Chapter 1: Basic Concepts

Kip Irvine

Chapter Overview

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

Welcome to Assembly Language

- How does assembly language (AL) relate to machine language?
- How do C++ and Java relate to AL?
- Is AL portable?
- Why learn AL?

Assembly Language Applications

- Some representative types of applications:
  - Business application for single platform
  - Hardware device driver
  - Business application for multiple platforms
  - Embedded systems & computer games

Comparing ASM to High-Level Languages

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.

What's Next

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

Virtual Machine Concept

- Virtual Machines
- Specific Machine Levels

Virtual Machines

- Tanenbaum: 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

Translating Languages

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

↓

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

↓

Assembly Language:

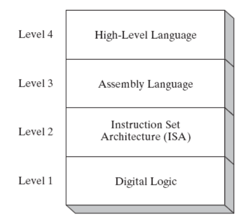
```
mov eax,A
mul B
add eax,C
call WriteInt
```

→

Intel Machine Language:

```
A1 00000000
F7 25 00000004
03 05 00000008
E8 00500000
```


Specific Machine Levels



High-Level Language

- Level 4
- Application-oriented languages
  - C++, Java, Pascal, Visual Basic . . .
- Programs compile into assembly language (Level 4)

Assembly Language

- Level 3
- Instruction mnemonics that have a one-to-one correspondence to machine language
- Programs are translated into Instruction Set Architecture Level - machine language (Level 2)

Instruction Set Architecture (ISA)

- Level 2
- Also known as conventional machine language
- Executed by Level 1 (Digital Logic)

Digital Logic

- Level 1
- CPU, constructed from digital logic gates
- System bus
- Memory
- Implemented using bipolar transistors

What's Next

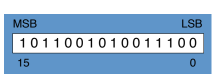
- Welcome to Assembly Language
- Virtual Machine Concept
- **Data Representation**
- Boolean Operations

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

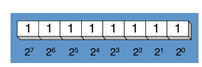
Binary Numbers

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



Binary Numbers

- 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

Table 1-3 Binary Bit Position Values.

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

Translating Binary to Decimal

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

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

D = binary digit

binary 00001001 = decimal 9:

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

Translating Unsigned Decimal to Binary

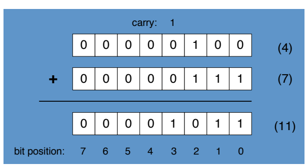
- Repeatedly divide the decimal integer by 2. Each remainder is a binary digit in the translated value:

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

Binary Addition

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



Integer Storage Sizes

Standard sizes:

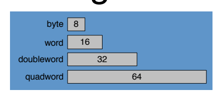


Table 1-4 Ranges of Unsigned Integers.

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 $(2^{64} - 1)$

What is the largest unsigned integer that may be stored in 20 bits?

Hexadecimal Integers

Binary values are represented in hexadecimal.

Table 1-5 Binary, Decimal, and Hexadecimal Equivalents.

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

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

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

Powers of 16

Used when calculating hexadecimal values up to 8 digits long:

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

Converting Decimal to Hexadecimal

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

decimal 422 = 1A6 hexadecimal

Hexadecimal Addition

- Divide the sum of two digits by the number base (16). The quotient becomes the carry value, and the remainder is the sum digit.

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

21 / 16 = 1, rem 5

Important skill: Programmers frequently add and subtract the addresses of variables and instructions.

Hexadecimal Subtraction

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

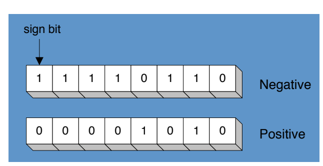
16 + 5 = 21

```
     −1
C6   75
A2   47
24   2E
```

Practice: The address of **var1** is 00400020. The address of the next variable after var1 is 0040006A. How many bytes are used by var1?

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

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

Binary Subtraction

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

```
  0 0 0 0 1 1 0 0        0 0 0 0 1 1 0 0
– 0 0 0 0 0 0 1 1  →   + 1 1 1 1 1 1 0 1
                         0 0 0 0 1 0 0 1
```

Practice: Subtract 0101 from 1001.

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

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 $(2^{63} - 1)$

Practice: What is the largest positive value that may be stored in 20 bits?

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*

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"

ASCII Code (7-bit)

American Standard Code for Information Interchange

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Extended ASCII Code (8-bit)

128	Ç	144	É	160	á	176	░	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	·
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

Source: www.LookupTables.com

What's Next

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

Boolean Operations

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

Boolean Algebra

- Based on symbolic logic, designed by George Boole
- Boolean expressions created from:

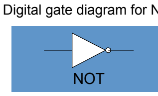
Expression	Description
¬X	NOT X
X ∧ Y	X AND Y
X ∨ Y	X OR Y
¬X ∨ Y	(NOT X) OR Y
¬(X ∧ Y)	NOT (X AND Y)
X ∧ ¬Y	X AND (NOT Y)

NOT

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

X	¬X
F	T
T	F

Digital gate diagram for NOT:

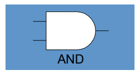


AND

- Truth table for Boolean AND operator:

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

Digital gate diagram for AND:

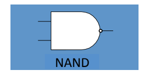


NAND

- Truth table for Boolean NAND operator:

X	Y	$\overline{X \wedge Y}$
F	F	T
F	T	T
T	F	T
T	T	F

Digital gate diagram for NAND:

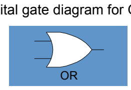


OR

- Truth table for Boolean OR operator:

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

Digital gate diagram for OR:

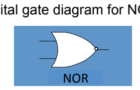


NOR

- Truth table for Boolean NOR operator:

X	Y	$\overline{X \vee Y}$
F	F	T
F	T	F
T	F	F
T	T	F

Digital gate diagram for NOR:



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

Truth Tables (1 of 3)

- 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

Truth Tables (2 of 3)

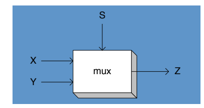
- Example: $X \wedge \neg Y$

X	Y	¬Y	X ∧ ¬Y
F	F	T	F
F	T	F	F
T	F	T	T
T	T	F	F

Truth Tables (3 of 3)

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

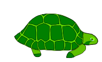
X	Y	S	Y ∧ S	¬S	X ∧ ¬S	(Y ∧ S) ∨ (X ∧ ¬S)
F	F	F	F	T	F	F
F	T	F	F	T	F	F
T	F	F	F	T	T	T
T	T	F	F	T	T	T
F	F	T	F	F	F	F
F	T	T	T	F	F	T
T	F	T	F	F	F	F
T	T	T	T	F	F	T



Two-input multiplexer

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



54 68 65 20 45 6E 64

What do these numbers represent?