

Integrantes

Pablo José Cárdenas Meneses - 2170080

Juan Pablo Beleño Mesa - 2204656

Diego Alejandro Arévalo Quintero - 2220066

Ana Valeria Barreto Téllez - 2215630

Brayan Julián Barrera Hernández - 2220097

Grupo BytesBuilders

Proyecto 6

Add:

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/6/add/Add.asm

// Computes R0 = 2 + 3 (R0 refers to RAM[0])

@2
D=A
@3
D=D+A
@0
M=D
```

El anterior código escrito en lenguaje ensamblador, guarda en el registro D el número 2 que al inicio con el primer @ se guarda en A, pasa lo mismo con 3, pero se hace una suma con el registro actual de D (que ahora es 2) y A (que ahora es 3) y su resultado se guarda nuevamente en el registro D. Finalmente con M se guar en RAM[0] (en este caso) lo que está en el registro D que actualmente es 5 (2+3).

Después de escribir el código ensamblador este se pasa al ensamblador, que traduce de este lenguaje computadora pero que sigue siendo comprensible para un humano directamente, a binario que este es comprendido por el procesador, en este caso uno simulado.

Para probar el programa Nand2Tetris ofrece un emulador de CPU que comprende las instrucciones que devuelve el ensamblador del mismo paquete. Se puede ver en la segunda imagen como el código es el mismo pero en ese caso lo que se carga fué lo que devuelve el ensamblador. Si este programa inicia recorre todas las líneas, realizando la lógica explicada anteriormente.

The screenshot shows the Nand2Tetris simulator interface. On the left, the 'Source' window displays assembly code. The 'Destination' window shows the binary representation of the memory location. A blue arrow points from the 'M=D' instruction in the Source window to the binary representation in the Destination window. The ROM and RAM windows show the current state of memory, with PC at 7 and A at 0.

Source	Destination
// This file is part of www.nand2tetris.org	00000000000000010
// and the book "The Elements of Computing Systems"	1110110000010000
// by Nisan and Schocken, MIT Press	00000000000000011
// File name: projects/6/add/Add.asm	1110000010010000
 	00000000000000000
// Computes R0 = 2 + 3 (R0 refers to RAM[0])	1110001100001000
@2	
D=A	
@3	
D=D+A	
@0	
M=D	

ROM	Asm
0	@2
1	D=A
2	@3
3	D=D+A
4	@0
5	M=D
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	
25	
26	
27	
28	

RAM	Value
0	5
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0

PC: 7 A: 0

Max:

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/06/max/Max.asm

// Computes R2 = max(R0, R1) (R0,R1,R2 refer to RAM[0],RAM[1],RAM[2])

@R0
D=M
@R1
D=D-M
@OUTPUT_FIRST
D;JGT
@R1
D=M
@OUTPUT_D
0;JMP
(OUTPUT_FIRST)
@R0
D=M
(OUTPUT_D)
@R2
M=D
(INFINITE_LOOP)
@INFINITE_LOOP
0;JMP
```

Su función es realizar una búsqueda del número mayor que este entre R1 y R0. El número que encuentra lo ubica en R2. El código comienza accediendo al contenido de la dirección de memoria R0 y lo guarda en el registro D. Después de cargar el valor de R0 en D, se resta el valor almacenado en R1 del valor en D. Esta operación nos da una diferencia ($D = R0 - R1$), que determinará cuál de los dos valores es mayor. Se verifica si D es positivo (es decir, si R0 es mayor que R1). Si $R0 > R1$, se realiza un salto a la etiqueta (OUTPUT_FIRST) para guardar el valor de R0 en R2. El comando D;JGT significa que el salto se realiza si D es mayor que cero. Si el salto no ocurre (es decir, $R0 \leq R1$), el código sigue su curso normal. En este caso, se carga el valor de R1 en D porque es el mayor o igual entre los dos. Después de cargar el valor de R1 o R2 dependiendo cual sea el mayor en D, el programa salta a la etiqueta (OUTPUT_D), donde se guarda el valor máximo (que ahora está en D) en R2.

Finalmente al igual que con el archivo anterior se pasa este código por el ensamblador que nos dará el .Hack para implementarlo en el emulador de CPU.

Source	Destination
// This file is part of www.nand2	0000000000000000
// and the book "The Elements of	1111110000010000
// by Nisan and Schocken, MIT Pre	0000000000000001
// File name: projects/6/max/Max.	1111010011010000
	0000000000001010
// Computes R2 = max(R0, R1) (R0	1110001100000001
// Usage: Before executing, put t	0000000000000001
	1111110000010000
// D = R0 - R1	0000000000001100
@R0	1110101010000111
D=M	0000000000000000
@R1	1111110000010000
D=D-M	0000000000000010
// If (D > 0) goto ITSR0	1110001100001000
@ITSR0	0000000000000110
D;JGT	1110101010000111
// Its R1	
@R1	
D=M	
@OUTPUT_D	
O;JMP	
(ITSR0)	
@R0	
D=M	
(OUTPUT_D)	
@R2	
M=D	
(END)	
@END	
O;JMP	

ROM	Asm		RAM	
0	@0		0	15
1	D=M		1	5
2	@1		2	15
3	D=D-M		3	0
4	@10		4	0
5	D;JGT		5	0
6	@1		6	0
7	D=M		7	0
8	@12		8	0
9	O;JMP		9	0
10	@0		10	0
11	D=M		11	0
12	@2		12	0
13	M=D		13	0
14	@14		14	0
15	O;JMP		15	0
16			16	0
17			17	0
18			18	0
19			19	0
20			20	0
21			21	0
22			22	0
23			23	0
24			24	0
25			25	0
26			26	0
27			27	0
28			28	0

PC 15
A 14

Rect:

1) Teniendo en cuenta las características del ensamblador, ¿Cuál es la principal limitante que observan? Justifique su respuesta.

Una de las principales limitaciones que se observan en el proyecto, que se centra en la construcción de un ensamblador, es la necesidad de gestionar adecuadamente las etiquetas y símbolos en el código fuente. Esto se debe a que el ensamblador debe traducir instrucciones de un lenguaje de alto nivel (o un lenguaje intermedio como Hack) a código máquina, lo que implica resolver direcciones y referencias de manera efectiva.

El ensamblador **debe recorrer dos veces** el código fuente para identificar y almacenar las etiquetas y reemplazarlas con las direcciones correspondientes. La implementación puede ser difícil debido a esto, especialmente si se tiene un código extenso con muchas referencias cruzadas.

El **número de símbolos disponibles es limitado**. Esto puede causar conflictos o requerir un manejo más cuidadoso de nombres, lo que afecta la legibilidad y mantenibilidad del código.

En comparación con otros lenguajes de programación, el ensamblador **no tiene muchas funciones avanzadas**. No puede realizar tareas complejas como la implementación de estructuras de control avanzadas o la manipulación de datos en tiempo de ejecución, lo que limita la complejidad de los programas que se pueden ensamblar.

La traducción de un lenguaje a otro siempre conlleva la posibilidad de errores, especialmente si el código fuente contiene errores de sintaxis o semántica. Puede ser un desafío que el ensamblador tenga una capacidad sólida para manejar estos casos.

Estas limitaciones hacen hincapié en la complejidad de la tarea de traducir instrucciones a código máquina y en la importancia de diseñar un ensamblador que funcione bien.

Bonus: ¿Por qué es tan importante el ensamblador?

La **traducción** de lenguaje ensamblador a código máquina permite que la CPU ejecute programas.



Control de hardware: facilita la interacción directa con hardware, esencial para controladores y sistemas embebidos.

Optimización: permite escribir código muy optimizado y eficiente, lo que mejora el rendimiento.

Comprensión interna: mejora las habilidades de depuración y optimización al aprender cómo funciona una computadora.

Interoperabilidad: Se puede usar con lenguajes de alto nivel para mejorar el rendimiento en áreas importantes.

Bibliografía:

-  Writing an Assembler - From NAND To Tetris
-  Symbol Tables - From NAND To Tetris
- Patterson, D. A., & Hennessy, J. L. (2009). *Computer organization and design : the hardware/software interface* (Fourth edition). Morgan Kaufmann Publishers.
- (Inggris) Jonathan Bartlett: *Programming from the Ground Up*. Bartlett Publishing, 2004.