Arquitectura de Computadoras

Diseño de una computadora simple con lógica de control alambrada

Mario Becerra
Luis Daniel Hernández

Febrero de 2017

1. Introducción

El objetivo de este trabajo es comprender claramente la integración de un computador sencillo. En particular, entender la forma en que las funciones de control gobiernan los demás componentes del sistema.

Según [2], un diseñador o arquitecto de computadoras debe determinar qué atributos son importantes para una nueva computadora y además diseñar la computadora de tal forma que se maximice el desempeño, y al mismo tiempo mantener dentro de un rango los costos. La arquitectura de computadoras se refiere al diseño del proceso que involucra la construcción de los tres aspectos generales de una computadora: repertorio de instrucciones, organización y hardware.

Las computadoras actuales están basadas en el modelo de Von Neumann, o arquitectura de Von Neumann, desarrollado en Princeton en la década de 1940. Para llegar a este modelo, primero John Mauchly y John Eckert construyeron la ENIAC (*Electronic Numerical Integrator And Computer*) con tubos de vacío. Completada en 1946, la ENIAC fue la primera computadora electrónica digital de propósito general. La principal desventaja de la ENIAC era que tenía que ser programada manualmente con cables y apagadores, lo cual hacía muy difícil y tediosa la tarea de hacer programas. Esta tarea podía facilitarse si el programa pudiera ser representado en una forma tal que el programa pudiera ser almacenado junto con los datos. Esta idea se le atribuye a John Von Neumann, un matemático que era consultor en el proyecto de ENIAC. En 1946, Von Neumann y su equipo empezaron el diseño de una computadora con estas características. El modelo de Von Neumann se refiere a una computadora que es construida con estas características [3].

El ciclo esencial de cada instrucción en una computadora que sigue el modelo de Von Neumann es *fetch* y *Execute*. Es decir, se lee una instrucción de memoria, se ejecuta y luego se lee otra instrucción, para así seguir indefinidamente.

2. Desarrollo

2.1. Especificaciones del sistema digital

La computadora desarrollada en este trabajo tiene las siguientes componentes:

- MAR (*Memory Address Register*): Registo que direcciona localidades de memoria
- MBR (*Memory Buffer Register*): Almacena temporalmente el dato leído o el que se va a escribir en memoria.
- PC (*Program Counter*): Registro que apunta a la siguiente instrucción a ser ejecutada. Tiene guardada la dirección de memoria donde está guardada la siguiente instrucción a ser ejecutada. Ante cada instrucción ejecutada, actualiza su valor acordemente.
- IR (*Instruction register*): Contiene el código de operación de la instrucción a ejecutar.

- Unidad de control: Conjunto de compuertas lógicas cuyo objetivo es orquestar las operaciones de la computadora a través de señales binarias que son enviadas a los demás elementos. Estas señales binarias y sus combinaciones determinan qué microoperación debe ejecutarse en qué tiempo.
- Contador de tiempo (T): Es un registro contador que aumenta su valor con cada pulso del reloj. Se reinicia (se le asigna el valor de 0) en ciertas instrucciones, e.g., al cargar un dato a un registro. Va conectado a la unidad de control para que en conjunto con las compuertas lógicas se sepa cuándo ejecutar qué microoperación.

Se tienen además las siguientes especificaciones:

- La longitud de palabra es de ocho bits.
- Se utilizará como RAM una memoria principal de 64×8 (2^6 palabras de 8 bits cada una).
- La computadora puede ejecutar las diez instrucciones que se especifican en la tabla 1. En donde, *dir* es una dirección de memoria, la cual se especifica en la siguiente palabra del formato de instrucción y N es el bit más significativo del registro B.
- Un programa finaliza con una instrucción $GOTO$ en esa misma dirección. El proceso se podrá detener por medio de una interrupción externa.
- El inicio de un programa se realiza por medio de una interrupción externa.

Tabla 1: Microoperaciones que puede ejecutar la computadora creada

Código de operación	Mnemónico	Descripción
00	MOVR	$B \leftarrow A$
01	LD *dir*	$A \leftarrow M[dir]$
02	COMPL	$B \leftarrow \overline{B}$
03	INCB	$B \leftarrow B + 1$
04	ADDR	$B \leftarrow A + B$
05	CLR A	$A \leftarrow 0$
06	SAVE *dir*	$M[dir] \leftarrow A$
07	INCA	$A \leftarrow A + 1$
08	GOTO *dir*	$PC \leftarrow dir$
09	JPN *dir*	Si $N = 1$, $PC \leftarrow dir$, si $N = 0$ se ejecuta la siguiente instrucción

La computadora fue implementada en *Logisim* [1]. El diagrama de la implementación se puede ver en la figura 1. En ella, están presentadas las componentes descritas anteriormente, es decir; reloj, MBR, MAR, PC, IR, RAM, unidad de control, y los registros A y B, además de dos unidades para calcular la suma y el complemento a unos, y las compuertas lógicas necesarias para la ejecución. Se agregó además un *display* de 7 segmentos para mostrar el contador T del tiempo. Las conexiones entre estas componentes están hechas siguiendo las asignaciones directas en la lógica de control. Por ejemplo, en el ciclo *fetch*, PC manda directamente su contenido a MAR, por lo que hay una conexión directa entre ellos. Los detalles de la implementación de las microoperaciones y la lógica de control están en las siguientes subsecciones.

2.2. Microoperaciones

En la tabla 2 se presenta el conjunto de microoperaciones de la computadora. Para cada una de las 10 operaciones incluidas en la tabla 1, se muestra la señal necesaria para ejecutar la microoperación. En este caso, q_i se refiere a la i-ésima operación de la tabla 1 y t_j al j-ésimo tiempo guardado en el registro contador T. La función $COMPL$ se refiere al complemento a unos de la palabra que está guardada en el registro B.

- **Fetch**

  t_0: MAR $\leftarrow$ PC
  t_1: MBR $\leftarrow$ M[MAR],
  PC $\leftarrow$ PC + 1
  t_2: IR $\leftarrow$ MBR

- **MOVR**

  q_0t_3: B $\leftarrow$ A,
  T $\leftarrow$ 0

- **LD dir**

  q_1t_3: MAR $\leftarrow$ PC
  q_1t_4: MBR $\leftarrow$ M[MAR],
  PC $\leftarrow$ PC + 1
  q_1t_5: MAR $\leftarrow$ MBR
  q_1t_6: MBR $\leftarrow$ M[MAR]
  q_1t_7: A $\leftarrow$ MBR,
  T $\leftarrow$ 0

- **COMPL**

  q_2t_3: B $\leftarrow$ COMPL(B),
  T $\leftarrow$ 0

- **INCB**

  q_3t_3: B $\leftarrow$ B + 1,
  T $\leftarrow$ 0

- **ADDR**

  q_4t_3: B $\leftarrow$ A + B,
  T $\leftarrow$ 0

- **CLR A**

  q_5t_3: A $\leftarrow$ 0,
  T $\leftarrow$ 0

- **SAVE dir**

  q_6t_3: MAR $\leftarrow$ PC
  q_6t_4: MBR $\leftarrow$ M[MAR],
  PC $\leftarrow$ PC + 1
  q_6t_5: MAR $\leftarrow$ MBR
  q_6t_6: MBR $\leftarrow$ A
  q_6t_7: M[MAR] $\leftarrow$ MBR,
  T $\leftarrow$ 0

- **INCA**

  q_7t_3: A $\leftarrow$ A + 1,
  T $\leftarrow$ 0

- **GOTO dir**

  q_8t_3: MAR $\leftarrow$ PC
  q_8t_4: MBR $\leftarrow$ M[MAR]
  q_8t_5: PC $\leftarrow$ MBR,
  T $\leftarrow$ 0

- **JPN dir**

  q_9t_3: MAR $\leftarrow$ PC
  q_9t_4: MBR $\leftarrow$ M[MAR],
  PC $\leftarrow$ PC + 1
  $q_9t_5\overline{N}$: T $\leftarrow$ 0
  q_9t_5N: PC $\leftarrow$ MBR,
  T $\leftarrow$ 0

Tabla 2: Microoperaciones de la computadora

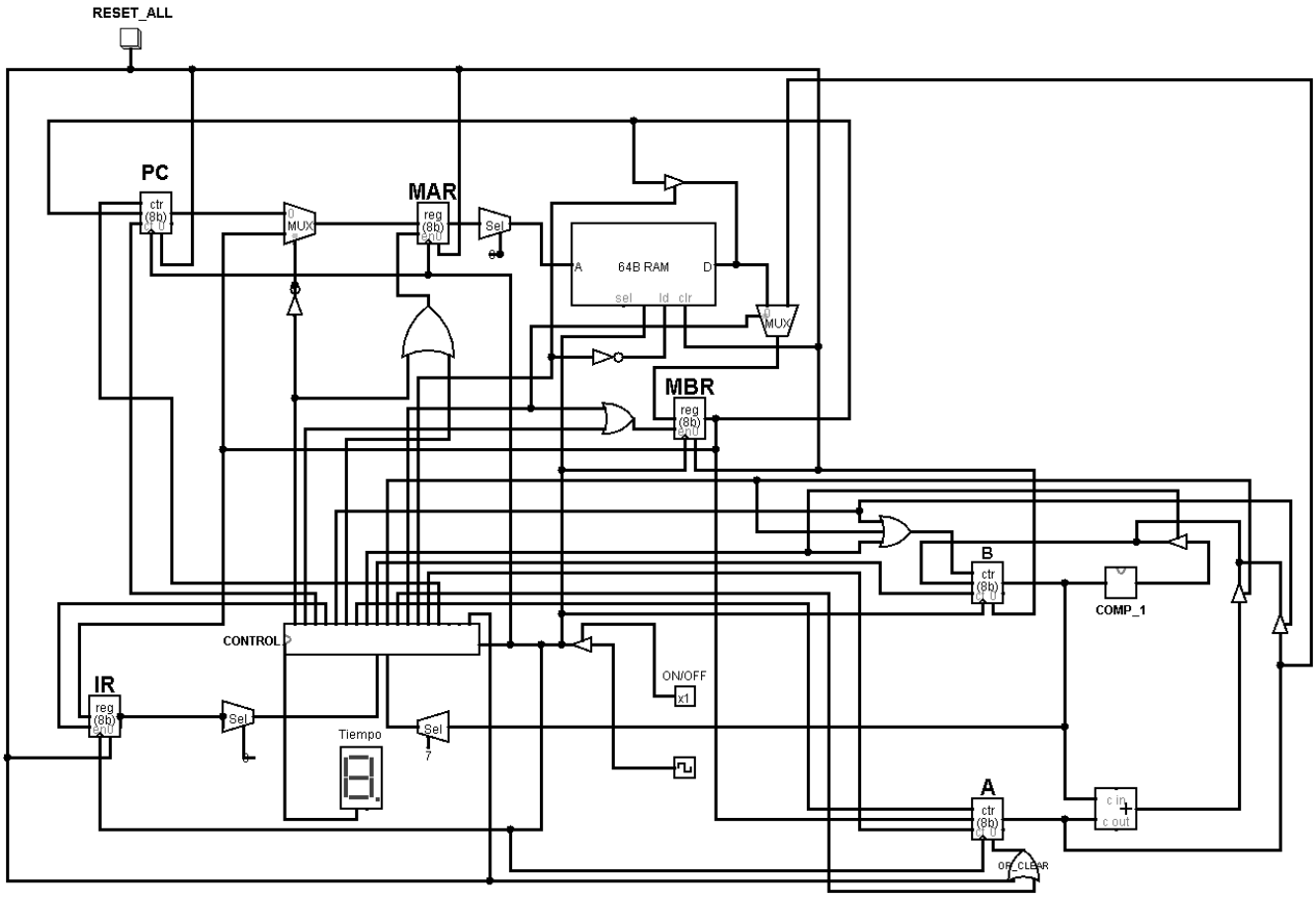

Figura 1: Computadora implementada en *Logisim*

2.3. Unidad de control

A partir de las operaciones y tiempos anteriores, se crean las siguientes señales para que la unidad de control funcione.

- MAR $\leftarrow$ PC: $X_1 = t_0 + q_1t_3 + q_6t_3 + q_8t_3 + q_9t_3$
- MBR $\leftarrow$ M[MAR]: $X_2 = t_1 + q_1t_4 + q_1t_6 + q_6t_4 + q_8t_4 + q_9t_4$
- PC $\leftarrow$ PC + 1: $X_3 = t_1 + q_1t_4 + q_6t_4 + q_9t_4$
- IR $\leftarrow$ MBR: $X_4 = t_2$
- B $\leftarrow$ A: $X_5 = q_0t_3$
- MAR $\leftarrow$ MBR: $X_6 = q_1t_5 + q_6t_5$
- A $\leftarrow$ MBR: $X_7 = q_1t_7$
- B $\leftarrow$ COMP(B): $X_8 = q_2t_3$
- B $\leftarrow$ B + 1: $X_9 = q_3t_3$
- B $\leftarrow$ A + B: $X_{10} = q_4t_3$
- A $\leftarrow$ 0: $X_{11} = q_5t_3$
- MBR $\leftarrow$ A: $X_{12} = q_6t_6$

- M[MAR] $\leftarrow$ MBR: $X_{13} = q_6 t_7$
- A $\leftarrow$ A + 1: $X_{14} = q_7 t_3$
- PC $\leftarrow$ MBR: $X_{15} = q_8 t_5 + q_9 t_5 N$
- $X_{16} = q_9 t_5 \overline{N}$
- T $\leftarrow$ 0: $X_{17} = X_5 + X_7 + X_8 + X_9 + X_{10} + X_{11} + X_{13} + X_{14} + X_{15} + X_{16}$

La unidad de control se puede ver implementada en la figura 2. Donde se tienen dos decodificadores, uno de ellos conectado al IR y otro al reloj; se tiene N, que es el bit más significativo del registro B, un contador T que aumenta con los pulsos del reloj; y compuertas lógicas que dan como salida 17 señales que son utilizadas en el resto de la computadora para ejecutar las microoperaciones. Estas 17 señales corresponden a las $X_1, \ldots, X_{17}$ que se acaban de definir.

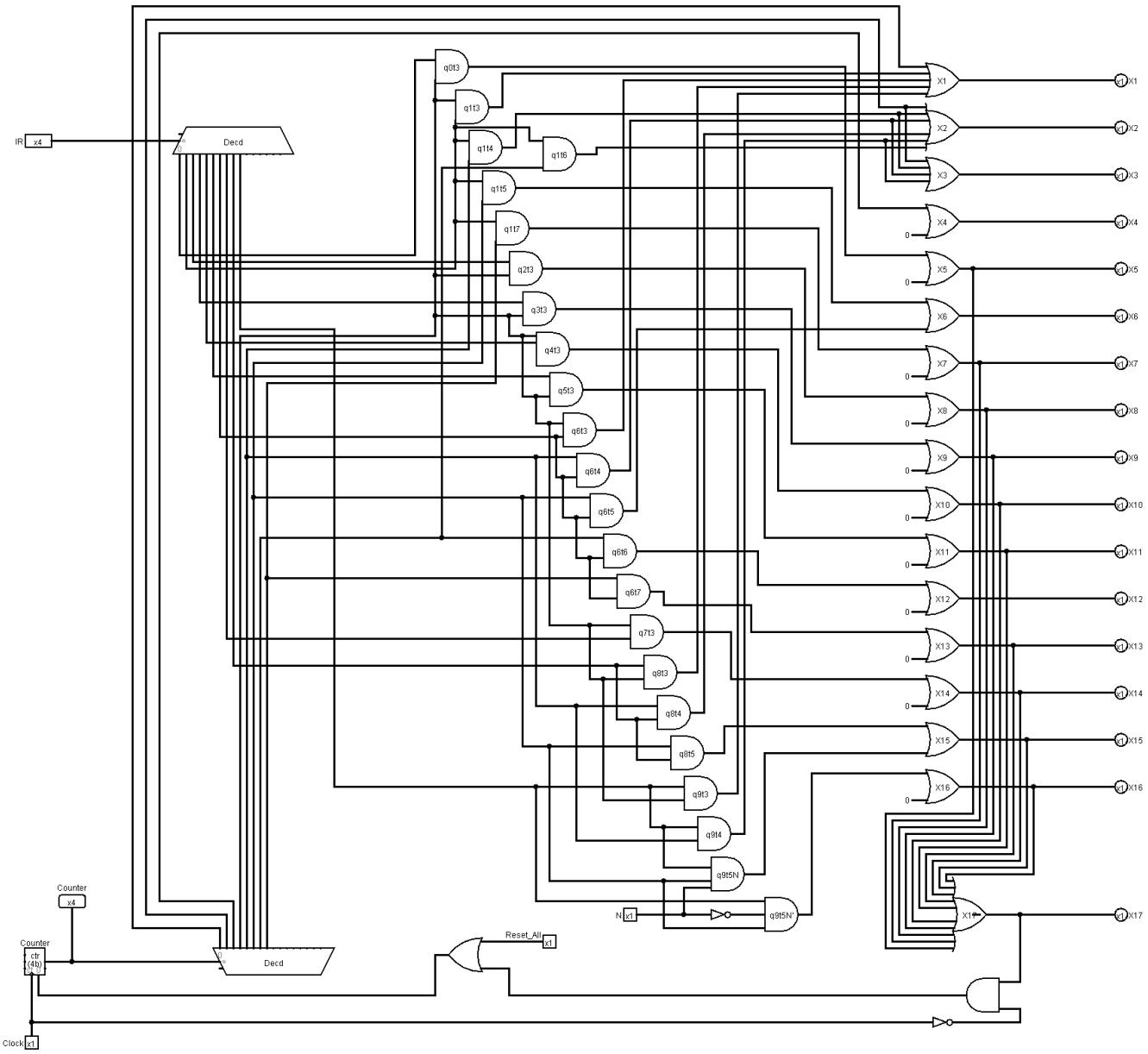

Figura 2: Unidad de control alambrada implementada en *Logisim*

3. Simulación de código de alto nivel

Con las instrucciones de la computadora anterior, se diseñó un programa en un pseudo lenguaje ensamblador para ejecutar el siguiente operador ternario de C.

```
x = (x < y) ? 0 : x + 1
```

Se asume que `x` se encuentra en la localidad de memoria 20, `y` en la 21 y que el programa se carga a partir de la localidad 0.

El código en el pseudo lenguaje ensamblador que ejecuta el operador ternario es el siguiente:

```
1       LD 20;
2       MOVR;
3       LD 21;
4       COMPL;
5       INCB;
6       ADDR:
7       JPN L1;
8       CLRA;
9       SAVE 20;
10      GOTO L2;
11
12      L1:
13      LD 20;
14      INCA;
15      SAVE 20;
16      GOTO L2;
17
18      L2:
19      GOTO L3;
20
21      L3:
22      GOTO L2;
```

La primera instrucción carga en el registro A el valor que está en la dirección de memoria 20, es decir, el valor de `x`. Posteriormente, se mueve el valor del registro A al registro B, para a continuación cargar en el registro A el valor de la dirección de memoria 21, es decir, `y`. La cuarta instrucción calcula el complemento a uno del registro B, para que en la quinta incremente su valor en 1; esto es equivalente a calcular el complemento a dos del registro B, lo cual a su vez es lo mismo que multiplicar `y` por -1. La instrucción 6 suma los valores del registro A y del registro B para guardarlos en el registro B (es decir, calcula `x - y`).

La instrucción 7 es la esencia del `if`, pues si el bit más significativo de B es igual a 0, significa que `x > y`. En ese caso sigue ejecutando las instrucciones que siguen, es decir, `CLRA` y `SAVE 20`, las cuales *resetean* el registro A y guardan ese valor en la localidad de memoria 20, i.e., hacen `x = 0`.

Por el otro lado, si el bit más significativo de B es igual a 1, significa que `x < y`. En la sintaxis de este pseudo lenguaje ensamblador, se le indica que brinque a la sección `L1` para ejecutar las instrucciones que están en las líneas 13 a 16. Lo que se está haciendo en la computadora, es brincar a la dirección 50 (elegido arbitrariamente), para cargar en el registro A el valor de la localidad de memoria 20 (`LD 20`), incrementar el valor del registro A (`INCA`) y guardar el resultado en la localidad 20, es decir, se hace `x = x + 1`.

Finalmente, en ambos casos, en las líneas 10 y 16, se manda a la sección `L2`, la cual manda a la sección `L3`, quien manda a la sección `L3`, etc.; es decir, se cicla.

A continuación, se presentan los registros de la memoria RAM en la computadora una vez que se guarda el programa para ejecutar la condición ternaria. Cada número de dos dígitos es un número hexadecimal que representa una instrucción, una dirección o un valor. La localidad 0 es la que está hasta la izquierda en la línea 1, la localidad 2 es la que le sigue a la derecha, y así sucesivamente hasta llegar a la localidad 63 que es la que está hasta la derecha en la línea 8.

```
01 14 00 01 15 02 03 04
09 32 05 06 14 08 0c 00
00 00 00 00 f8 f4 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 01 14 07 06 14 08
37 00 00 00 00 00 00 00
```

Para ejecutar el programa, el PC debe contener el número 0 pues se asume que el programa empieza en la localidad 0. Lo primero que se hace es ejecutar el ciclo *fetch* y mandar al IR la instrucción `01` (`LD`), la cual carga el valor de la dirección de memoria `0x14` (20 en notación decimal) en el registro A. La localidad 2 tiene la instrucción `00`, esto significa que debe ejecutar el `MOVR`. Las localidades 3 y 4 indican que se haga un `LD` de la dirección `0x15` (21 en decimal) en el registro A. La localidad 5 manda ejecutar el complemento a uno del registro B (*opcode* `02`) para luego en la 6 incrementar el valor de B (*opcode* `03`). Posteriormente, en la 7 se ejecuta la suma de los registros A y B (*opcode* `04`).

En la localidad 8 se ejecuta el `JPN` (*opcode* `09`). En caso de que `N` sea 0, se ejecuta la siguiente instrucción de memoria que está en la localidad 10, en otro caso, brinca a la localidad `0x32` (50 en decimal).

La localidad 10 tiene el valor de `05`, por lo que si `N` es 0, se ejecutará el comando `CLRA` y posteriormente `SAVE` en la localidad `0x14` (20 en decimal). Después, en la localidad 13, se manda a llamar un `GOTO` a la misma localidad 13 (`0x0c` en decimal) para ciclar el programa y darlo por terminado.

En el caso que `N` sea 1, el PC se actualiza con el valor `0x32` (50 en decimal) y ejecuta un `LD` (*opcode* `01`) de la localidad 20 (`0x14` en hexadecimal). En la localidad 52 hace un incremento (`INCA`, *opcode* `07`) y después `SAVE` en la localidad `0x14` (20 en decimal). Finalmente, en la localidad 55, se manda a llamar un `GOTO` a la misma localidad 55 (`0x37` en decimal) para ciclar el programa y darlo por terminado.

4. Conclusiones

El diseño de una computadora, incluso una sencilla como la mostrada aquí, es una tarea complicada y laboriosa que no debe ser tomada a la ligera. Hay muchos detalles que se tienen que tomar en cuenta que si no se toman con el debido cuidado pueden hacer que la funcionalidad se venga abajo. Este trabajo ayudó a que tomáramos en cuenta esos detalles y pudiéramos abstraer mucho mejor las tareas que hacen las computadoras.

Si se desea probar la computadora creada, se puede acceder a ella en
`https://github.com/mariobecerra/ComputerArchitecturePP1`.

Referencias

[1] Carl Burch. «Logisim: A Graphical System for Logic Circuit Design and Simulation». En: *J. Educ. Resour. Comput.* 2.1 (mar. de 2002), págs. 5-16. ISSN: 1531-4278. DOI: 10.1145/545197.545199. URL: http://doi.acm.org/10.1145/545197.545199.

[2] John L. Hennessy y David A. Patterson. *Computer Architecture, Fourth Edition: A Quantitative Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006. ISBN: 0123704901.

[3] William Stallings. *Computer Organization and Architecture: Designing for Performance (7th Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2005. ISBN: 0131856448.