Lecture 1

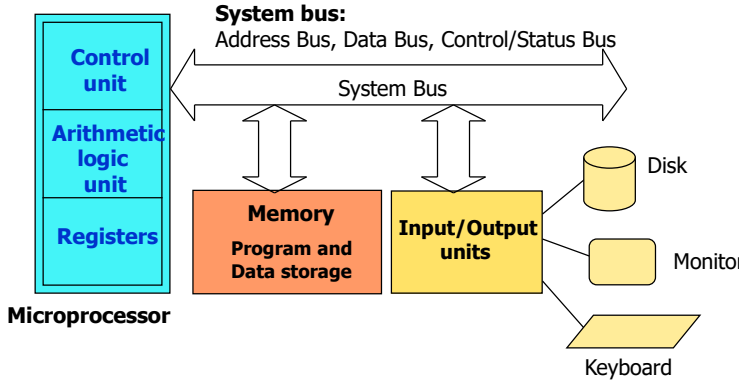
Introduction

Topics

- Microprocessor based Systems
- Number Systems

Microprocessor based Systems

- **Microprocessor** is a programmable device (chip: integrated circuit) that takes in numbers and performs on them arithmetic / logical operations according to a program stored in memory.



Microprocessor Languages

- **Machine language:** Low level instructions in binary form.
  - Every instruction is described by binary digits (bits).
  - Example machine code: 11001101 may mean the "Load" instruction.
- **Assembly Language:** Symbolic language to represent low level machine code.
  - Instructions are letters (mnemonics) that represent the operation performed.
  - Example command : LDA A, 5 (Load the accumulator A with value 5).
- Machine language and its associated Assembly language are completely machine dependent.
  - They are not transferable from one microprocessor to a different one.

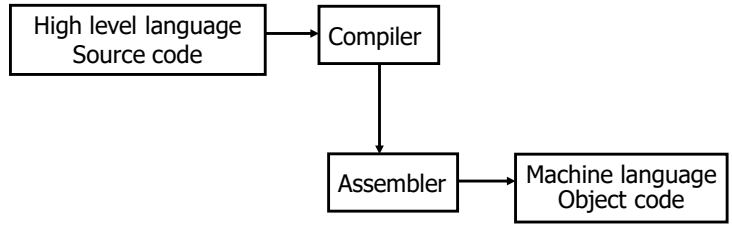
Assembly Language

- Assembly language instructions give direct control over the hardware components such as CPU, memory, Input/Output device, disk, etc.
- Comparing to high-level languages (C, C++, etc.), programs written in Assembly language require less memory, and run faster.
- An Assembler program (assembly compiler) should be used to compile an Assembly program code and generate the corresponding Machine Language code (binary).
- Usage examples for Assembly language:
  - Device drivers
  - Embedded systems
  - Real-time systems
  - BIOS booting
  - Compilers

5

High Level Languages

- To allow programs to be developed for multiple machines, high level languages were developed.
  - Programs are translated into microprocessor-specific assembly language, using a compiler or interpreter program.
  - High level statements are translated to machine language, compatible with the microprocessor being used.



6

Comparison of Microprocessor and Microcontroller

- **Microprocessor**
  - The processor is implemented on a Very Large Scale Integration (VLSI) chip.
  - Memory and Peripheral (Input/Output) chips are needed separately to construct a product.
  - They are mostly used in **computers**.
  - Examples: Intel X86, Motorola 68000, ARM
- **Microcontroller**
  - The processor,Memory, and Peripheral functions are implemented on the same VLSI chip.
  - They are mostly used in **embedded systems**.
  - Examples: Intel 8051, Microchip PIC, Atmel AVR

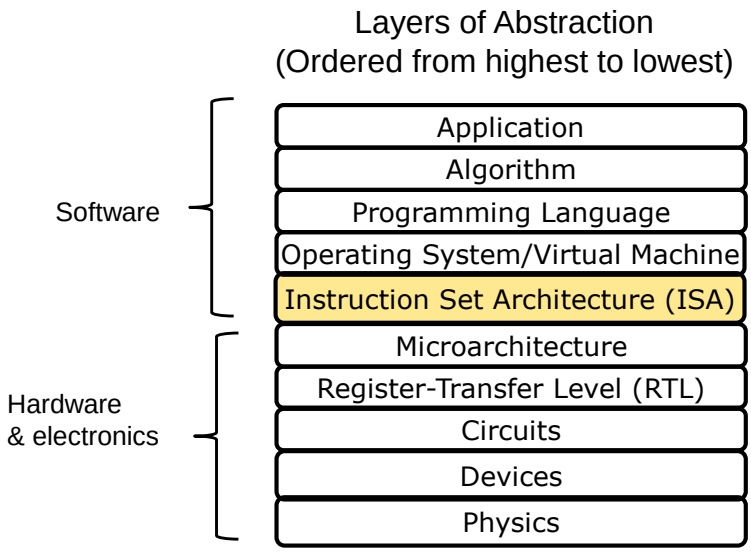
7

Embedded System

- **Embedded System** is a special-purpose system designed to perform only specific functions.
- Usually **microcontrollers** are used in embedded systems.
- Assembly language or a high level language such as C can be used.
- Some Application Areas:
  - **Mobile :** Cellular phone, Tablet PC, Scientific calculator
  - **Automotive :** Engine and Fuel control system, Navigation, Parking assistance, Cruise control, etc.
  - **Consumer Electronics :** Camera, Music/DVD player, Remote control, Washing machine, Refrigirator, Air conditioning system, etc.
  - **Banking Systems:** ATM, etc.
  - **Other :** Medical, Military, Robotics, Aviation , etc.

8

Operation of a microcomputer



9

Topics

- Microprocessor based Systems
- Number Systems

10

Number Systems

System	Base	Digits used
Binary	2	0 1
Octal	8	0 1 2 3 4 5 6 7
Decimal	10	0 1 2 3 4 5 6 7 8 9
Hexadecimal	16	0 1 2 3 4 5 6 7 8 9 A B C D E F

11

Binary Number System

- Digital systems have only two states : 0 and 1
  (OFF : 0 volts , ON : 5 volts)
- Computers use binary number system.
- 1 Byte is a unit containing 8 bits (Binary Digit).
  Example : 10101001

Binary to Hexadecimal conversion :

Method : From right-to-left , group the binary digits into four bits, then write the hexadecimal equivalents for each group.

$10101001 = (1010\ 1001)_2 = (A9)_{16}$

12

Example: Number system conversion table

- For decimal numbers between 0 - 15, binary representation can be written with 4 bits.

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

- For decimal numbers between 16 - 255, at least 8 bits should be used.

Decimal	Binary	Hexadecimal
16	0001 0000	10
17	0001 0001	11
18	0001 0010	12
19	0001 0011	13
20	0001 0100	14
...	...	...
127	0111 1111	7F
128	1000 0000	80
...	...	...
254	1111 1110	FE
255	1111 1111	FF

13

Binary to Decimal Conversion

- **Example:** Convert the Binary number 101010 to Decimal number.
- The digits of a binary number are used as coefficients.
- The exponents of base 2 number are increasing from right-to-left.

Exponents	2^5	2^4	2^3	2^2	2^1	2^0
Digits	1	0	1	0	1	0

- The sum of the following formula gives the decimal result.

$$101010 = 1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 1x2^1 + 0x2^0$$
$$= 32 + 8 + 2$$
$$= (42)_{10}$$

14

Conversion Examples

Hexadecimal to Decimal Conversion :

A2F = $10*16^2 + 2*16^1 + 15*16^0$

= 2560 + 32 + 15

= 2607

Decimal value of A is 10.
Decimal value of F is 15.

Decimal to BCD (Binary Coded Decimal) Conversion :

25 = 0010 0101

2 5

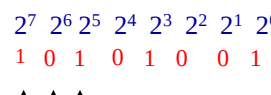
Each decimal digit is converted separately to binary (4 bits).

15

Decimal to Binary Conversion

Method 1: Divide decimal number into 2's repeatedly, until reaching 1 as the division result.

DECIMAL		Remainder
169	169 / 2 = 84	1
84	84 / 2 = 42	0
42	42 / 2 = 21	0
21	21 / 2 = 10	1
10	10 / 2 = 5	0
5	5 / 2 = 2	1
2	2 / 2 = 1	0
1	1 / 2 = 0	1



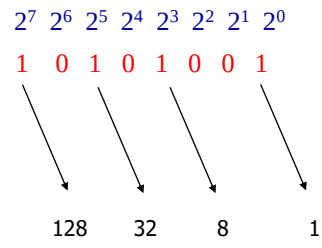
The remainders are written in reverse order, to obtain the Binary number.
169 = 10101001

16

Decimal to Binary Conversion

Method 2: Look for the highest 2^n , and subtract from decimal number repeatedly.

DECIMAL	Largest 2^n
169	169 - 128 = 41
41	41 - 32 = 9
9	9 - 8 = 1
1	1 - 1 = 0



The coefficients are written from left to right, to obtain the Binary number.
169 = 10101001

17

Representing Numbers in Computers

Unsigned numbers (1 Byte = 8 bits)

Decimal	Binary	Hexadecimal
0	0000 0000	00
1	0000 0001	01
2	0000 0010	02
...	...	...
126	0111 1110	7E
127	0111 1111	7F
128	1000 0000	80
129	1000 0001	81
...	...	...
253	1111 1101	FD
254	1111 1110	FE
255	1111 1111	FF

18

Signed Numbers : Signed Magnitude Method

- The Most Significant Bit (MSB) is the sign bit (leftmost).
  - 0 means positive, 1 means negative.
- The remaining bits in the number indicate the magnitude (or absolute value).
- In this representation there are two ways to represent zero:
  00000000 (0) and
  10000000 (-0).

8 bit signed magnitude:

Binary value	Signed magnitude interpretation	Unsigned interpretation
00000000	0	0
00000001	1	1
...	...	...
01111111	127	127
10000000	-0	128
...	...	...
11111111	-127	255

19

Signed Numbers : Two's Complement Method

- 2's complement is an efficient method to represent signed numbers, which is used in **computers**.
- **Method:** To obtain a negative number:
  1) Firstly complement the number (replace 0s with 1s, 1s with 0s)
  2) Then add 1.
- The Most Significant Bit (MSB) is the sign bit (leftmost).
  0 is positive, 1 is negative.

8 bit two's complement:

Binary value	Two's complement interpretation	Unsigned interpretation
00000000	0	0
00000001	1	1
...	...	...
01111110	126	126
01111111	127	127
10000000	-128	128
10000001	-127	129
10000010	-126	130
...	...	...
11111110	-2	254
11111111	-1	255

20

Example: Signed Number

Find the binary representation of -7.

0000 0111 ⟶ +7

Signed Magnitude Method :

Just change the MSB bit from 0 to 1, to make it negative.

1000 0111 ⟶ -7

Two's Complement Method :

Firstly write the 1's complement.
Then add 1.

1's complement : 1111 1000
2's complement : 1111 1001 ⟶ -7

21

Carry and Borrow conditions

- **Carry**: When adding two unsigned numbers, if the result is too big, then the carry flag bit is set to 1.
- When numbers are added and subtracted, the **carry flag** (CF) represents
  - **9th bit**, if 8-bit numbers are added
  - **17th bit**, if 16-bit numbers are added
- **Borrow**: When subtracting two unsigned numbers, if the second number is bigger than the first number, then borrow flag bit is set to 1.

22

Carry Example

- **Carry bit** : The 9th bit is the carry bit, when adding two 8-bit unsigned numbers.

```
   1010 0001
+  1000 1111
-------------

 1 0011 0000
 ↑
```

Carry bit
(9th bit)

Addition

- Add two **signed** numbers including their sign bits, then discard the carry bit.
- Negative numbers are represented in 2's complement.

```
+6 :    0000 0110
+13:  + 0000 1101
+19:    0001 0011
```

```
-6 :    1111 1010
+13:  + 0000 1101
+7:   1 0000 0111
```

End carry: discarded
Result is positive

```
+6 :    0000 0110
-13:  + 1111 0011
-7 :    1111 1001
```

```
-6 :    1111 1010
-13:  + 1111 0011
-19:  1 1110 1101
```

End carry: discarded
Result is negative

Subtraction

- Subtraction is done by using the addition operation.
- The negative number should be converted to 2's complement number.

5-2=3

```
5:   0000 0101                    0000 0101
2:   0000 0010   2's complement  + 1111 1110
3:                                1 0000 0011
```

End carry: discarded

Result is positive

2-5=-3

```
2:   0000 0010                    0000 0010
5:   0000 0101   2's complement  + 1111 1011
-3:                               1111 1101
```

No end carry

Result is negative

25

Overflow conditions

- **Overflow**: When adding two **signed** numbers, if the result is bigger than the maximum number, then overflow occurs.
  - Or if the result is smaller than the minimum number.
  - For example, when the sum of two positive 8-bit numbers exceeds 127, there is an overflow.
  - Similarly, if sum of two negative 8-bit numbers is less than or equal to -128, there is an overflow.
- When two **unsigned** numbers are added, the **Carry** bit also indicates an overflow.
  There is a carry out of the leftmost (most significant) bit.

26

Overflow examples

- Overflow occurs if the **sign bit of result** changes for the **addition** of two positive numbers, or two negative numbers.
- Overflow can NOT occur if one number is positive and the other is negative.

ADDING TWO POSITIVE NUMBERS

```
 70:   0100 0110
 80: + 0101 0000
150:   1001 0110   (Overflow: 150>127)
```

Both operands are positive.
The result sign (8th bit) is negative.

ADDING TWO NEGATIVE NUMBERS

```
-70:    1011 1010
-80: +  1011 0000
-150: 1 0110 1010   (Overflow: -150<-128)
      Carry
```

Both operands are negative.
The result sign (8th bit) is positive.

27

Overflow Detection

- Distinguishing between signed and unsigned is based on programmer's interpretation.

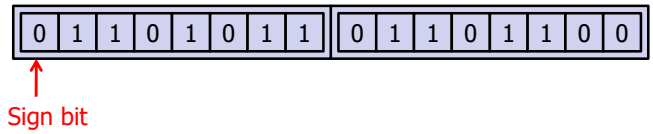
  For example: 11111011 can be interpreted as an unsigned number (251).
  Or, it can be also interpreted as a two's complement signed number (-5).

- The distinction between signed and unsigned data types is very important when **detecting an overflow** after addition or subtraction.
- Overflow is detected by considering two separate cases:
  - Overflow when adding two **signed** numbers is indicated by the **Overflow flag**.
  - Overflow when adding two **unsigned** numbers is indicated by the **Carry flag**.

28

Representation of Integer Numbers

- 16-bit integer numbers are stored in two memory locations (each is 1 byte).
- Example: The integer 27500 can be represented as follows.



Range of 16-bit integers :

Signed range : **-32768** to **+32767**
Unsigned range : **0** to **65535**

Representation of Floating Point Numbers

- 3 bytes (24 bits) can be used for floating point numbers.

Example number = 207.4
Mantissa = 2074
Exponent = 3

$207.4 = 0.2074 \times 10^3$