# Integrantes

Pablo José Cárdenas Meneses - 2170080

Juan Pablo Beleño Mesa - 2204656

Diego Alejandro Arévalo Quintero - 2220066

Ana Valeria Barreto Téllez - 2215630

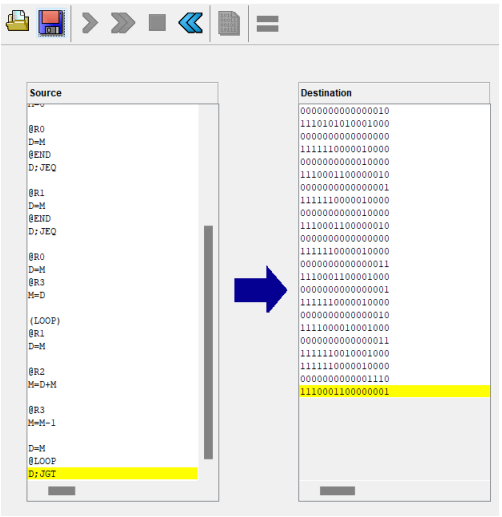
Brayan Julián Barrera Hernández - 2220097

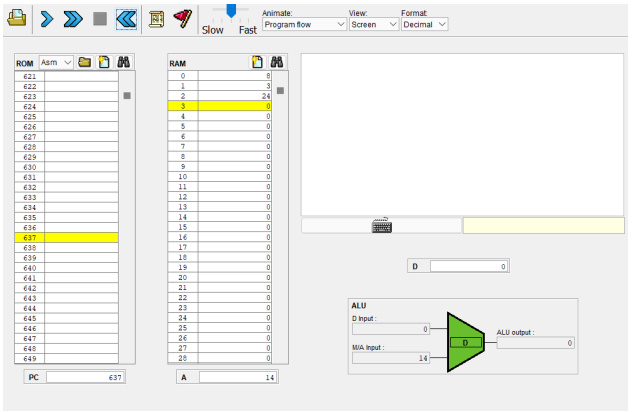
# Grupo BytesBuilders

# Proyecto 4

## Multiplicador con RAM:

```
@R2
M=0

@R0
D=M
@END
D;JEQ //jump if equal

@R1
D=M
@END
D;JEQ

@R0
D=M
@R3
M=D

(LOOP)
@R1
D=M

@R2
M=D+M

@R3
M=M-1

D=M
@LOOP
D;JGT
```



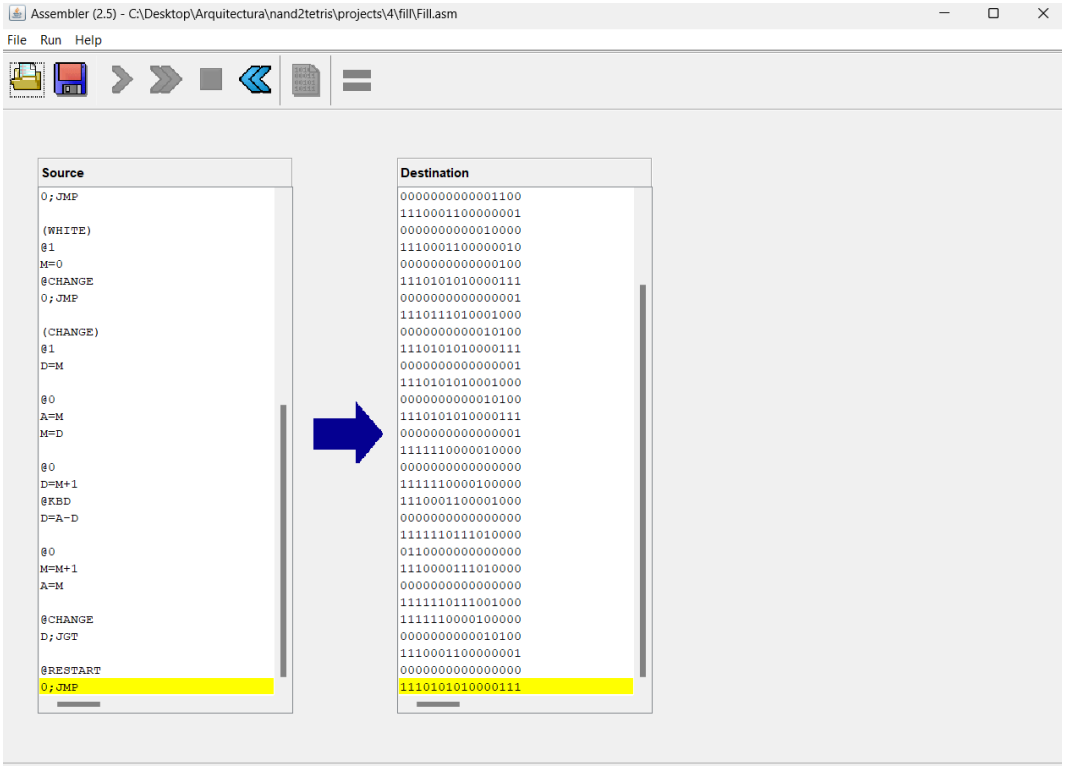

Se basa en multiplicar dos números con tres espacios en RAM, guardando el primer multiplicador en RAM 0, el segundo en RAM 1 y el resultado en RAM 2, primero se hace, se escribe en lenguaje ensamblador Hack en donde se escriben instrucciones directamente al procesador, posteriormente se utiliza un ensamblador para traducirlo en binario donde cada instrucción se compone de 16 bits.

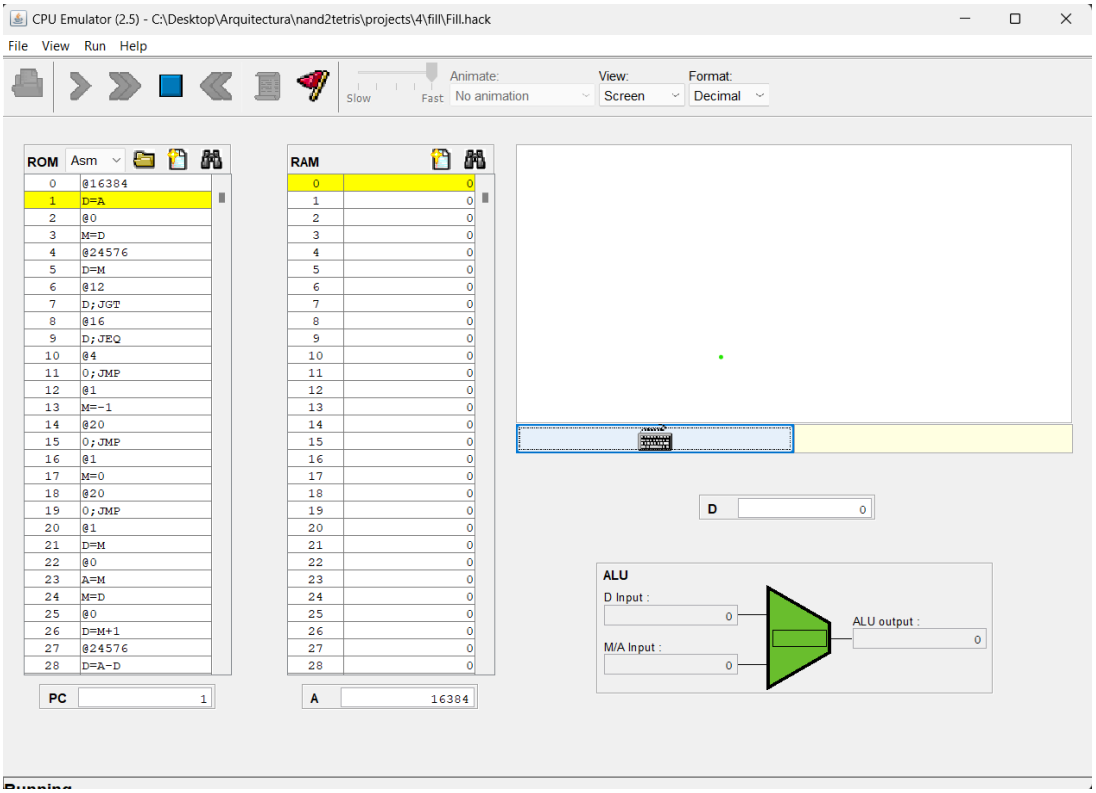
Con ayuda del ensamblador que se encuentra en la carpeta de herramientas de Nand2Tetris, se traduce este código en instrucciones de 16 bits. Para posteriormente cargarlo en un simulador de CPU donde recorre todas las instrucciones de manera lineal generadas por el ensamblador, entonces el número en RAM 0 y en RAM 1 se multiplican guardando este valor en RAM 3.

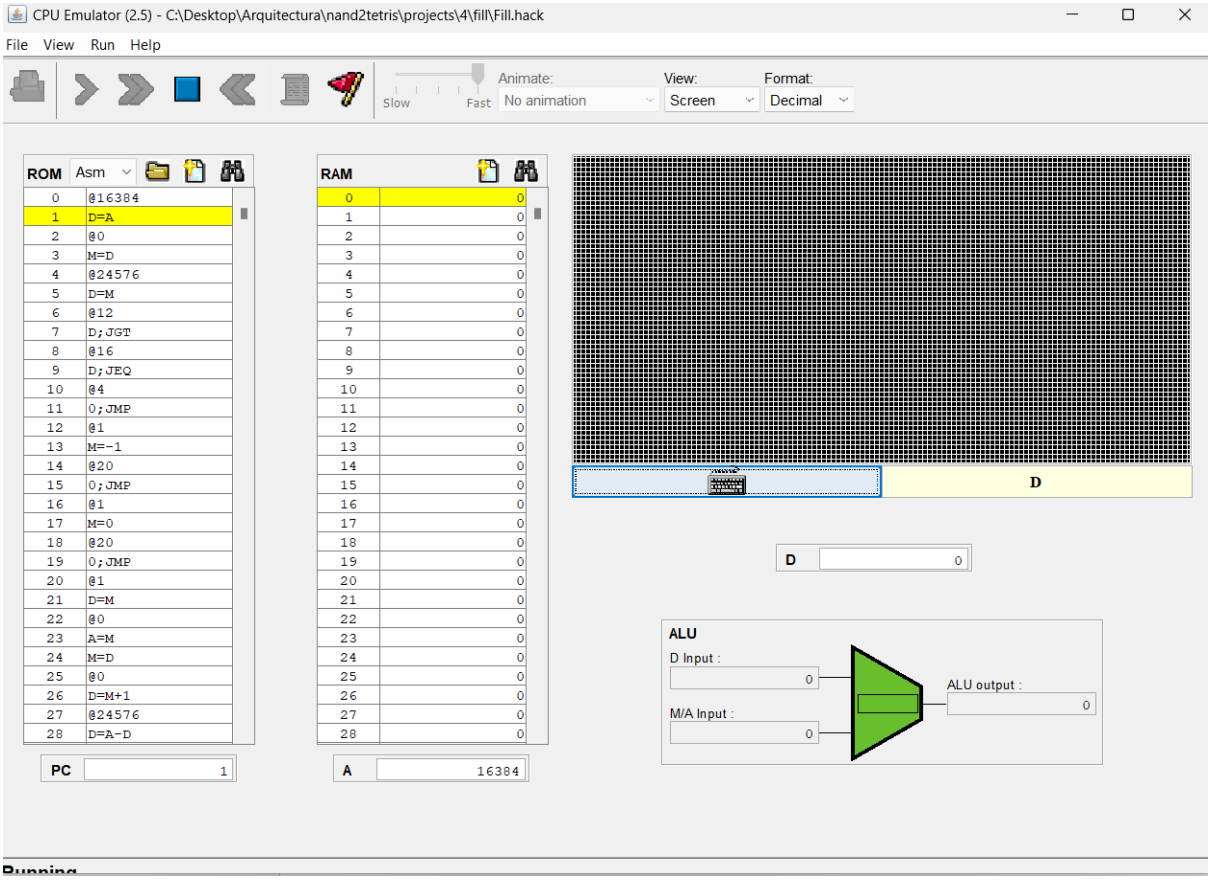
## I/O Handling:

```
(RESTART)
@SCREEN
D=A
@0
M=D

(KBDCHECK)

@KBD
D=M
@BLACK
D;JGT
@WHITE
D;JEQ

@KBDCHECK
0;JMP

(BLACK)
@1
M=-1
@CHANGE
0;JMP

(WHITE)
@1
M=0
@CHANGE
0;JMP

(CHANGE)
@1
D=M

@0
A=M
M=D

@0
D=M+1
@KBD
D=A-D
```

```
@0
M=M+1
A=M

@CHANGE
D;JGT

@RESTART
0;JMP
```


Assembler (2.5) - C:\Desktop\Arquitectura\nand2tetris\projects\4\fill\Fill.asm
File Run Help
Source
0;JMP
(WHITE)
@1
M=0
@CHANGE
0;JMP
(CHANGE)
@1
D=M
@0
A=M
M=D
@0
D=M+1
@KBD
D=A-D
@0
M=M+1
A=M
@CHANGE
D;JGT
@RESTART
0;JMP
Destination
0000000000001100
1110001100000001
0000000000010000
1110001100000010
0000000000000100
1110101010000111
0000000000000001
1110111010001000
0000000000010100
1110101010000111
0000000000000001
1110101010001000
0000000000010100
1110101010000111
0000000000000001
1111110000010000
0000000000000000
1111110000100000
1110001100001000
0000000000000000
1111110111010000
0110000000000000
1110000111010000
0000000000000000
1111110111001000
1111110000100000
0000000000010100
1110001100000001
0000000000000000
1110101010000111


CPU Emulator (2.5) - C:\Desktop\Arquitectura\nand2tetris\projects\4\fill\Fill.hack
File View Run Help
Animate:
Slow
Fast
No animation
View:
Screen
Format:
Decimal
ROM
Asm
0 @16384
1 D=A
2 @0
3 M=D
4 @24576
5 D=M
6 @12
7 D;JGT
8 @16
9 D;JEQ
10 @4
11 0;JMP
12 @1
13 M=-1
14 @20
15 0;JMP
16 @1
17 M=0
18 @20
19 0;JMP
20 @1
21 D=M
22 @0
23 A=M
24 M=D
25 @0
26 D=M+1
27 @24576
28 D=A-D
PC 1
RAM
0 0
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
A 16384
D 0
ALU
D Input : 0
M/A Input : 0
ALU output : 0




El programa ejecuta un bucle infinito que responde a la entrada del teclado. Cuando se presiona una tecla (cualquier tecla), el programa oscurece la pantalla, es decir, escribe "negro" en cada píxel. Después de tener el código listo tenemos que traducir el código ASM a código binario,utilizando el Assembler que nos dará el Fill.hack que finalmente será ejecutable en el CPUEmulator.

# Proyecto 5

**Memory:**

Define un componente Memory que actúa como un sistema de memoria en la arquitectura Hack. Permite cargar datos en dos módulos: RAM16K y Screen, basándose en la dirección proporcionada. Además, gestiona la entrada del teclado, asegurando que solo se procesen las direcciones válidas. Finalmente, utiliza multiplexores para determinar qué salida (de RAM, pantalla o teclado) se envía, facilitando el acceso a diferentes tipos de memoria y dispositivos de entrada/salida en el sistema.

```
CHIP Memory {
    IN in[16], load, address[15];
    OUT out[16];

    PARTS:
    DMux(in=load,sel=address[14],a=ramload,b=skload);
    DMux(in=skload,sel=address[13],a=sload,b=nothing);

    //load ram and screen
    RAM16K(in=in,load=ramload,address=address[0..13],out=ramout);
    Screen(in=in,load=sload,address=address[0..12],out=screenout);

    //Deal with keyboard, make sure that all bits in [0..12] are 0
    Keyboard(out=kbd);
    Or8Way(in=address[0..7], out=notkbd1);
    Or8Way(in[0..4]=address[8..12], in[5..7]=false, out=notkbd2);
    Or(a=notkbd1,b=notkbd2,out=notkbd);
    Mux16(a=kbd,b=false,sel=notkbd,out=kbdout);

    //Determinar cual es la salida
    Mux16(a=ramout,b=outsk,sel=address[14],out=out);
    Mux16(a=screenout,b=kbdout,sel=address[13],out=outsk);
}
```

**CPU:**

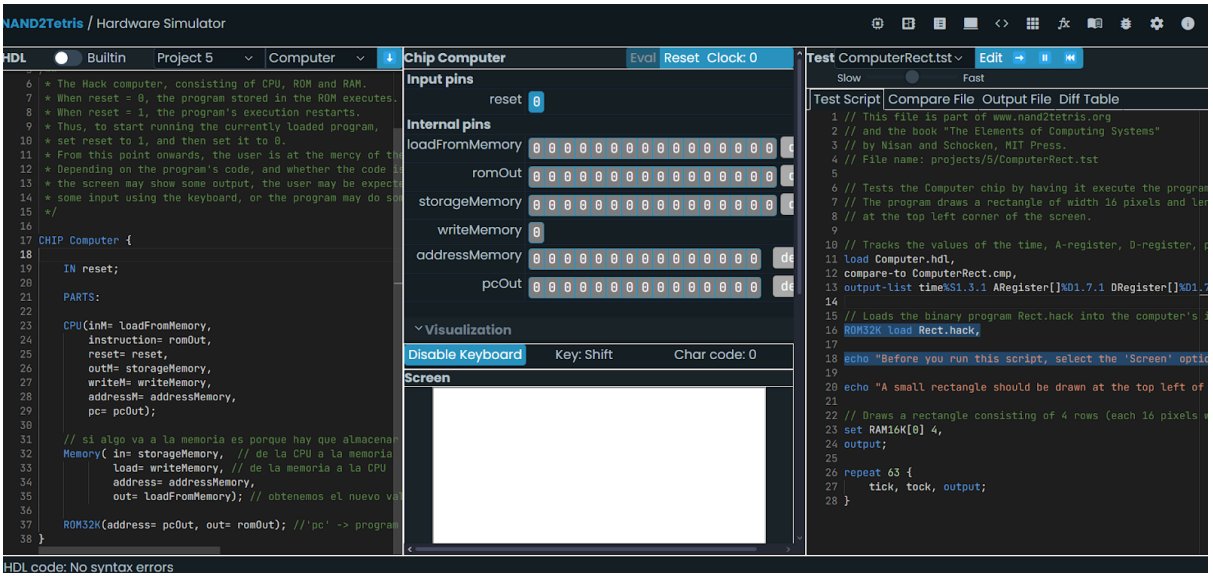
define un componente CPU que ejecuta instrucciones en la arquitectura Hack. Recibe un valor de entrada (inM), una instrucción y una señal de reinicio, y produce salidas que incluyen el valor a escribir en memoria (outM), una señal de escritura (writeM), la dirección de memoria (addressM) y la dirección de la próxima instrucción (pc). La CPU gestiona registros A y D, utiliza una ALU para realizar operaciones y controla el flujo del programa mediante la lógica de saltos condicionales.

```
CHIP CPU {

    IN  inM[16],      // M value input  (M = contents of RAM[A])
        instruction[16], // Instruction for execution
        reset;           // Signals whether to re-start the current
                         // program (reset=1) or continue executing
                         // the current program (reset=0).

    OUT outM[16],        // M value output
        writeM,          // Write into M?
        addressM[15],    // Address in data memory (of M)
        pc[15];          // address of next instruction

    PARTS:
  Not(in=instruction[15],out=ni);
  Mux16(a=outtM,b=instruction,sel=ni,out=i);

  Or(a=ni,b=instruction[5],out=intoA);
  ARegister(in=i,load=intoA,out=A,out[0..14]=addressM);

  And(a=instruction[15],b=instruction[12],out=AorM);
  Mux16(a=A,b=inM,sel=AorM,out=AM);


ALU(x=D,y=AM,zx=instruction[11],nx=instruction[10],zy=instruction[9]
,ny=instruction[8],f=instruction[7],no=instruction[6],out=outtM,out=
outM,zr=zr,ng=ng);

  And(a=instruction[15],b=instruction[4],out=intoD);
  DRegister(in=outtM,load=intoD,out=D);
```

```
  And(a=instruction[15],b=instruction[3],out=writeM);

  Not(in=ng,out=pos);
  Not(in=zr,out=nzr);
  And(a=instruction[15],b=instruction[0],out=jgt);
  And(a=pos,b=nzr,out=posnzr);
  And(a=jgt,b=posnzr,out=ld1);

  And(a=instruction[15],b=instruction[1],out=jeq);
  And(a=jeq,b=zr,out=ld2);

  And(a=instruction[15],b=instruction[2],out=jlt);
  And(a=jlt,b=ng,out=ld3);

  Or(a=ld1,b=ld2,out=ldt);
  Or(a=ld3,b=ldt,out=ld);

  PC(in=A,load=ld,inc=true,reset=reset,out[0..14]=pc);

}
```

## Computer

Una vez tenemos creado el CPU realmente la implementación de un computador es bastante sencilla, básicamente tendremos que enlazar el CPU, la memoria del dispositivo (implementación realizada en este mismo proyecto) y el ROM32K (que se trata del lugar donde se almacena la memoria del programa, es decir las instrucciones). Lo más importante de entender en esta implementación es que tipo de relación tienen los componentes, por ejemplo, si el input del CPU será el actual valor almacenado, o si será el nuevo valor que tengamos por almacenar.




[Computer, Concluded! From NAND To Tetris, Part 8]

## ¿Por qué el lenguaje de máquina es importante para definir la arquitectura computacional?

El lenguaje máquina es la interfaz que comunica directamente el hardware con el software, siendo este el nivel más bajo de abstracción, es el conjunto de instrucciones que puede ejecutar una computadora directamente, sin la necesidad de una traducción adicional. Este lenguaje está directamente ligado con las instrucciones que soporta un procesador, como cargar y almacenar datos en memoria, operaciones aritméticas, comparaciones y saltos condicionales.

Ahora, dado que el lenguaje máquina está directamente ligado con su hardware, este permite optimizar un conjunto específico de tareas como el procesamiento paralelo o rendimiento en operaciones aritméticas complejas o muy extensas. Aunque cada procesador maneja y entiende su propio lenguaje de máquina, entonces cambiando esta arquitectura las instrucciones dejan de funcionar, como sucede con el lenguaje C y no con Python, por ejemplo o Java.

## ¿Qué diferencia ven entre arquitectura computacional, arquitectura de software y arquitectura del sistema?

Arquitectura Computacional se refiere a la organización y diseño de los componentes físicos de un sistema de cómputo, es decir, el hardware. Centrándose en aspectos como en la unidad Central de Proceso (CPU), memoria, bus de datos,

unidades de almacenamiento, etc. Cómo los datos se mueven dentro del sistema (controladores, caché), el diseño de conjuntos de instrucciones (ISA - Instruction Set Architecture), optimización del rendimiento y consumo energético. Un ejemplo podría ser la arquitectura x86 de Intel.

Arquitectura de Software  se refiere al diseño y organización de los componentes de software de un sistema, se centra en cómo los módulos de software interactúan entre sí, basándose en el diseño de los componentes de software, patrones de diseño, interacciones entre componentes, interfaces, y flujos de datos. Un ejemplo podría ser un sistema basado en microservicios donde cada módulo es independiente y se comunica mediante APIs.

Arquitectura del Sistema es más amplia que la arquitectura de software o la computacional y se refiere a cómo todos los componentes, tanto hardware como software, interactúan en un sistema completo, esta integra el hardware, software, redes, y otras partes del sistema para formar una solución completa, considera la distribución del sistema, infraestructura, y comunicaciones entre componentes, además de incluir aspectos como escalabilidad, fiabilidad, seguridad y disponibilidad en un nivel global, implica decisiones de integración entre hardware, software, almacenamiento, red, y otros elementos. Un ejemplo puede ser el diseño de un sistema distribuido basado en la nube donde los servicios, bases de datos y hardware están distribuidos geográficamente y necesitan interactuar de manera eficiente.