DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
ESCUELA DE INGENIERÍA
PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

IIC2343 - Arquitectura de Computadores (II/2025)

Etapa 1 del Proyecto

Desarrollo y comunicación

Durante el desarrollo del proyecto, se les asignará a su grupo un repositorio en GitHub donde deberán subir cada una de las etapas que van realizando. Las dudas **generales** del proyecto deberán ser realizadas mediante *issues* en el repositorio del curso, mientras que para las dudas más específicas, como temas de código, deberán agendar una pequeña reunión con alguno de los ayudantes de laboratorio o crear *issues* en sus repositorios privados. Adicionalmente, podrán resolver las dudas que tengan durante las instancias de laboratorio antes de cada entrega.

Para cualquier otro caso, **deberán** contactar a la coordinadora de proyecto: Javiera Pinto (`jpints@uc.cl`).

Descripción

Su trabajo en esta primera etapa será exclusivamente dentro de la CPU.

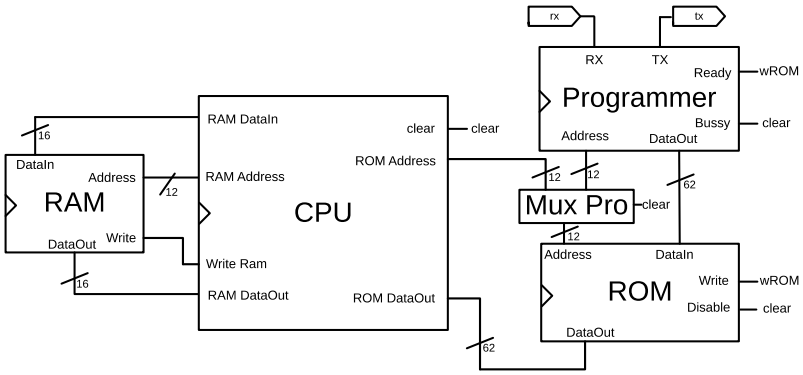

Figura 1: Diagrama parcial del computador básico de proyecto dentro del componente Basys3.

Comenzarán el proyecto implementando las características que les permitan hacer operaciones de carga, aritmético-lógicas y saltos en su computador. Para esto, tendrán que implementar la arquitectura descrita en el siguiente diagrama. Pueden importar, usar y/o crear los componentes que sean necesarios, siempre y cuando estén implementados **a través de lógica combinacional**. Cualquier componente secuencial será entregado en el proyecto base en sus repositorios, los cuales **no deben modificar**.

El desafío de esta etapa es el diseño de distintos componentes dentro del archivo `CPU.vhd`, entre los que se encuentran: una unidad de control; el registro *status*; y un componente que contiene cuatro registros de carga. Este último componente será llamado `Register File` y, a partir de él, se implementará otro llamado `Super Register File`, que contiene adicionalmente el registro contador `PC` o *Program Counter*. Sumado a lo anterior, la computadora tendrá un lenguaje de máquina que se almacenará en las palabras de 62 *bits* de la `ROM`. `CPU.vhd` debe ser conectado al componente `Basys3`, el que tendrá las conexiones necesarias para esta entrega. Por lo tanto, **el archivo `Basys3.vhd` no debe ser modificado**.

Adicionalmente, se le proveerá un ensamblador para que pueda probar código Assembly en su máquina programable. A partir de esta herramienta se evaluará la correctitud de su implementación.

Hardware

A continuación, se incluyen los diagramas de las composiciones internas de los componentes `CPU` y `Register File`. Es importante destacar en ellos la señal `clear`. Esta proviene del componente `Programmer` y permite reiniciar el estado de los registros después de re-programar la `ROM`, por lo que es importante que implementen su conexión para obtener el resultado esperado.

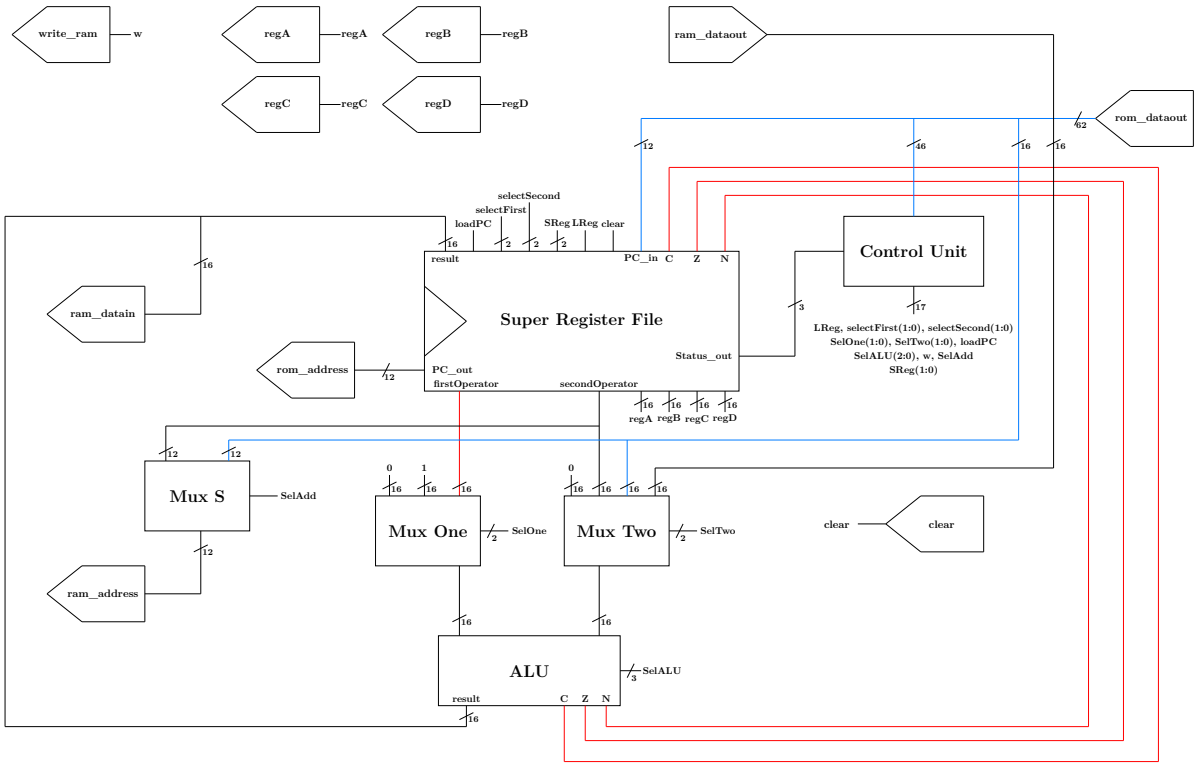

Figura 2: Diagrama del componente `CPU`.

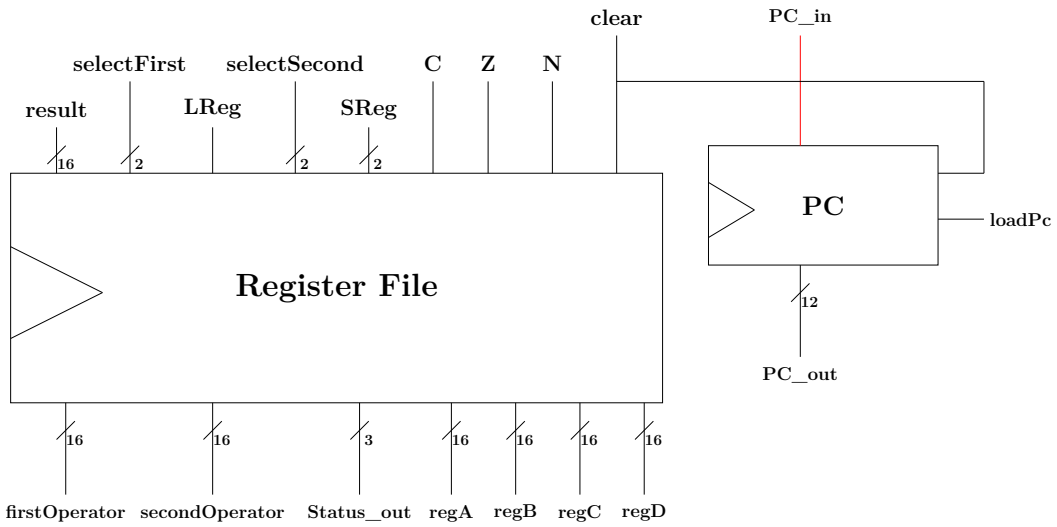

Figura 3: Diagrama interno del componente `Super Register File`.

Por otra parte, la composición **interna** del `Register File` es la siguiente:

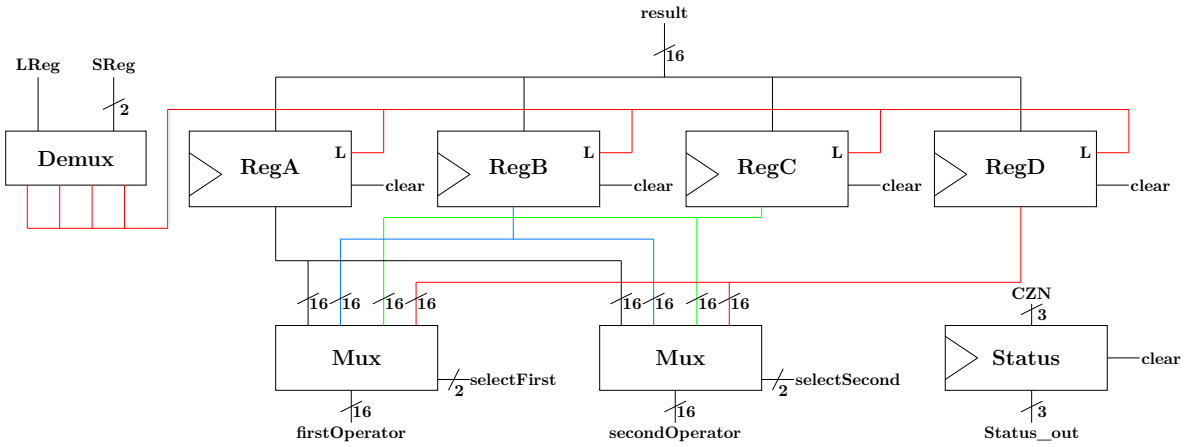

Figura 4: Diagrama interno de las conexiones del `Register File`.

Donde sus selectores y Demux tendrán el comportamiento descrito en la siguiente tabla:

Selector/Demux	Selección	Tipo
00	`RegA`	Salida
01	`RegB`	Salida
10	`RegC`	Salida
11	`RegD`	Salida

A diferencia del computador básico visto en clase, tanto los registros `A`, `B`, `C`, `D` contenidos en el `Register File` como la `ALU` son de 16 *bits*, mientras que el registro `Status` es de 3 *bits* y el contador `PC` es de 12 *bits*. Además, la `ALU` no tiene soporte para operar sobre números en complemento de 2, es decir, tanto los operandos como el resultado se interpretan como números positivos en todo momento.

Registros

Para esta entrega, deberán construir sus registros a partir de un **flip-flop tipo D** que se les entregará en el proyecto base. **Este es el único componente que puede llevar *process* (sin considerar las simulaciones).** En la figura 5, se puede ver cómo implementar un registro de 4 bits, conectando 4 flip-flops tipo D con las señales de entrada; mientras que en la figura 6 se muestra cómo implementar las señales *load* y *reset* en un registro de 1 bit. Es su tarea lograr utilizar los diagramas de referencia para construir un registro de 16 bits que contenga las señales *load* y *reset* con el comportamiento esperado.

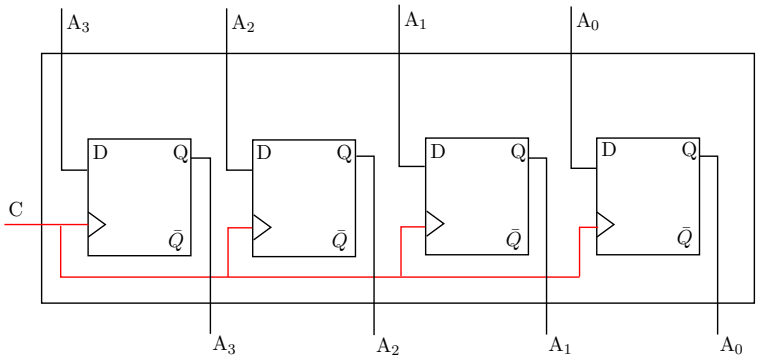

Figura 5: Implementación de un Registro de 4 bits usando 4 *flip-flop* D.

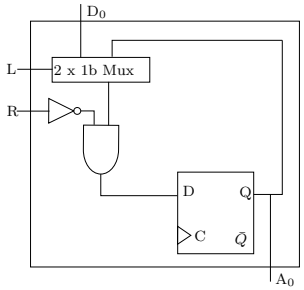

Figura 6: Implementación de las señales *load* y *reset* para un registro de 1 bit.

Program Counter (PC)

El *Program Counter* (`PC`) es un registro especial que funciona como un contador. Su comportamiento es el siguiente: en cada flanco de subida del reloj, el `PC` entrega el valor previamente almacenado incrementado en uno. Inicialmente comienza en cero, luego del primer flanco de subida pasa a uno, en el siguiente a dos, y así sucesivamente.

Además, el `PC` incorpora una señal de *clear*, la cual permite forzar el valor del contador a cero de manera inmediata independiente del valor almacenado en ese instante. De esta forma, el `PC` asegura que la secuencia de direcciones de instrucción pueda reiniciarse cuando el sistema lo requiera.

El comportamiento del `PC` de 12 *bits* se resume en la siguiente tabla:

Entradas		Salida
clock	*clear*	*dataout*(11:0)
Flanco Subida	1	”0000000000000000”
Flanco Subida	0	*dataout* + 1
*	0	*dataout*

Figura 7: Tabla de valores del Program Counter (PC).

Componentes

Su proyecto **debe** tener lo siguiente dentro del `Register File`:

- Cuatro **registros** `Reg A`, `Reg B`, `Reg C` y `Reg D` de uso general, de 16 *bits* cada uno.
- Dos **multiplexores**, cada uno de cuatro entradas de 16 *bits*.
- Un **demultiplexor**, de un bit con cuatro salidas.
- Un **registro** `Status` para las *flags* `C`, `Z` y `N`, donde `C` es el bit más significativo y `N` el menos significativo.

Dentro del componente `Super Register File`:

- Un **conjunto de registros** `Register File` implementado con lógica combinacional.
- Un **registro contador** `PC`, que va incrementando su valor en una unidad por ciclo.

Dentro del componente `CPU`:

- Un conjunto de registros `Super Register File` implementado con lógica combinacional.
- Una **unidad de control** `Control Unit` implementada con lógica combinacional.
- Una **unidad aritmético lógica** `ALU` con 8 operaciones de 16 *bits*.
- Dos **multiplexores** `Mux One` y `Mux Two`, uno de tres entradas y otro de cuatro entradas, ambos de 16 *bits*.
- Un **multiplexor** `Mux S` para la dirección de la `RAM` que seleccione una de dos entradas, cada una de 12 *bits*.

** ***Hint***: No es necesario crear diferentes componentes para cada una de ellas. Recuerden que en Vivado pueden crear distintas **instancias** de un componente para los elementos que tienen el mismo comportamiento. **

Software

Se utilizará un ensamblador para traducir y transferir distintos programas en lenguaje de *Assembly* a la `ROM` de su computador y, de ese modo, poder ejecutarlos. Escriba los programas que estime convenientes para probar su arquitectura. Al final del enunciado, se encuentran algunos ejemplos. No está de más mencionar que estos son, como se señaló, **ejemplos** y no son *tests*. Debido a esto, el funcionamiento

correcto de estos ejemplos no asegura que su computador sea 100 % funcional.

En la siguiente tabla, se describe cómo las palabras en la ROM son codificadas para representar las instrucciones. El ensamblador entregado se encargará de leer las instrucciones en *Assembly*, codificarlas y enviarlas al computador programable. En caso de que ninguno de los *bits* de la palabra de la ROM esté activo para la selección de un multiplexor, estos deben seleccionar una de sus señales de entrada de forma arbitraria a través de `with others` en Vivado. A continuación, se comparten algunos ejemplos:

- 00000000000000000000000000000000000010010100001001001000000001
  Guarda en el registro `C` el resultado de la operación `C + B`. En *Assembly*: `ADD C,B`.
- 00000000000101110000000000000000010010010000000000100000001
  Guarda el literal 23 en el registro `D`. En *Assembly*: `MOV D,23`.
- 00000000000011110100100000000000000000000000000000000000000000
  Salta a la dirección 0x0F asociada a `label` si se cumple la condición de un salto por ser Menor. En *Assembly*: `JLT label`.

Bits	Acción
0	1 cuando la operación de la ALU es Sumar
1	1 cuando la operación de la ALU es Restar
2	1 cuando la operación de la ALU es `AND`
3	1 cuando la operación de la ALU es `OR`
4	1 cuando la operación de la ALU es `XOR`
5	1 cuando la operación de la ALU es `NOT`
6	1 cuando la operación de la ALU es *shift right*
7	1 cuando la operación de la ALU es *shift left*
8	1 cuando se carga un registro
9	1 cuando el primer operando es `A`
10	1 cuando el primer operando es `B`
11	1 cuando el primer operando es `C`
12	1 cuando el primer operando es `D`
13	1 cuando el segundo operando es `A`
14	1 cuando el segundo operando es `B`
15	1 cuando el segundo operando es `C`
16	1 cuando el segundo operando es `D`
17	1 cuando se carga el registro `A`
18	1 cuando se carga el registro `B`
19	1 cuando se carga el registro `C`
20	1 cuando se carga el registro `D`
21	1 cuando en el `MuxOne` se selecciona el primer operando de `Super Register File`
22	1 cuando en el `MuxOne` se selecciona el Uno
23	1 cuando en el `MuxOne` se selecciona el Cero
24	1 cuando en el `MuxTwo` se selecciona el segundo operando de `Super Register File`
25	1 cuando en el `MuxTwo` se selecciona el Cero
26	1 cuando en el `MuxTwo` se selecciona el Literal
27	1 cuando en el `MuxTwo` se selecciona la salida de la RAM
28	**Se implemetará en la siguiente etapa, por ahora siempre es 0**
29	**Se implemetará en la siguiente etapa, por ahora siempre es 0**
30	**Se implemetará en la siguiente etapa, por ahora siempre es 0**
31	1 cuando en el `MuxS` se selecciona el Literal
32	1 cuando en el `MuxS` se selecciona el segundo operando de `Super Register File`
33	**Se implemetará en la siguiente etapa, por ahora siempre es 0**
34	**Se implemetará en la siguiente etapa, por ahora siempre es 0**
35	**Se implemetará en la siguiente etapa, por ahora siempre es 0**
36	1 cuando hay un salto incondicional
37	1 cuando hay un salto por Igualdad
38	1 cuando hay un salto por Desigualdad
39	1 cuando hay un salto por ser Mayor
40	1 cuando hay un salto por ser Mayor o Igual
41	1 cuando hay un salto por ser Menor
42	1 cuando hay un salto por ser Menor o Igual
43	1 cuando hay un salto por un Acarreo
44	**Se implemetará en la siguiente etapa, por ahora siempre es 0**
45	**Se implemetará en la siguiente etapa, por ahora siempre es 0**
61 al 46	Valor del Literal de 16 bits

Tabla 1: Tabla de palabras de control.

Conectividad con la placa

Para esta entrega, el archivo `Basys3.vhd` cuenta con todas las conexiones con la `CPU` ya establecidas para la interacción con la FPGA. Lo que muestra el *display* de la placa dependerá de un segmento de la señal `sw`, cuyo comportamiento se describe en la siguiente tabla:

`sw(15:12)`	Selección	Tipo
000	`regA`	Salida
001	`regB`	Salida
010	`regC`	Salida
011	`regD`	Salida
100	`ram_address`	Salida
101	`rom_dataout`	Salida
110	`ram_datain`	Salida
111	`rom_address`	Salida

Por otro lado, la velocidad del reloj será dada por el componente `ClockDivider`. Este ya se encuentra definido en el proyecto, y su velocidad será manipulada por otro segmento de la señal `sw`, cuyo comportamiento se describe a continuación.

`sw(1:0)`	Selección	Tipo
00	`Instantáneo - 25 MHz`	Entrada
01	`Rápido - 8 MHz`	Entrada
10	`Normal - 2 MHz`	Entrada
11	`Lento - 0.5 MHz`	Entrada

Todos los componentes se conectarán a la misma señal *clock* proveniente del componente `ClockDivider`, asegurando sincronía entre los registros y la memoria de datos en cada flanco de subida.

Finalmente, **no se utilizarán los botones de la placa en esta entrega**.

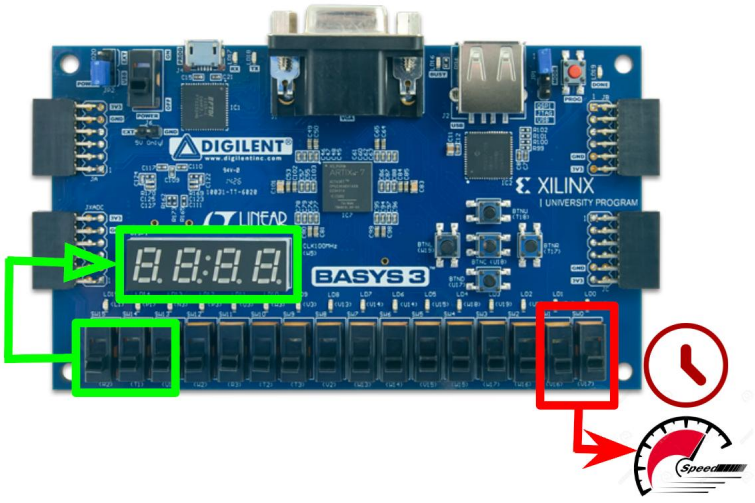

Figura 8: Diagrama interno de la placa indicando los interruptores descritos en las tablas.

Presentación

Se aceptarán envíos de *commits* hasta **el martes 23 de septiembre a las 13:30 horas**, momento en que se recolectarán todos los repositorios. Deberán presentar sus proyectos **en los computadores del laboratorio en no más de 10 minutos** en el día y horario correspondiente a su sección. Esto se hará con una descarga del comprimido de su repositorio, que el equipo docente proveerá al momento de evaluar. El día de la presentación deberán mostrar **presencialmente** su proyecto y correr los *test* proporcionados por el equipo docente.

A la hora de la entrega, su repositorio **deberá contener todos los archivos necesarios para correr su proyecto**.

Puntaje

La nota será calculada utilizando 4 *tests*. Si logra pasar el primer *test*, se considerará la entrega aprobada en su totalidad. Si no se logra pasar el primer *test* por completo, se probarán los siguientes tres *test*, en donde se asignará puntaje acorde a la siguiente distribución:

- ***Test* 1**: Se prueba lectura y escritura en memoria; todas las operaciones de la `ALU`; saltos incondicionales y condicionales. (36 puntos)
- ***Test* 2**: Se prueban todas las operaciones de la `ALU`. (12 puntos).
- ***Test* 3**: Se prueban todas las instrucciones que interactuaren con la memoria. (12 puntos).
- ***Test* 4**: Se prueban saltos incondicionales y condicionales. (12 puntos)

Cálculo de nota entrega:

$$\text{mín}\left(round\left(\frac{\text{Puntaje Obtenido}}{6}+1,\ 2\right),\ 7\right)$$

Es de suma importancia mencionar que cualquier proyecto en el que se use *process* en algún componente que no lo incluya originalmente, **obtendrá el puntaje mínimo en la entrega.** En otras palabras, queda totalmente prohibido su uso.

Recuperación de puntaje y requisitos

Durante las semanas previas a la presentación de la entrega, se realizarán salas de ayuda en el horario de laboratorio para que trabajen en el proyecto y resuelvan dudas con sus ayudantes. Si su grupo termina con un 100 % de asistencia en estas instancias, se le permitirá **recuperar hasta la mitad del puntaje descontado en la evaluación**. Para que la asistencia sea considerada, **al menos una persona** del grupo debe estar presente **durante los dos bloques de laboratorio** que correspondan a su sección. **Solo se aceptará asistencia al laboratorio de su sección**, es decir, cualquier persona que intente asistir a un laboratorio que no le corresponde no será admitida y su asistencia no será registrada.

La recuperación del puntaje, en caso de cumplir con los criterios de asistencia, se realizará durante la próxima sala de ayuda que corresponda a su sección (semana del 29 de septiembre). En esta, debe mostrar que arregló **todos** los errores de su entrega original. El código de la recuperación se presentará el día de su laboratorio correspondiente, donde deberá incluir los arreglos de esta entrega, pero **nada posterior a ella**.

Assembly

MOV	R1,R2	Guarda el valor de R2 en R1
	R1,Lit	Guarda un literal Lit en R1
	R1,(Dir)	Guarda el valor de Mem[Dir] en R1
	(Dir),R1	Guarda el valor de R1 en Mem[Dir]
	R1,(R2)	Guarda el valor de Mem[R1] en R2
	(R2),R1	Guarda el valor de R1 en Mem[R2]
	(R2),Lit	Guarda un literal Lit en Mem[R2]
ADD SUB AND OR XOR	R1,R2	Guarda el resultado de R1 op R2 en R1
	R1,Lit	Guarda el resultado de R1 op Lit en R1
	R1,(Dir)	Guarda el resultado de R1 op Mem[Dir] en R1
	R1,(R2)	Guarda el resultado de R1 op Mem[R2] en R1
NOT SHL SHR	R1	Guarda el resultado de op R1 en R1
	(Dir),R1	Guarda el resultado de op R1 en Mem[Dir]
	(R2),R1	Guarda el resultado de op R1 en Mem[R2]
INC	R1	Incrementa el valor de R1 en una unidad
	(Dir)	Incrementa el valor de Mem[Dir] en una unidad
	(R1)	Incrementa el valor de Mem[R1] en una unidad
DEC	R1	Decrementa el valor de R1 en una unidad
CMP	R1,R2	Ejecuta la instrucción SUB R1,R1 sin actualizar el valor en ningún registro
	R1,Lit	Ejecuta la instrucción SUB R1,Lit sin actualizar el valor en ningún registro
	R1,(Dir)	Ejecuta la instrucción SUB R1,(Dir) sin actualizar el valor en ningún registro
	R1,(R2)	Ejecuta la instrucción SUB R1,(R2) sin actualizar el valor en ningún registro
JMP	Ins	Carga la dirección de la instrucción Ins en PC
JEQ	Ins	Carga la dirección de la instrucción Ins en PC si en Status se cumple Z = 1
JNE	Ins	Carga la dirección de la instrucción Ins en PC si en Status se cumple Z = 0
JGT	Ins	Carga la dirección de la instrucción Ins en PC si en Status se cumple N = 0 y Z = 0
JGE	Ins	Carga la dirección de la instrucción Ins en PC si en Status se cumple N = 0
JLT	Ins	Carga la dirección de la instrucción Ins en PC si en Status se cumple N = 1
JLE	Ins	Carga la dirección de la instrucción Ins en PC si en Status se cumple N = 1 o Z = 1
JCR	Ins	Carga la dirección de la instrucción Ins en PC si en Status se cumple C = 1
NOP		No hace cambios