SIMULADOR DE ASSEMBLER

Ordenador VE-1004

Creado por: Renzo Mauricio Díaz Díaz

1. Marco teórico

Assembler o lenguaje ensamblador (para futuras menciones *Assembler*) es un lenguaje de programación de bajo nivel que apareció aproximadamente en el año 1949 y permite interactuar casi directamente con el hardware. Es la forma más común de interpretar el código máquina (cabe resaltar que *Assembler* no es código máquina, aunque son muy parecidos) y con él se pueden realizar operaciones muy sencillas que, en conjunto, forman programas complejos.

Es importante aclarar que no existe solo un lenguaje *Assembler*, por cada arquitectura de software existe un lenguaje *Assembler* capaz de comunicarse con ella. Por ejemplo, las instrucciones que utiliza un procesador *Intel* son diferentes, en cuanto a ejecución se refiere, a las que utiliza un procesador *AMD*. Por ejemplo, ambos tienen una instrucción que suma los valores de dos posiciones de memoria, pero se "refieren" a ellas de forma diferente.

"Referir una instrucción" es la manera mediante la cual un procesador identifica una acción y la diferencia de las demás. Este es un concepto importante para entender como funciona el lenguaje *Assembler*, las instrucciones se identifican mediante un código único el cual le indica al procesador exactamente qué debe hacer en un determinado momento.

Assembler es un lenguaje secuencial, lo cual significa que realiza una instrucción por turno o *tick* de reloj. El concepto de secuencia es uno de los pilares fundamentales de la programación, ninguna instrucción es más importante que otra, todas esperan su turno para ser ejecutadas y ninguna se adelanta o retrasa.

Finalmente, para empezar a entender la sintaxis del lenguaje, hay que conocer para qué hardware estará destinado el lenguaje *Assembler* que se explicará a continuación.

2. Características generales del ordenador

Para fines prácticos se modeló las partes funcionales básicas de un entorno de programación, compilación y ejecución de un software en *Assembler*, básicamente, se desarrolló un IDE completo.

El ordenador VE-1004 es una computadora que posee un procesador RD-1601 de una arquitectura de 16-bits con 8 registros multipropósito (A - H), un puntero a stack multipropósito (SP), un puntero a instrucción (IP), dos marcadores (Zero y Carry), un indicador de entrada (Input), un indicador de fallo (Fault). El ordenador se conecta mediante un bus virtual de 16 bits a una memoria RAM de 128 .

A. Registros multipropósito

Un registro multipropósito es, en términos simples un espacio dentro del propio procesador donde este puede acceder a la información almacenada de manera más directa. En el ordenador V-R1004 posee cuatro registros [A, B, C, D, E, F, G, H] y su función principal es almacenar datos y direcciones de memoria para que el procesador pueda acceder a ella.

Por ejemplo, si se desea acceder a la dirección de memoria 232 entonces hay dos maneras. Acceder a ella mediante una llamada a memoria ([232]) o guardar el valor 232 en un registro (por ejemplo, A) y hacer una llamada a memoria mediante registro ([A]). La ventaja de la segunda opción es que la dirección de memoria guardada en 'A' es versátil y puede variar durante la ejecución del programa. Por el contrario, con una llamada directa esto es imposible.

B. Puntero de pila multipropósito

El ordenador V-R1004 posee un puntero de pila (stack pointer SP) multipropósito. Este se comporte como un registro corriente, y puede ser utilizado como tal sin ningún inconveniente. Pero está directamente ligado con las instrucciones PUSH, POP, CALL, RET y otras. SP almacena

la posición de memoria en la que se encuentra la pila y se inicializa en la dirección de memoria que se defina en el IDE.

Las instrucciones PUSH y POP se encargan del manejo general de pila (ingreso y salida de datos), en cambio, CALL y RET manejan el uso de llamadas a funciones, que usan la pila para darle continuidad al programa, de forma que cuando se llama a una función (CALL) se guarda en la pila la posición de memoria de la instrucción siguiente y se salta a la parte del código deseada. Por el contrario, cuando se finaliza una llamada (RET), se salta a la posición de memoria que indica el primer elemento de la pila. En consecuencia, al finalizar una llamada la pila debe estar balanceada (se deben retirar de la pila todos los elementos que se ingresaron durante la ejecución de la función).

C. Marcadores Zero y Carry

Los marcadores (Flags) Zero y Carry indican valores booleanos (1 y 0).

Son modificables por, las instrucciones aritméticas (en caso se de *overflow* en alguna operación el marcador Carry almacenará el valor 1), las instrucciones de comparación (primero se guarda el valor de Zero en Carry y posteriormente guardará el resultado de la operación lógica en Zero) y las instrucciones de manejo de marcadores.

Las instrucciones de salto condicional se basan en los valores de Zero y Carry como fuente de información para funcionar correctamente.

D. Indicador de fallo Fault

En caso se trate de dividir entre cero o se dé un fallo en el procesador el indicador Fault cambiará a 1 y el ordenador se detendrá.

E. Memoria de Acceso Aleatorio, Input y Output

El ordenador posee 126 kilobytes de memoria de acceso aleatorio (Random Access Memory, RAM). En ella se almacenan las instrucciones en código máquina, los datos almacenados con la instrucción DB y los datos generados por el programa durante su ejecución.

Cada vez que el ordenador detecta que se ha pulsado una tecla el marcador Input cambia a 'true' y se guarda el código ASCII referente a la tecla en la posición de memoria indicada en el IDE.

El ordenador tiene dos formatos de salida: Un display de 36 caracteres ubicado secuencialmente desde la posición elegida en el IDE que ocupa 72 bytes o 36 posiciones de memoria RAM y un monitor de 128 x 128 pixeles a blanco y negro que ocupa 1 kilobyte de RAM.

3. Lenguaje ASM-1004

Existen 62 instrucciones diferentes en el lenguaje ASM-R1004 y se dividen en las siguientes categorías principales:

Categoría	Cantidad
Manejo de procesador	2
Manejo de datos	2
Aritméticas	10
Lógicas bit a bit	4
Manejo de bits	7
Comparación	3
Manejo de marcadores	10
Manejo de pila	6
Saltos condicionales	15
Llamada de funciones	2
Manejo de registros	1

Un término es un valor con el cual opera una instrucción. Existen cuatro tipos de términos:

Tipo	Ejemplo
Registro	A, B, C, D
Dirección de memoria	[12], [42], [145]
Dirección por registro	[A], [B], [C], [D]
Constante	43, 47, 194

Las constantes numéricas se pueden denotar en función de su base de la siguiente forma:

Base	Signo	Notacion	Numero decimal
Decimal	-	41	41
Hexadecimal	0x	0x3E	63
Octal	0o	0o12	10
Binaria	0b	0b10010	18

Una instrucción puede tender desde 0 hasta 2 términos o parámetros. Las instrucciones también se subdividen en tres categorías por la cantidad de términos que maneja:

Subcategoría	Cantidad
Dos términos	21
Un término	24
Sin términos	17

Durante la codificación, y para comodidad del usuario se puede hacer uso de marcadores que serán utilizados para saltar de una parte del código a otra con los saltos condicionales. Al momento de la compilación estos marcadores serán reemplazados por la posición de memoria asociadas a ellos al momento de su declaración.

La declaración de un marcador se realiza a través del uso de ‘:’ luego de la palabra que se desea asociar.

4. Instrucciones Código-Máquina

Una instrucción código-máquina del ordenador VR-1004 luce de esta manera:

0001 1010 0010 0011

Una cadena de 16 bits almacena la información necesaria para que el ordenador pueda ejecutar una instrucción correctamente. Si representamos la cadena anterior como un número hexadecimal se vería de esta forma:

1A23

Los dos primeros números (en el ejemplo 1A) codifican la instrucción que se desea usar. Cada instrucción está enlazada con un identificador. El tercer termino (2) indica el tipo del primer operando (Registro, Dirección de memoria, etc.). Y el cuarto y último termino el tipo del segundo operando. En caso la instrucción referenciada no posea ambos términos se denotará con un 0.

A continuación, una lista de todas las instrucciones que posee el ordenador y una breve explicación:

0	0	HLT	Detiene el procesador
1	1	DB A	Escritura en la memoria
2	2	MOV α, β	Copia el contenido de β en α
3	3	XCH α, β	Intercambia el contenido entre α y β
4	4	ADD α, β	Le suma a α e contenido de β
5	5	SUB α, β	Le resta a α e contenido de β
6	6	MUL α, β	Le multiplica a α el contenido de β
7	7	DIV α, β	Divide α entre β y almacena en α
8	8	IMUL α, β	División con signo
9	9	IDIV α, β	Multiplicación con signo
10	A	MOD α, β	Obtiene el modulo de la division α entre β
11	B	INC α	Aumenta α en 1
12	C	DEC α	Decrementa α en 1
13	D	INV α	Complemento a dos del numero
14	E	AND α, β	Opera α & β y lo almacena en α
15	F	OR α, β	Opera α | β y lo almacena en α
16	10	NOT α	Opera ~α y lo almacena en α
17	11	XOR α, β	Opera α ^ β y lo almacena en α
18	12	SHL α, β	Opera α << β y lo almacena en α
19	13	SHR α, β	Opera α >> β y lo almacena en α
20	14	REV α	Invierte la cadena de bits
21	15	BIT α, β	Guarda el valor del bit β de α en Zero

28	1C	FZR	Reset al marcador Zero (Se vuelve false)
29	1D	FZS	Set al marcador Zero (Se vuelve true)
30	1E	FZC	Complemento del marcador Zero (Se invierte)
31	1F	FCR	Reset al marcador Carry (Se vuelve false)
32	20	FCS	Set al marcador Carry (Se vuelve true)
33	21	FCC	Complemento del marcador Carry (Se invierte)
34	22	FIR	Reset al marcador Input (Se vuelve false)
35	23	FIS	Set al marcador Input (Se vuelve true)
36	24	FIC	Complemento del marcador Input (Se invierte)
37	25	PUSH α	Coloca α en la pila
38	26	POP α	Saca un valor de la pila y lo pone en α
39	27	PUSHF	Empuja los marcadores a la pila
40	28	POPF	Obtiene los marcadores de la pila
41	29	PUSHR	Empuja todos los registros a la pila
42	2A	POPR	Obtiene todos los registros de la pila
43	2B	JMP α	Salta a una posicion de memoria α
44	2C	JZ α	Zero = True
45	2D	JNZ α	Zero = False
46	2E	JC α	Carry = True
47	2F	JNC α	Carry = False
48	30	JI α	Input = True
49	31	JNI α	Input = False
50	32	JAT α	Zero = True && Carry = True
51	33	JATF α	Zero = True && Carry = False
52	34	JAFT α	Zero = False && Carry = True
53	35	JAF α	Zero = False && Carry = False
54	36	JOT α	Zero = True || Carry = True
55	37	JOF α	Zero = True || Carry = False
56	38	JOTF α	Zero = False || Carry = True
57	39	JOFT α	Zero = False || Carry = False
58	3A	CALL α	Inicia una llamada, se interrumpe con 'RET'
59	3B	RET	Finaliza la llamada (CALL)
60	3C	CREG	Limpia los registros (Vuelven a 0)
61	3D	CFLG	Limpia los marcadores (Vuelven a false)

5. Limitaciones

El ordenador VE-1004 es incapaz, por el momento, de procesar números decimales, únicamente números enteros. Y solo puede procesar números menores a 2^{16}.

6. Conclusiones

En conclusión, con el presente trabajo se demostró que es posible la virtualización de una arquitectura de software de 16 bits, emulando todos los procesos que sigue un procesador desde la codificación, pasando por la compilación para finalmente ejecutar un programa procedural.

Se espera que con este trabajo, futuros estudiantes puedan acercarse al lenguaje ensamblador de manera más didáctica y que aprendan a desarrollar algoritmos para este ordenador.

7. Fuentes

- Clark, J., (2009). But How Do It Know?, United States of America.
- UnioViedo. (2016). *Manual de Microcontroladores PIC*. Recuperado de https://www.unioviedo.es/ate/alberto/manualPic.pdf