The packed BCD data stored at byte address 0100016 equals 10010001_2. What is the two digit decimal number?

Solution:

Writing the value 10010001_2 as separate BCD digits gives

\[ 10010001_2 = 1001_{BCD}0001_{BCD} = 91_{10} \]

2.4 Data Types

The ASCII (American Standard Code for Information Interchange) digit
2.4 Data Types

**EXAMPLE**

Byte addresses $01100_{16}$ through $01104_{16}$ contain the ASCII data $01000001$, $01010011$, $01000011$, $01001001$, and $01001001$, respectively. What do the data stand for?

**Solution:**

Using the ASCII table, the data are converted to ASCII code:

$$(01100H) = 01000001_2 = A$$

$$(01101H) = 01010011_2 = S$$

$$(01102H) = 01000011_2 = C$$

$$(01103H) = 01001001_2 = I$$

$$(01104H) = 01001001_2 = I$$

2.5 Segment Registers and Memory Segmentation

- A **segment** represents an independently addressable unit of memory consisting of 64K consecutive byte-wide storage locations.
- Each segment is assigned a **base address** that identifies its starting point.
- Only four segments can be active at a time:
  - The code segment
  - The stack segment
  - The data segment
  - The extra segment
- The addresses of the active segments are stored in the four internal segment registers: CS, SS, DS, ES.
2.5 Segment Registers and Memory Segmentation

- Four segments give a maximum of 256Kbytes of active memory.
  - Code segment – 64K
  - Stack – 64K
  - Data storage – 128K
- The base address of a segment must reside on a 16-byte address boundary.
- User accessible segments can be set up to be contiguous, adjacent, disjointed, or even overlapping.
2.5 Segment Registers and Memory Segmentation

- **DATA: DS:**
- **CODE: CS:**
- **STACK: SS:**
- **EXTRA: ES:**

Contiguous, adjacent, disjointed, and overlapping segments

2.6 Dedicated, Reserved, and General-Used Memory

- **The dedicated memory** (00000₁₆ ~ 00013₁₆) are used for storage of the pointers to 8088’s internal interrupt service routines and exceptions.
- **The reserved memory** (00014₁₆ ~ 0007F₁₆) are used for storage of the pointers to user-defined interrupts.
- The 128-byte dedicated and reserved memory can contain 32 interrupt pointers.
- **The general-use memory** (00080₁₆ ~ FFFEF₁₆) stores data or instructions of the program.
- The dedicated memory (FFE0₁₆ ~ FFFEB₁₆) are used for hardware reset jump instruction.
2.6 Dedicated, Reserved, and General-Used Memory

<table>
<thead>
<tr>
<th>MEMORY</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESERVED</td>
<td>FFFFH</td>
</tr>
<tr>
<td>DEDICATED</td>
<td>FFFCH, FFFBH, FFF0H</td>
</tr>
<tr>
<td>OPEN</td>
<td>80H, 7FH</td>
</tr>
<tr>
<td>RESERVED</td>
<td>14H, 13H</td>
</tr>
<tr>
<td>DEDICATED</td>
<td>0H</td>
</tr>
</tbody>
</table>

2.7 Instruction Pointer

- The instruction pointer (IP) identifies the location of the next word of instruction code to be fetched from the current code segment of memory.
- The offset in IP is combined with the current value in CS to generate the address of the instruction code.
- During normal operation, the 8088 fetches instructions from the code segment of memory, stores them in its instruction queue, and executes them one after the other.
2.8 Data Registers

- Data registers are used for temporary storage of frequently used intermediate results.
- The contents of the data registers can be read, loaded, or modified through software.
- The four data registers are:
  - Accumulator register, A
  - Base register, B
  - Counter register, C
  - Data register, D
- Each register can be accessed either as a whole (16 bits) for word data or as 8-bit data for byte-wide operation.

![Diagram of Data Registers](image-url)
### 2.8 Data Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX</td>
<td>Word multiply, word divide, word I/O</td>
</tr>
<tr>
<td>AL</td>
<td>Byte multiply, byte divide, byte I/O, translate, decimal arithmetic</td>
</tr>
<tr>
<td>AH</td>
<td>Byte multiply, byte divide</td>
</tr>
<tr>
<td>BX</td>
<td>Translate</td>
</tr>
<tr>
<td>CX</td>
<td>String operations, loops</td>
</tr>
<tr>
<td>CL</td>
<td>Variable shift and rotate</td>
</tr>
<tr>
<td>DX</td>
<td>Word multiply, word divide, indirect I/O</td>
</tr>
</tbody>
</table>

**Dedicated register functions**

---

### 2.9 Pointer and Index Register

- The pointer registers and index registers are used to store offset addresses.
- Values held in the index registers are used to reference data relative to the data segment or extra segment.
- The pointer registers are used to store offset addresses of memory location relative to the stack segment register.
- Combining SP with the value in SS (SS:SP) results in a 20-bit address that points to the top of the stack (TOS).
- BP is used to access data within the stack segment of memory. It is commonly used to reference subroutine parameters.
2.9 Pointer and Index Register

- The index register are used to hold offset addresses for instructions that access data in the data segment.
- The source index register (SI) is used for a source operand, and the destination index (DI) is used for a destination operand.
- The four registers must always be used for 16-bit operations.

```
  16 0
   SP  BP  SI  DI
Stack pointer  Base pointer  Source index  Destination index
```

2.10 Status Register

- The status register, also called the flags register, indicate conditions that are produced as the result of executing an instruction.
- Only nine bits of the register are implemented. Six of these bits represent status flags and the other three bits represent control flags.
- The 8088 provides instructions within its instruction set that are able to use these flags to alter the sequence in which the program is executed.
### 2.10 Status Register

Status and control bits maintained in the flags register

<table>
<thead>
<tr>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>OF</td>
<td>DF</td>
<td>IF</td>
<td>TF</td>
<td>SF</td>
<td>ZF</td>
<td>x</td>
<td>AF</td>
<td>x</td>
<td>PF</td>
<td>x</td>
<td>CF</td>
</tr>
</tbody>
</table>

- Generally Set and Tested Individually
- 9 1-bit flags in 8086; 7 are unused

### Status flags indicate current processor status.

- **CF**: Carry Flag | Arithmetic Carry/Borrow
- **OF**: Overflow Flag | Arithmetic Overflow
- **ZF**: Zero Flag | Zero Result; Equal Compare
- **SF**: Sign Flag | Negative Result; Non-Equal Compare
- **PF**: Parity Flag | Even Number of “1” bits
- **AF**: Auxiliary Carry | Used with BCD Arithmetic
2.10 Status Register

- Control flags influence the 8086 during execution phase
  
  **DF** Direction Flag          Auto-Increment/Decrement  
  used for "string operations"

  **IF** Interrupt Flag         Enables Interrupts  
  allows "fetch-execute" to be interrupted

  **TF** Trap Flag             Allows Single-Step  
  for debugging; causes interrupt after each op

2.11 Generating a Memory Address

- A logical address in the 8088 microcomputer system is described by a segment base and an offset.
- The physical addresses that are used to access memory are 20 bits in length.
- The generation of the physical address involves combining a 16-bit offset value that is located in the instruction pointer, a base pointer, an index register, or a pointer register and a 16-bit segment base value that is located in one of the segment register.
2.11 Generating a Memory Address

Generating a physical address

Boundary of a segment

Highest addressed byte

DS:FFFFH

DS:BX

Data segment

DS:0000H

Lowest addressed byte

Memory

8088/8086

BX

DS

Boundary of a segment
2.11 Generating a Memory Address

EXAMPLE

What would be the offset required to map to physical address location 002C316 if the contents of the corresponding segment register are 002A16?

Solution:

The offset value can be obtained by shifting the contents of the segment of the segment register left by four bit positions and then subtracting from the physical address. Shifting left gives 002A016.

Now subtracting, we get the value of the offset:

\[ 002C3_{16} - 002A0_{16} = 0023_{16} \]
2.11 Generating a Memory Address

- Different logical addresses can be mapped to the same physical address location in memory.

2.12 The Stack

- The stack is implemented for temporary storage of information such as data or addresses.
- The stack is 64KBytes long and is organized from a software point of view as 32K words.
- The contents of the SP and BP registers are used as offsets into the stack segment memory while the segment base value is in the SS register.
- Push instructions (PUSH) and pop instructions (POP)
- Top of the stack (TOS) and bottom of the stack (BOS)
- The 8088 can push word-wide data and address information onto the stack from registers or memory.
- Many stacks can exist but only one is active at a time.
2.12 The Stack

EXAMPLE

Push operation

\[ \text{A}_{\text{BOS}} = 01050_{16} + \text{FFFE}_{16} = 01104_{16} \]

\[ \text{A}_{\text{TOS}} = 01050_{16} + 0008_{16} = 01058_{16} \]
### 2.12 The Stack

#### EXAMPLE
- Pop operation

![Stack Diagram](image)

- The Stack is a data structure used to store and retrieve data in a last-in, first-out (LIFO) order.
- It is commonly used in programming and computer science.
- The Stack can be implemented using a Last In, First Out (LIFO) list.

### 2.13 Input/Output Address Space

- The 8088 has separate memory and input/output (I/O) address space.
- The I/O address space is the place where I/O interfaces, such as printer and terminal ports, are implemented.
- The I/O address range is from $0000_{16}$ to $FFFF_{16}$. This represents 64KByte addresses.
- The I/O addresses are 16 bits long. Each of these addresses corresponds to one byte-wide I/O port.
- Certain I/O instructions can only perform operations to addresses $0000_{16}$ thru $00FF_{16}$ (page 0).
2.13 Input/Output Address Space

I/O address space

- OPEN
- RESERVED
- OPEN

PAGE 0

0H - FFH

F8H - F7H

0H - FFFFH