Computer Organization and Assembly Language

Lecture 1 – Basic Concepts

Virtual Machine

High-level language	Level 5
Assembly language	Level 4
Operating System	Level 3
Instruction Set Arch.	Level 2
Microarchitecture	Level 1
Digital Logic	Level 0

The Intel Microprocessor Family

- The Intel family owes its origins to the ***8080***, an 8-bit processor which could only access 64 kilobytes of memory.
- The ***8086*** (1978) had 16-bit registers, a 16-bit data bus, 20-bit memory using segmented memory. The IBM PC used the ***8088***, which was identical except it used an 8-bit data bus.
- ***8087*** - a math co-processor that worked together with the 8086/8088. Without it, floating point arithmetic require complex software routines.
- ***80286*** - ran in real mode (like the 8086/8088) or in protected mode could access up tp 16MB using 24-bit addressing with a clock spped between 12 and 25 MHz. Its math co-processor was the 80287.

The Intel Microprocessor Family (continued)

- 80386 or ***i386*** (1985) - used 32-bit registers and a 32-bit data bus. It could operate in real, protected or virtual mode. In virtual mode, multiple real-mode programs could be run.
- ***i486*** - The instruction set was implemented with up to 5 instructions fetched and decoded at once. SX version had its FPU disabled.
- The Pentium processor had an original clock speed of 90 MHz and cold decode and executed two instructions at the same time, using ***dual pipelining***.

Number Systems - Base 10

The number system that we use is base 10:

$1734 = 1000 + 700 + 30 + 4$

$= 1x1000 + 7x100 + 3x10 + 4x1$

$= 1x10^3 + 7x10^2 + 3x10^1 + 4x10^0$

$724.5 = 7x100 + 2x10 + 4x1 + 5x0.1$

$= 7x10^2 + 2x10^1 + 4x10^0 + 5x10^{-1}$

Why use base 10?

Number Systems - Base 2

For computers, base 2 is more convenient (why?)

$10011_2 = 1x16 + 0x8 + 0x4 + 1x2 + 1x1 = 19_{10}$

$100010_2 = 1x32 + 0x16 + 0x8 + 0x4 + 1x2 + 0x1 = 34_{10}$

$101.001_2 = 1x4 + 0x2 + 1x1 + 0x0.5 + 0x0.25 + 1x0.125$

$= 5.125_{10}$

<u>Example</u> - $1101011_2 = ?$

$10110111_2 = ?$

$10100.1101_2 = ?$

Number Systems - Base 16

Hexadecimal (base 16) numbers are commonly used because it is convert them into binary (base 2) and vice versa.

$8CE_{16}$ = 8x256 + 12x16 + 14x1

= 2048 + 192 + 14

= 2254

3F9 = 3x256 + 15x16 + 9x1

= 768 + 240 + 9 = 1017

Number Systems - Base 16 (continued)

Base 2 is easily converted into base 16:

100011001110_2 = 1000 1100 1110 = 8 C E $_{16}$

11101101110101001_2 = 1 1101 1011 1010 1001 = 1 D B A 9_{16}

10110001010000010111_2 = $?_{16}$

10110101001011011_2 = $?_{16}$

Number Systems - Base 16 (continued)

Converting base 16 into base 2 works the same way:

$F3A5_{16}$ = 1111 0011 1010 01012

$76EF_{16}$ = 0111 0110 1110 11112

$AB3D_{16} = ?_2$

$15C.38_{16} = ?_2$

Converting From Decimal to Binary

19
9 R 1
4 R 1
2 R 0
1 R 0
0 R 1

10011_2

Converting From Decimal to Binary

```
 34
17 R 0
 8 R 1
 4 R 0
 2 R 0
 1 R 0
 0 R 1
```

100010_2

Converting From Binary to Decimal

$1001010_2 = 1x64 + 0x32 + 0x16 + 1x8 + 0x4 + 1x2 + 0x1$

$= 64 + 8 + 2 = 74_{10}$

$1101101011_2 = 1x512 + 1x256 + 0x128 + 1x64 + 1x32$

$+ 0x16 + 8x1 + 0x4 + 1x2 + 1x1$

$= 512 + 256 + 64 + 32 + 8 + 2 + 1 = 875_{10}$

Signed numbers

0	1	0	0	1	0	1	1	$= 75_{10}$

sign bit

1	0	1	1	0	1	0	1	$= -75_{10}$

sign bit

```
   01001011
   10110101
1  00000000
```

overflow bit

Binary Bit Position Values

2^0	1	2^8	256
2^1	2	2^9	512
2^2	4	2^{10}	1024
2^3	8	2^{11}	2048
2^4	16	2^{12}	4096
2^5	32	2^{13}	8192
2^6	64	2^{14}	16384
2^7	128	2^{15}	32768

Binary, Decimal and Hexadecimal Equivalents

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

Types of Numbers

Storage Type	**Bits**	**Range (low-high)**
Signed byte	7	-128 to +127
Unsigned byte	8	0 to 255
Signed word	15	-32,768 to +32,767
Unsigned word	16	0 to 65,535
Signed doubleword	31	-2,147,483,648 to +2,147,483,648
Unsigned doubleword	32	0 to 4,294, 967,295
Signed quadword	63	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
Unsigned quadword	64	0 to 8,446,744,073,709,551,615

ASCII representation of characters

- ASCII (***A***merican ***S***tandard ***C***ode for ***I***nformation ***I***nterchange) is a numeric code used to represent characters.
- All characters are represented this way including:
  - words (character strings)
  - numbers
  - punctuation
  - control characters
- There are separate values for upper case and lower case characters:

A	65	z	122
B	66	*blank*	32
Z	90	$	52
a	97	0	48
b	98	9	57x

Boolean Values and Expressions

- A ***boolean*** value is either true or false
- Boolean expressions involve boolean values and boolean operators.
- There are three primary boolean operators about which we are interested:
  - NOT
  - AND
  - OR

The **NOT** Operator

x	**~x**
F	T
T	F

The **AND** Operator

X	**Y**	**X $\wedge$ Y**
F	F	F
F	T	F
T	F	F
T	T	T

The **OR** Operator

X	**Y**	**$X \vee Y$**
F	F	F
F	T	T
T	F	T
T	T	T

Operator Precedence

NOT
AND
OR

Higher precedence

Examples:

$\sim x \vee y$ NOT, then OR

$\sim(x \vee y)$ OR, then NOT

$x \vee (y \wedge z)$ AND, then OR

Boolean Functions – An Example

Boolean functions take boolean inputs and produce boolean outputs, e.g., $\sim x \vee y$

x	**~x**	**y**	**~x ∨ y**
F	T	F	T
F	T	T	T
T	F	F	F
T	F	T	T

Boolean Functions – Another Example

E. g., $x \wedge \sim y$

x	**y**	**~Y**	~x ∧ y
F	F	T	F
F	T	F	F
T	F	T	T
T	T	F	F

One Last Example - $(y \wedge s) \vee (x \wedge \sim s)$

x	y	s	$y \wedge s$	$\sim s$	$x \wedge \sim s$	$(y \wedge s) \vee (x \wedge \sim s)$
F	F	F	F	T	F	F
F	F	T	F	F	F	F
F	T	F	F	T	F	F
F	T	T	T	F	F	T
T	F	F	F	T	T	T
T	F	T	F	F	T	T
T	T	F	F	T	F	F
T	T	T	T	F	F	T