Duration: 13 Sessions | Undergraduate Program | Winter 2023 | Department: Computer Science

Intro Assembly Lang & Op Sys

Instructor: Dr. Amin Safaei

Winter 2023

Lakehead UNIVERSITY

Lesson overview

Assembly-level machine organization.
Memory system organization and architecture.
Writing simple I/O routines and interrupt handlers.
Introduction to initialization and process management in a Unix or Unix-like operating system.

Pre-requisite

- **A knowledge of C and C++ will be useful**

Course Resources

- **Primary text**
  - Slides and instructor study materials.
  - There is no required textbook for this course.
- **Additional text**
  - Barry B. Brey, The Intel Microprocessor: Architecture, Programming, and Interfacing, eight edition, Prentice Hall India, 2008
  - M. A. Mazidi, R. D. McKinlay, J. G. Mazidi, 8051 Microcontroller, The: A Systems Approach

Office Hours:

- **Instructor:** Monday and Wednesday at 12:00 pm to 1:00 pm, **asafaei@lakeheadu.ca**
- **TA:**

Take Home Assignments

- There will be two assignments this term that must submit through d2l (Courselink)
- Sending assignment to instructor email or TAs will not evaluate and receive 0.
- **Late submission is not accept**

Software Tools

- Visual Studio 2019/22
- MASM (Microsoft Assembler)
- NASM (Netwide Assembler)

Lakehead UNIVERSITY

Evaluations

- Assignment: (10+10) 20%
- Lab (Report) 10%
- Mid-term exam (Closed book) 20%
- POP Quizzes 10%
- Final exam (Closed book) 40%

Lakehead UNIVERSITY

What’s in it for you ?

1. Basic Concepts
2. Processor Architecture
3. Assemble Language Fundamentals
4. Data Transfer Addressing, and Arithmetic
5. Procedures
6. Conditional Processing
7. Integer Arithmetic
8. Advanced Procedures

Lakehead UNIVERSITY

What’s in it for you ?

9. Process Management

10. Processes and threads

Lakehead UNIVERSITY

Duration: 90 min | Undergraduate Program | Fall 2023 | Department: Computer Science

Basic Concepts

Instructor: Dr. Amin Safaei

Winter 2023

Lakehead UNIVERSITY

- **This set of lecture slides is made from the following textbooks:**
  - Barry B. Brey, The Intel Microprocessor: Architecture, Programming, and Interfacing, eight edition, Prentice Hall India, 2008.
  - M. A. Mazidi, R. D. McKinlay, J. G. Mazidi, 8051 Microcontroller, The: A Systems Approach
  - S. P. Dandamudi, Introduction to Assembly Language Programming For Pentium and RISC Processors
  - Kip R. Irvine, Assembly Language for x86 Processors (8th Edition)

- **The slides are picked or adapted from the set of slides provided by the following textbooks:**
  - Barry B. Brey, The Intel Microprocessor: Architecture, Programming, and Interfacing, eight edition, Prentice Hall India, 2008.
  - M. A. Mazidi, R. D. McKinlay, J. G. Mazidi, 8051 Microcontroller, The: A Systems Approach
  - S. P. Dandamudi, Introduction to Assembly Language Programming For Pentium and RISC Processors
  - Kip R. Irvine, Assembly Language for x86 Processors (8th Edition)

1.1 Overview

- **Welcome to Assembly Language**
- Virtual Machine Concept
- Data Representation
- Boolean Operations

Lakehead UNIVERSITY

1.2 Welcome to Assembly Language

- Some Good Questions to Ask
- Assembly Language Applications
- **Questions to Ask**
  - Why am I learning Assembly Language?
  - What background should I have?
  - What is an assembler?
  - What hardware/software do I need?
  - What types of programs will I create?
  - How does assembly language (AL) relate to machine language?
  - How do C++ and Java relate to AL?
  - Is AL portable?
  - **Why learn AL?**

```
int   Y;
int   X = (Y + 4) * 3;

mov   eax,Y                    ; move Y to the EAX register
add   eax,4                    ; add 4 to the EAX register
mov   ebx,3                    ; move 3 to the EBX register
imul  ebx                      ; multiply EAX by EBX
mov   X,eax                    ; move EAX to X
```

Lakehead UNIVERSITY

1.3 Assembly Language Applications

- Some representative types of applications:
  - Business application for single platform
  - Hardware device driver
  - Business application for multiple platforms

Type of Application	High-Level Languages	Assembly Language
Business application software, written for single platform, medium to large size.	Formal structures make it easy to organize and maintain large sections of code.	Minimal formal structure, so one must be imposed by programmers who have varying levels of experience. This leads to difficulties maintaining existing code.
Hardware device driver.	Language may not provide for direct hardware access. Even if it does, awkward coding techniques must often be used, resulting in maintenance difficulties.	Hardware access is straightforward and simple. Easy to maintain when programs are short and well documented.
Business application written for multiple platforms (different operating systems).	Usually very portable. The source code can be recompiled on each target operating system with minimal changes.	Must be recoded separately for each platform, often using an assembler with a different syntax. Difficult to maintain.
Embedded systems and computer games requiring direct hardware access.	Produces too much executable code, and may not run efficiently.	Ideal, because the executable code is small and runs quickly.

1.3 What's Next (1 of 3)

- Welcome to Assembly Language
- **Virtual Machine Concept**
  - **Virtual Machines**
  - **Specific Machine Levels**
- Data Representation
- Boolean Operations

1.4 Virtual Machines

- Virtual machine concept
- Programming Language analogy:
  - Each computer has a native machine language (language L0) that runs directly on its hardware
  - A more human-friendly language is usually constructed above machine language, called Language L1
- Programs written in L1 can run two different ways:
  - Interpretation – L0 program interprets and executes L1 instructions one by one
  - Translation – L1 program is completely translated into an L0 program, which then runs on the computer hardware

English: Display the sum of A times B plus C.

↓

C++: cout << (A * B + C);

↓

Assembly Language:

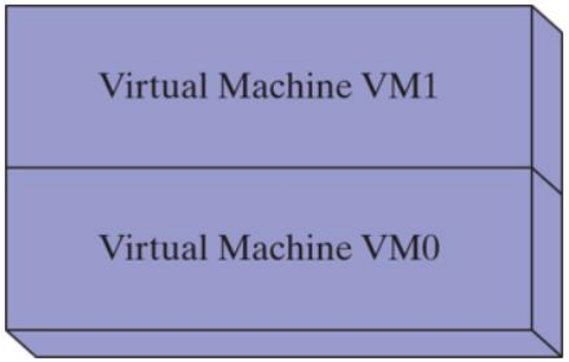
```
mov eax,A
mul B
add eax,C
call WriteInt
```

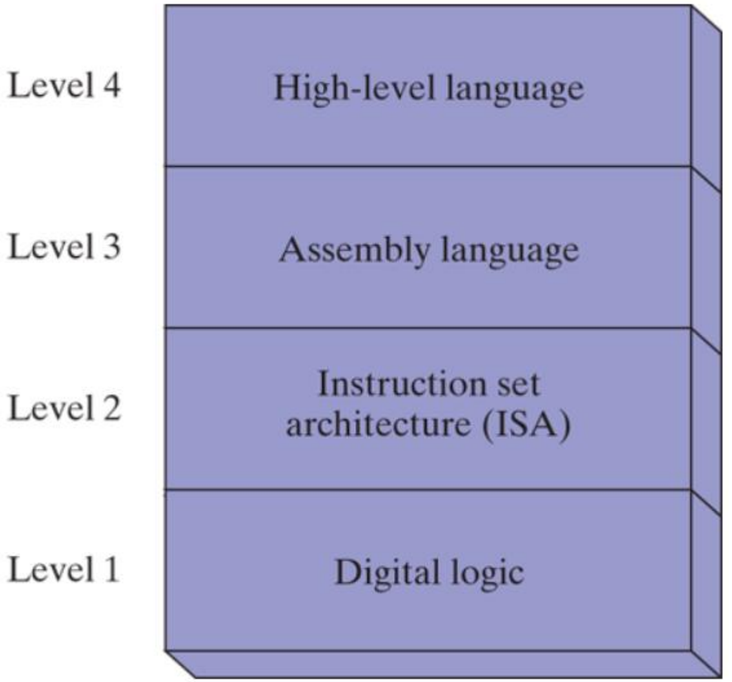
→

Intel Machine Language:

A1 00000000
F7 25 00000004
03 05 00000008
E8 00500000

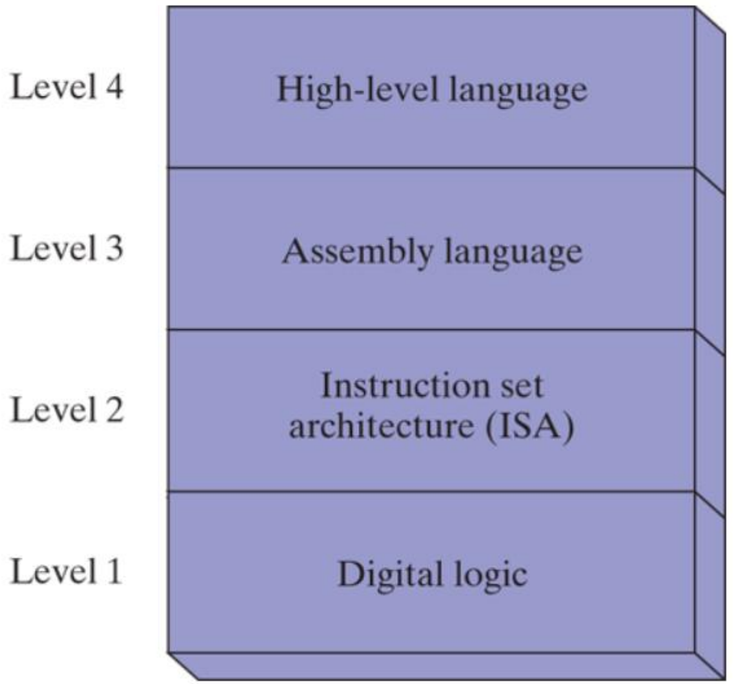
1.6 Specific Machine Levels





1.6 Specific Machine Levels

- **Level 4 - High-Level Language**
  - Application-oriented languages
    - C++, Java, Pascal, Visual Basic . . .
  - Programs compile into assembly language (Level 4)
- **Level 3 - Assembly Language**
  - Instruction mnemonics that have a one-to-one correspondence to machine language
  - Programs are translated into Instruction Set Architecture Level - machine language (Level 2)
- **Level 2 - Instruction Set Architecture (ISA)**
  - Also known as conventional machine language
  - Executed by Level 1 (Digital Logic)
- **Level 1 - Digital Logic**
  - CPU, constructed from digital logic gates
  - System bus
  - Memory
  - Implemented using bipolar transistors



Lakehead UNIVERSITY

1.7 What's Next (2 of 3)

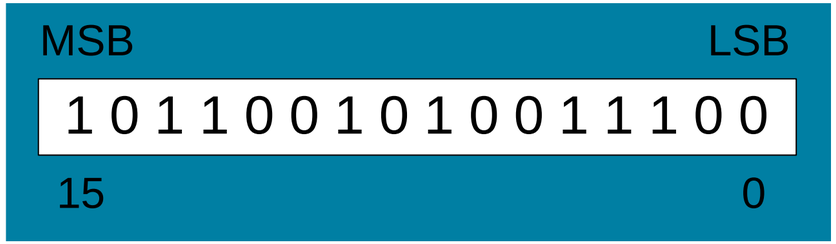
- Welcome to Assembly Language
- Virtual Machine Concept
- **Data Representation**
- Boolean Operations

1.8 Data Representation

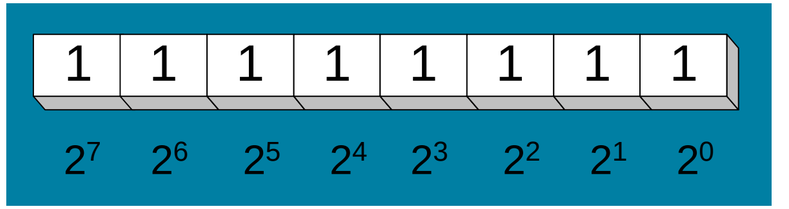
- Binary Numbers
  - Translating between binary and decimal
- Binary Addition
- Integer Storage Sizes
- Hexadecimal Integers
  - Translating between decimal and hexadecimal
  - Hexadecimal subtraction
- Signed Integers
  - Binary subtraction
- Character Storage

1.9 Binary Numbers

- Digits are 1 and 0
- 1 = true
- 0 = false
- MSB – most significant bit
- LSB – least significant bit
- Bit numbering:



Each digit (bit) is either 1 or 0
Each bit represents a power of 2:



Every binary number is a sum of powers of 2

2^n	Decimal Value	2^n	Decimal Value
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

Lakehead UNIVERSITY

1.10 Translating Binary to Decimal

- Weighted positional notation shows how to calculate the decimal value of each binary bit:

$$dec = (D_{n-1} \times 2^{n-1}) + (D_{n-2} \times 2^{n-2}) + \ldots + (D_1 \times 2^1) + (D_0 \times 2^0)$$

D = binary digit → binary 00001001 =

decimal 9: $(1 \times 2^3) + (1 \times 2^0) = 9$

Lakehead UNIVERSITY

1.11 Translating Unsigned Decimal to Binary

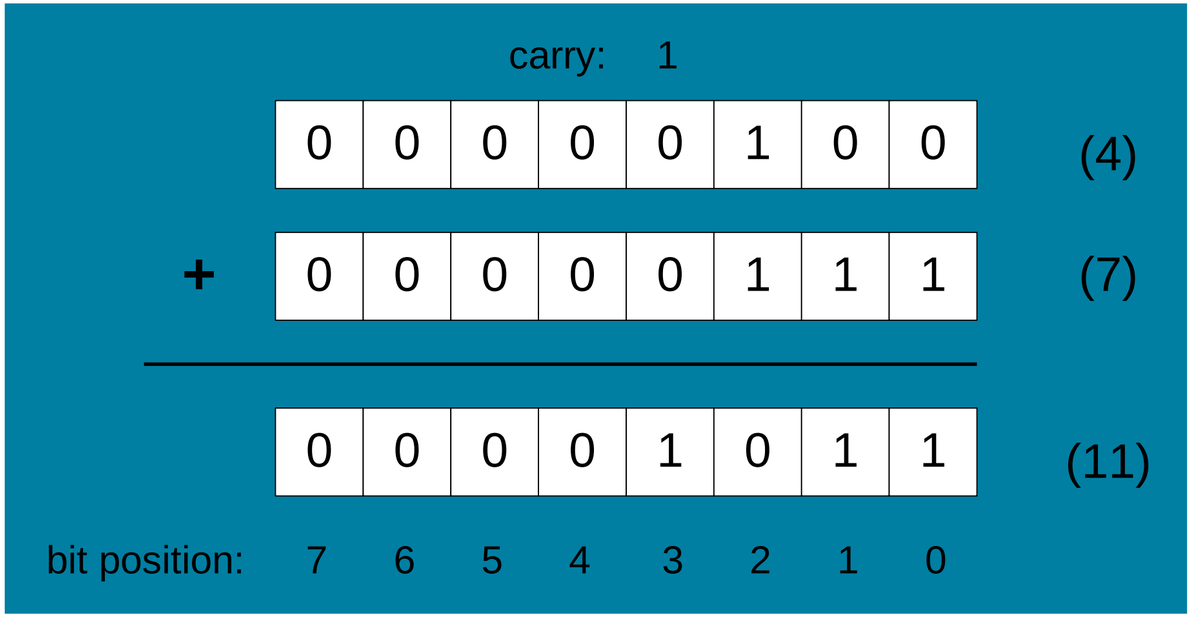
- Repeatedly divide the decimal integer by 2. Each remainder is a binary digit in the translated value:

Division	Quotient	Remainder
37 / 2	18	1
18 / 2	9	0
9 / 2	4	1
4 / 2	2	0
2 / 2	1	0
1 / 2	0	1

37 = 100101

1.12 Binary Addition

- Starting with the LSB, add each pair of digits, include the carry if present.



1.13 Integer Storage Sizes

Standard sizes:

Storage Type	Range (low–high)	Powers of 2
Unsigned byte	0 to 255	0 to $(2^8 - 1)$
Unsigned word	0 to 65,535	0 to $(2^{16} - 1)$
Unsigned doubleword	0 to 4,294,967,295	0 to $(2^{32} - 1)$
Unsigned quadword	0 to 18,446,744,073,709,551,615	0 to $(2^{64} - 1)$

One ***kilobyte*** is equal to 2^{10}, or 1024 bytes.
One ***megabyte*** (1 MByte) is equal to 2^{20}, or 1,048,576 bytes.
One ***gigabyte*** (1 GByte) is equal to 2^{30}, or 1024^3, or 1,073,741,824 bytes.
One ***terabyte*** (1 TByte) is equal to 2^{40}, or 1024^4, or 1,099,511,627,776 bytes.
One ***petabyte*** is equal to 2^{50}, or 1,125,899,906,842,624 bytes.
One ***exabyte*** is equal to 2^{60}, or 1,152,921,504,606,846,976 bytes
One ***zettabyte*** is equal to 2^{70} bytes.
One ***yottabyte*** is equal to 2^{80} bytes.

Lakehead UNIVERSITY

1.14 Hexadecimal Integers

- Binary values are represented in hexadecimal.

Binary	Decimal	Hexadecimal	Binary	Decimal	Hexadecimal
0000	0	0	1000	8	8
0001	1	1	1001	9	9
0010	2	2	1010	10	A
0011	3	3	1011	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

1.15 Translating Binary to Hexadecimal

- Each hexadecimal digit corresponds to 4 binary bits.
- Example: Translate the binary integer 000101101010011110010100 to hexadecimal:

1	6	A	7	9	4
0001	0110	1010	0111	1001	0100

1.16 Converting Hexadecimal to Decimal

- Multiply each digit by its corresponding power of 16:
- $dec = (D_3 \times 16^3) + (D_2 \times 16^2) + (D_1 \times 16^1) + (D_0 \times 16^0)$
- Hex 1234 equals

  $(1 \times 16^3) + (2 \times 16^2) + (3 \times 16^1) + (4 \times 16^0)$, or decimal 4,660.
- Hex 3BA4 equals

  $(3 \times 16^3) + (11 \times 16^2) + (10 \times 16^1) + (4 \times 16^0)$, or decimal 15,268.

16^n	Decimal Value	16^n	Decimal Value
16^0	1	16^4	65,536
16^1	16	16^5	1,048,576
16^2	256	16^6	16,777,216
16^3	4096	16^7	268,435,456

1.17 Converting Decimal to Hexadecimal

- Decimal 422 = 1A6 hexadecimal

Division	Quotient	Remainder
422 / 16	26	6
26 / 16	1	A
1 / 16	0	1

Lakehead UNIVERSITY

1.18 Hexadecimal Addition/Subtraction

- Divide the sum of two digits by the number base (16).
- The quotient becomes the carry value, and the remainder is the sum digit.
- **Important skill: Programmers frequently add and subtract the addresses of variables and instructions**.

		1	1
36	28	28	6A
42	45	58	4B
78	6D	80	B5

21 / 16 = 1, rem 5

- When a borrow is required from the digit to the left, add 16 (decimal) to the current digit's value:

16 + 5 = 21

	–1
C6	75
A2	47
24	2E

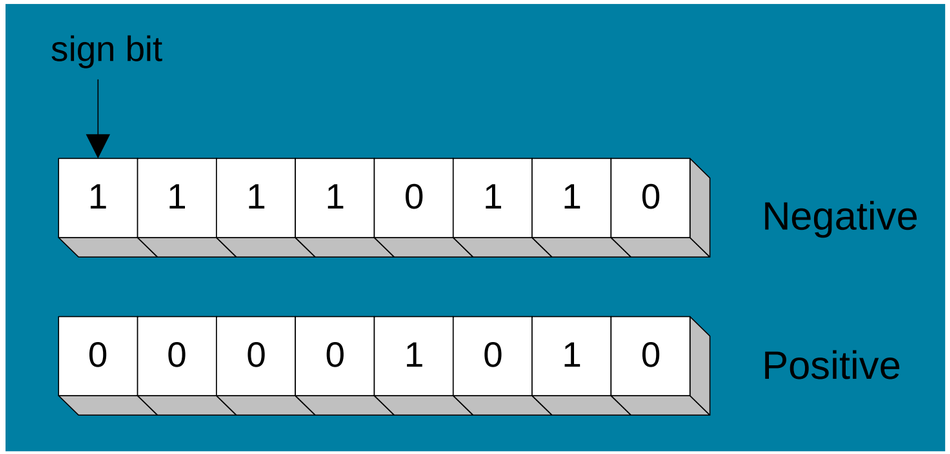
Lakehead UNIVERSITY

1.19 Signed Integers

- The highest bit indicates the sign.

  1 = negative,
  0 = positive

- If the highest digit of a hexadecimal integer is > 7, the value is negative. Examples: 8A, C5, A2, 9D



1.20 Forming the Two's Complement

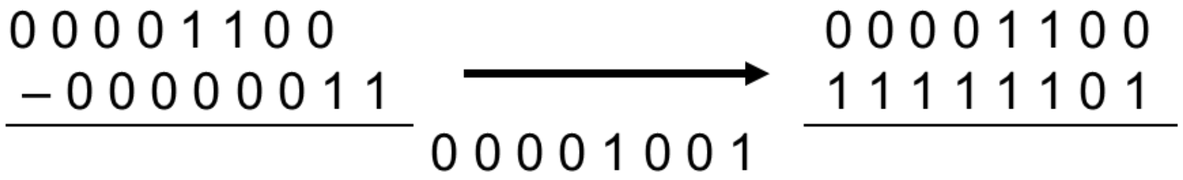
- Negative numbers are stored in two's complement notation
- Represents the additive Inverse

Starting value	00000001
Step 1: reverse the bits	11111110
Step 2: add 1 to the value from Step 1	11111110 +00000001
Sum: two's complement representation	11111111

Note that 00000001 + 11111111 = 00000000

1.21 Binary Subtraction

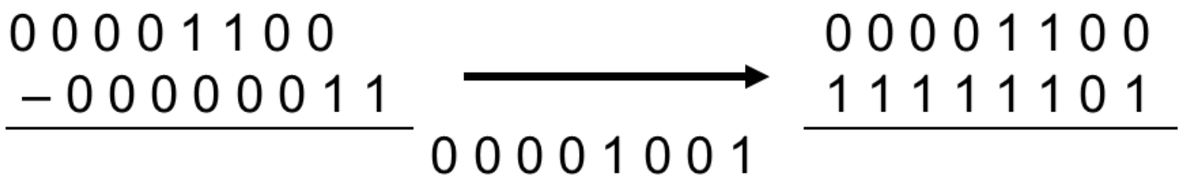
- When subtracting A – B, convert B to its two's complement
- Add A to (–B)



Lakehead UNIVERSITY

1.21 Binary Subtraction

- When subtracting A – B, convert B to its two's complement
- Add A to (–B)



- **Learn How To Do the Following:**
  - Form the two's complement of a hexadecimal integer
  - Convert signed binary to decimal
  - Convert signed decimal to binary
  - Convert signed decimal to hexadecimal
  - Convert signed hexadecimal to decimal

1.22 Ranges of Signed Integers

- The highest bit is reserved for the sign. This limits the range:

Storage Type	Range (low–high)	Powers of 2
Signed byte	−128 to +127	-2^7 to $(2^7 - 1)$
Signed word	−32,768 to +32,767	-2^{15} to $(2^{15} - 1)$
Signed doubleword	−2,147,483,648 to 2,147,483,647	-2^{31} to $(2^{31} - 1)$
Signed quadword	−9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	-2^{63} to $(2^{63} - 1)$

1.23 Character Storage

- Character sets
  - Standard ASCII (0 – 127)
  - Extended ASCII (0 – 255)
  - ANSI (0 – 255)
  - Unicode(0 – 65,535)
- Null-terminated String
  - Array of characters followed by a null byte
- Using the ASCII table

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	*[NULL]*	32	20	*[SPACE]*	64	40	@	96	60	`
1	1	*[START OF HEADING]*	33	21	!	65	41	A	97	61	a
2	2	*[START OF TEXT]*	34	22	"	66	42	B	98	62	b
3	3	*[END OF TEXT]*	35	23	#	67	43	C	99	63	c
4	4	*[END OF TRANSMISSION]*	36	24	$	68	44	D	100	64	d
5	5	*[ENQUIRY]*	37	25	%	69	45	E	101	65	e
6	6	*[ACKNOWLEDGE]*	38	26	&	70	46	F	102	66	f
7	7	*[BELL]*	39	27	'	71	47	G	103	67	g
8	8	*[BACKSPACE]*	40	28	(	72	48	H	104	68	h
9	9	*[HORIZONTAL TAB]*	41	29	)	73	49	I	105	69	i
10	A	*[LINE FEED]*	42	2A	*	74	4A	J	106	6A	j
11	B	*[VERTICAL TAB]*	43	2B	+	75	4B	K	107	6B	k
12	C	*[FORM FEED]*	44	2C	,	76	4C	L	108	6C	l
13	D	*[CARRIAGE RETURN]*	45	2D	-	77	4D	M	109	6D	m
14	E	*[SHIFT OUT]*	46	2E	.	78	4E	N	110	6E	n
15	F	*[SHIFT IN]*	47	2F	/	79	4F	O	111	6F	o
16	10	*[DATA LINK ESCAPE]*	48	30	0	80	50	P	112	70	p
17	11	*[DEVICE CONTROL 1]*	49	31	1	81	51	Q	113	71	q
18	12	*[DEVICE CONTROL 2]*	50	32	2	82	52	R	114	72	r
19	13	*[DEVICE CONTROL 3]*	51	33	3	83	53	S	115	73	s
20	14	*[DEVICE CONTROL 4]*	52	34	4	84	54	T	116	74	t
21	15	*[NEGATIVE ACKNOWLEDGE]*	53	35	5	85	55	U	117	75	u
22	16	*[SYNCHRONOUS IDLE]*	54	36	6	86	56	V	118	76	v
23	17	*[END OF TRANS. BLOCK]*	55	37	7	87	57	W	119	77	w
24	18	*[CANCEL]*	56	38	8	88	58	X	120	78	x
25	19	*[END OF MEDIUM]*	57	39	9	89	59	Y	121	79	y
26	1A	*[SUBSTITUTE]*	58	3A	:	90	5A	Z	122	7A	z
27	1B	*[ESCAPE]*	59	3B	;	91	5B	[	123	7B	{
28	1C	*[FILE SEPARATOR]*	60	3C	<	92	5C	\	124	7C	|
29	1D	*[GROUP SEPARATOR]*	61	3D	=	93	5D	]	125	7D	}
30	1E	*[RECORD SEPARATOR]*	62	3E	>	94	5E	^	126	7E	~
31	1F	*[UNIT SEPARATOR]*	63	3F	?	95	5F	_	127	7F	*[DEL]*

1.24 Numeric Data Representation

- pure binary
  - can be calculated directly
- ASCII binary
  - string of digits: "01010101"
- ASCII decimal
  - string of digits: "65"
- ASCII hexadecimal
  - string of digits: "9C"

1.25 What's Next (2 of 3)

- Welcome to Assembly Language
- Virtual Machine Concept
- Data Representation
- **Boolean Operations**

Lakehead UNIVERSITY

1.26 Boolean Operations

- NOT
- AND
- OR
- Operator Precedence
- Truth Tables

1.27 Boolean Algebra

- Based on symbolic logic, designed by George Boole
- Boolean expressions created from:
  - NOT, AND, OR

Expression	Description
$\neg X$	NOT X
$X \wedge Y$	X AND Y
$X \vee Y$	X OR Y
$\neg X \vee Y$	(NOT X) OR Y
$\neg (X \wedge Y)$	NOT (X AND Y)
$X \wedge \neg Y$	X AND (NOT Y)

1.27 NOT

- Inverts (reverses) a boolean value
- Truth table for Boolean NOT operator:

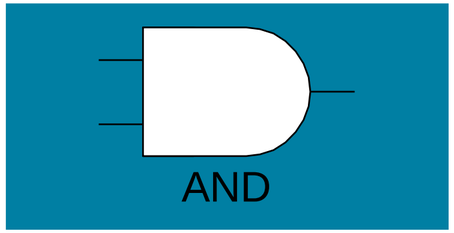
X	¬X
F	T
T	F

1.28 AND

- Truth table for Boolean AND operator:

X	Y	X ∧ Y
F	F	F
F	T	F
T	F	F
T	T	T

Digital gate diagram for AND

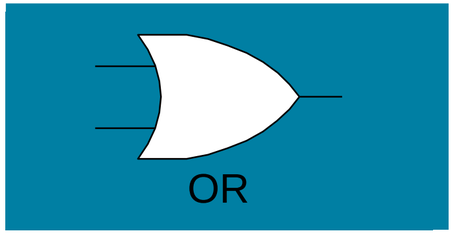


1.28 OR

- Truth table for Boolean OR operator:

X	Y	X ∨ Y
F	F	F
F	T	T
T	F	T
T	T	T

Digital gate diagram for OR



1.28 Operator Precedence

- Examples showing the order of operations:

Expression	Order of Operations
$\neg X \vee Y$	NOT, then OR
$\neg(X \vee Y)$	OR, then NOT
$X \vee (Y \wedge Z)$	AND, then OR

1.29 Truth Tables

- A Boolean function has one or more Boolean inputs, and returns a single Boolean output.
- A truth table shows all the inputs and outputs of a Boolean function

Example: $\neg X \vee Y$

X	$\neg X$	Y	$\neg X \vee Y$
F	T	F	T
F	T	T	T
T	F	F	F
T	F	T	T

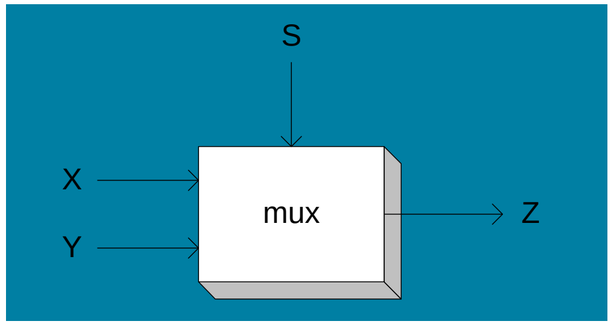
Example: $X \wedge \neg Y$

X	Y	$\neg Y$	$X \wedge \neg Y$
F	F	T	F
F	T	F	F
T	F	T	T
T	T	F	F

Lakehead UNIVERSITY

1.29 Truth Tables

Example: $(Y \wedge S) \vee (X \wedge \neg S)$



Two-input multiplexer

X	Y	S	$Y \wedge S$	$\neg S$	$X \wedge \neg S$	$(Y \wedge S) \vee (X \wedge \neg S)$
F	F	F	F	T	F	F
F	T	F	F	T	F	F
T	F	F	F	T	T	T
T	T	F	F	T	T	T
F	F	T	F	F	F	F
F	T	T	T	F	F	T
T	F	T	F	F	F	F
T	T	T	T	F	F	T

1.30 Summary

- Assembly language helps you learn how software is constructed at the lowest levels
- Assembly language has a one-to-one relationship with machine language
- Each layer in a computer's architecture is an abstraction of a machine
- layers can be hardware or software
- Boolean expressions are essential to the design of computer hardware and software