

Conceptos básicos del lenguaje ensamblador¹

El lenguaje ensamblador exige una mayor capacidad de abstracción del programador. Si bien no es un lenguaje altamente estructurado, todas las usuales estructuras de control las podemos implementar en ensamblador. Una restricción asociada con el modelo de memoria de los microprocesadores hace que en cada paso del programa tengamos solo cinco localidades de memoria –llamadas registros (B. Brey, 2006, págs. 50-55)- disponibles para realizar las operaciones elementales, es por ello que en la programación en ensamblador se vuelven muy importantes las instrucciones para mover datos.

Cuando observamos los registros de memoria en la emulación de un programa, los valores guardados en ellos se encuentran por lo general en hexadecimal, por lo que se requiere la habilidad para reconocer el valor binario, decimal o ASCII que representa. Es por ello que se comenzará el curso con un repaso de la conversión de números entre diversos sistemas de representación.

También es muy importante que te familiarices con el emulador que se usará en el curso. El emulador [emu8086](#) (Ashkelon, 2014) permite la emulación de la ejecución de código ensamblador de 16 y 32 bits, presentando además los registros de memoria, la pila y la ventana de comandos.

- El lenguaje ensamblador es, por definición, el lenguaje de programación de más bajo nivel. Es decir, es el lenguaje más cercano entre el programador y el hardware. Hoy en día los lenguajes de programación dominantes son de alto nivel, son muy estructurados y son lejanos al hardware, al que sólo pueden acceder mediante la intervención del sistema operativo y microprogramas llamados drivers. Las técnicas de programación en estos lenguajes predominantes, a diferencia del ensamblador, son la Programación Orientada a Objetos y la Programación Orientada a Eventos.

En un ambiente altamente estructurado, el lenguaje ensamblador no es un lenguaje muerto.

Más bien, es un lenguaje reservado en aplicaciones que requieran un manejo muy efectivo de hardware, o de seguridad en el software. Es por ello que lo encontraremos –combinado con otros lenguajes- encapsulado en rutinas de video (como en muchos videojuegos), drivers y, por sobre todo, en la programación de microprocesadores y microcontroladores dedicados, lo que se denomina sistemas embebidos.

¹Documento elaborado por el Dr. Carlos Jiménez Gallegos.

El modelo de programación de ensamblador se basa en el uso de segmentos específicos de memoria con utilidad determinada, en la que se pueden colocar parámetros, datos fijos, el código mismo del programa y las rutinas a ejecutar; una serie de localidades fijas en la memoria llamadas registros, que son la principal comunicación del estado del proceso y del microprocesador. Finalmente, las instrucciones tienen una estructura simple de comandos cortos cuyos nombres son una ayuda nemotécnica de la acción que realizan, por ello, también a los comandos frecuentemente se les llama nemónicos (Irvine, 2007, págs. 619-645).

Si en este momento te pusieras a buscar cual programa, de los que usas diariamente, está codificado completamente en ensamblador, la respuesta más probable es que ninguno.

Pero tal vez te sorprenderá saber que en el proceso de compilación y creación de un ejecutable, muchos otros lenguajes pasan por ensamblar el código del programador, es decir, lo traduce a ensamblador. También podrás encontrar que muchas subrutinas de gráficos o video se han codificado en ensamblador, así como algunos drivers de dispositivos.

Hay también muchos dispositivos electrónicos como caminadoras, medidores de presión, cafeteras, etc., que poseen microprocesadores que son muchas veces programadas con ensamblador.

¿Sabías que la “computadora” de los autos modernos muchas veces se programa en ensamblador? Claro, no siempre es un humano quien escribe las líneas.

Sistemas numéricos

Dado que los sistemas electrónicos digitales se basan en bits que poseen en cada instante sólo dos posibles valores que son: el “1 lógico” igual conocido como cierto, verdadero, o simplemente “1”; y el “0 lógico”, también llamado falso, o “0”; las cantidades numéricas se deben representar en un sistema binario, lo que siempre es posible con la cantidad de bits adecuados.

La base del sistema binario es el dos, podemos decir que el binario representa números en base 2, donde el valor decimal de un guarismo es 2^n donde n es la posición del guarismo. Por ejemplo, 1010 en binario representa en decimal al número:

$$1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 = 1 * (8) + 0 * (4) + 1 * (2) + 0 * (1) = 10$$

Otro ejemplo es el número binario 1111 que en decimal es:

$$1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 = 1 * (8) + 1 * (4) + 1 * (2) + 1 * (1) = 15$$

Como la base es fija, a veces se le llama código 8421 cuando tenemos grupos de 4 bits, grupos que se les llama *nibbles*.

Para convertir un número en decimal a binario, solo hay que realizar la división por módulo de la cantidad sobre la base, luego el cociente entre la base, y repetir tantas veces como sea posible, e ir registrando los residuos. Por ejemplo, para el número 13 tenemos:

División	Cociente	Residuo
13/2	6	1
6/2	3	0
3/2	1	1
1/2	0	1

Por lo que en binario se representaría como: 1101 (ponga mucha atención en el orden en que se escribe).

Hagamos un segundo ejemplo con un número que conocemos de unos renglones atrás, el 15:

División	Cociente	Residuo
15/2	7	1
7/2	3	1
3/2	1	1
1/2	0	1

Lo que prueba que 1111 representa al 15 decimal. Ahora probemos con el número 10:

División	Cociente	Residuo
10/2	5	0
5/2	2	1
2/2	1	0
1/2	0	1

Lo que nos da 1010 tal como lo sabíamos renglones atrás.

Existen otros sistemas de representación de números decimales a través de la base 8 llamado **sistema octal** y la base 16, al que llamamos **sistema hexadecimal**, que resultan importantes pues se relacionan con el binario mediante la agrupación de conjuntos de 3 bits para el octal y de 4 bits para el hexadecimal.

El sistema octal permite la representación de números con los guarismos del 0 al 7, es decir, el número 10^0 (la o de octal) representa al número 8 decimal:

$$1 * 8^1 + 0 * 8^0 = 8$$

En el sistema hexadecimal tenemos quince guarismos disponibles: del 0 al 9 y las letras A, B, C, D, E, F, estas últimas representando respectivamente a los decimales 10, 11, 12, 13, 14 y 15. Nótese que el 15 decimal es un Fh en hexadecimal, 17^0 en octal y 1111 en binario. Revisa esta tabla con los primeros 15 valores decimales:

Decimal	Binario	Octal	Hexadecimal
0	0000	00	0
1	0001	01	1
2	0010	02	2
3	0011	03	3
4	0100	04	4
5	0101	05	5
6	0110	06	6
7	0111	07	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

La conversión a cualquier base, a partir del número decimal, funciona con la misma metodología. Pero ahora podemos convertir números más grandes a binario con la ayuda del hexadecimal. Por ejemplo, el número 100 decimal:

División	Cociente	Residuo
100/16	6	4
6/16	0	6

Es decir, que el 64h es el 100 decimal. Ahora trasformemos a binario cada guarismo del hexadecimal. Con la ayuda de la tabla sabemos que el 6 hexadecimal es 0110 y el 4 se representa como 0100.

Entonces, 100 decimal se representa como 0110 0100 en binario (el espacio es opcional, pero le da legibilidad al número). Ahora, si el número en binario lo agrupamos en conjuntos de tres bits tenemos (agregando un cero a la izquierda para completar 9 símbolos): 001 100 100 y usando la tabla en octal tenemos: 144°, es decir:

Decimal	Binario	Octal	Hexadecimal
100	0110 0100	144	64

Métodos de representación de datos en la computadora

Otra forma de representación de cantidades numéricas es el Método BCD (decimal codificado en binario, por sus siglas en inglés), cuya principal diferencia radica en que cada grupo de 4 bits sólo puede representar del 0 al 9 y las demás combinaciones son consideradas prohibidas. El BCD permite una muy fácil conversión de decimal a binario y viceversa, pues basta con convertir el grupo de 4 bits a decimal para obtener el guarismo que representa:

BCD	1001 0110 0111		
Decimal	9	6	7

Además de números nos interesa trabajar con datos alfabéticos, que son con cadenas de caracteres. A estos datos los trabajamos en binario con una codificación estandarizada de uso mundial llamada ASCII (Código estándar americano para el intercambio de información por sus siglas en inglés). El ASCII no sólo representa las letras, sino también algunas teclas especiales como la tecla enter, ESC, mayúsculas, etc. A continuación reproducimos la tabla básica del sitio www.asciitable.com

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

Esta tabla nos servirá de referencia al momento de interpretar lo que el usuario a tecleado, pues el *caracter* lo recuperaremos en formato hexadecimal. Por otro lado, el código ASCII también es utilizado para mandar los *caracteres* a la pantalla, los cuales pueden ser complementados con otros símbolos, como el alfabeto griego o pequeñas grecas para formar figuras en pantalla:

128	Ç	144	É	160	á	176	⌘	192	Ł	208	⌚	224	α	240	≡
129	ü	145	æ	161	í	177	⌘	193	ł	209	ƒ	225	β	241	±
130	é	146	Æ	162	ó	178	⌘	194	ƒ	210	π	226	Γ	242	≥
131	â	147	ô	163	ú	179		195	ƒ	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	ƒ	229	σ	245	∫
134	â	150	û	166	²	182	‡	198	†	214	ƒ	230	μ	246	÷
135	ç	151	ù	167	°	183	π	199	‡	215	‡	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	‡	200	⌚	216	‡	232	Φ	248	°
137	ë	153	Ö	169	ƒ	185	‡	201	ƒ	217	ƒ	233	⊖	249	.
138	è	154	Ü	170	ƒ	186	‡	202	⌚	218	ƒ	234	⊖	250	.
139	ï	155	◊	171	½	187	‡	203	ƒ	219	■	235	δ	251	√
140	î	156	£	172	¼	188	‡	204	‡	220	■	236	∞	252	∞
141	ï	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	ƒ	207	±	223	■	239	∩	255	

Source: www.LookupTables.com

En el caso de ensamblador recuperamos una cadena del teclado, a diferencia de otros lenguajes, caracter por caracter. Pero se cuenta con convertidores en línea que nos ayudan a verificar los datos codificados de forma rápida, por ejemplo, en el sitio:

http://www.roubaixinteractive.com/PlayGround/Binary_Conversion/Binary_To_Text.asp

Para finalizar, es muy importante tener presente que cuando un usuario introduce cantidades numéricas, el teclado regresa el valor hexadecimal del código ASCII del número tecleado, cifra por cifra. Por ejemplo, cuando tecleamos el “0” el teclado mandara un 30h equivalente a un 48 decimal. Siempre será necesario ajustar y armar el número que se introduce por el teclado.

¿Qué dirá este misterioso mensaje?

```
1010000101001000011011110110110001100001001000010010000010111111010100010111010111101001
001000000110100001100001011000110110010100111111
```