**PONTIFICIA UNIVERSIDAD JAVERIANA CALI**

**PROYECTO SISTEMA DE CÓMPUTO**

**INGENIERIA DE SISTEMAS Y COMPUTACION**

**FACULTAD DE INGENIERIA**

**MARIBELL SACANAMBOY**

**ARQUITECTURA DE COMPUTADOR II**

**ALEJANDRO MEZA BARRERA**

**GERMÁN ANDRÉS CAYCEDO MUTIS**

**DAVID HERNÁNDEZ CÁRDENAS**

**MARÍA CAMILA LÓPEZ LEAL**

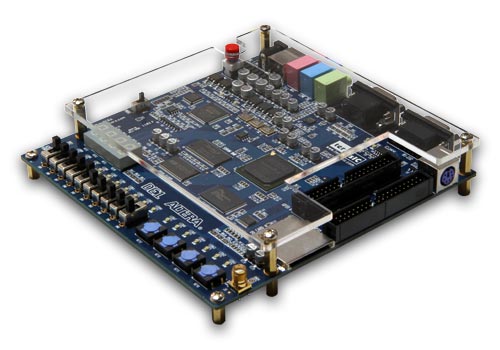
**SANTIAGO DE CALI, NOVIEMBRE DE 2018**

**APLICACIÓN**

Se desarrollará un sistema de cómputo donde se correrá una aplicación enfocada en el manejo, organización y conocimiento del periodo menstrual de las mujeres. Para ello se hará uso de información personal acerca de cada mujer, como la duración de su ciclo, la última vez de su periodo, la duración de días con periodo, entre otros. Además, la aplicación logrará alertar por medio de datos como la planificación y el hecho de estar en el periodo o no, cuándo se tienen señales de embarazo, retrasos anormales o en su defecto si su ciclo se encuentra correcto sin ningún tipo de preocupación. Al contrario de las aplicaciones normalmente vistas en las tiendas, que son abordadas para llevar un calendario y cómo con el tiempo se lleva una regulación del periodo menstrual, la nuestra se centrará en generar diferentes resultados inmediatos a partir de información predeterminada de cada mujer según sea su caso.

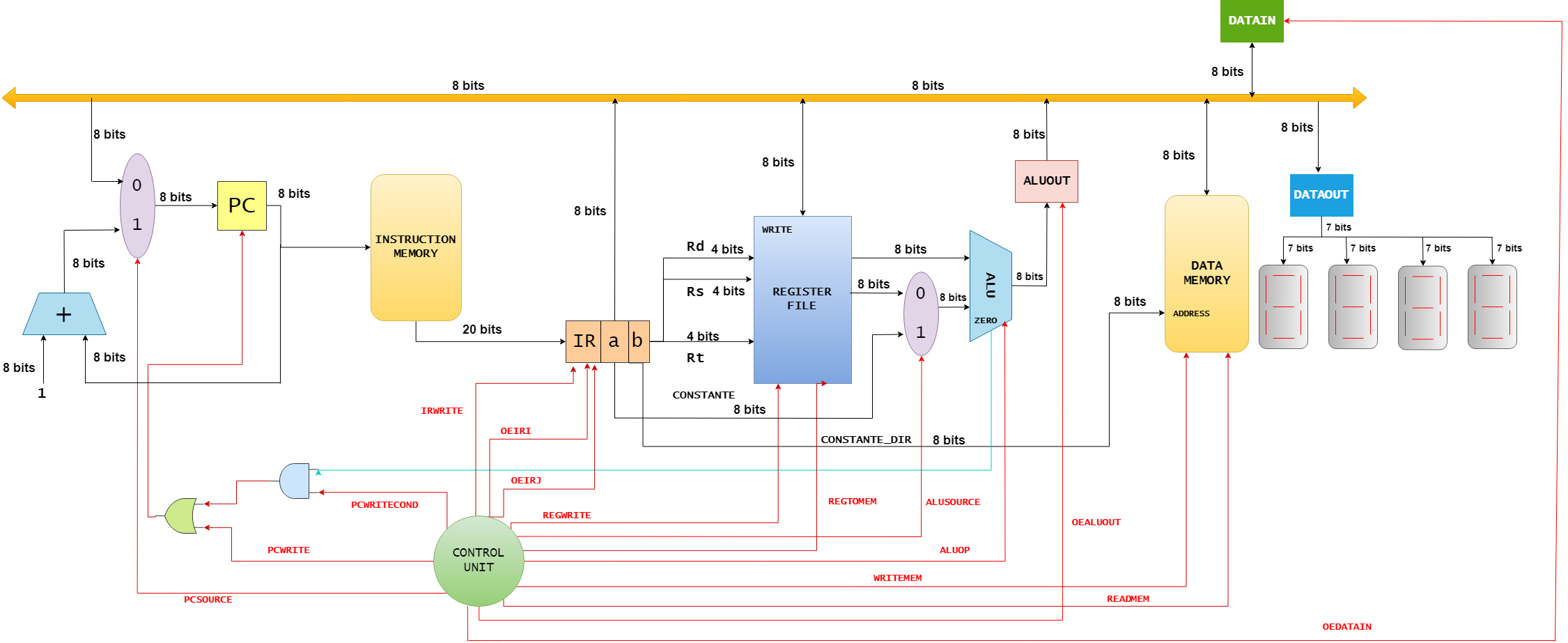
**RECURSOS**

La iniciación del sistema de cómputo se basa en el proceso de creación del algoritmo en alto y bajo nivel el cual será usada en el sistema ya mencionado. Para ello, se usarán aplicaciones de modificación de texto y comprobación del funcionamiento aritmético lógico de ambos algoritmos (bajo y alto nivel). Consumado esto, para la realización y/o el desarrollo del sistema de cómputo se hará uso de la herramienta Quartus II Edición Web para el análisis y el compendio de los diseños a realizar en lenguaje HDL (Lenguaje de descripción de hardware). En este caso, el lenguaje a emplear es VHDL derivado del anterior mencionado. El dispositivo en el cual se trabajarán las entradas, señales y salidas será la FPGA Cyclone II [EP2C20F484C7](https://www.google.com.co/search?q=EP2C20F484C7&oq=EP2C20F484C7&aqs=chrome..69i57j0l5.720j0j7&sourceid=chrome&ie=UTF-8).

****

**DIAGRAMA ESTRUCTURAL DE LA ARQUITECTURA DEL PROCESADOR**

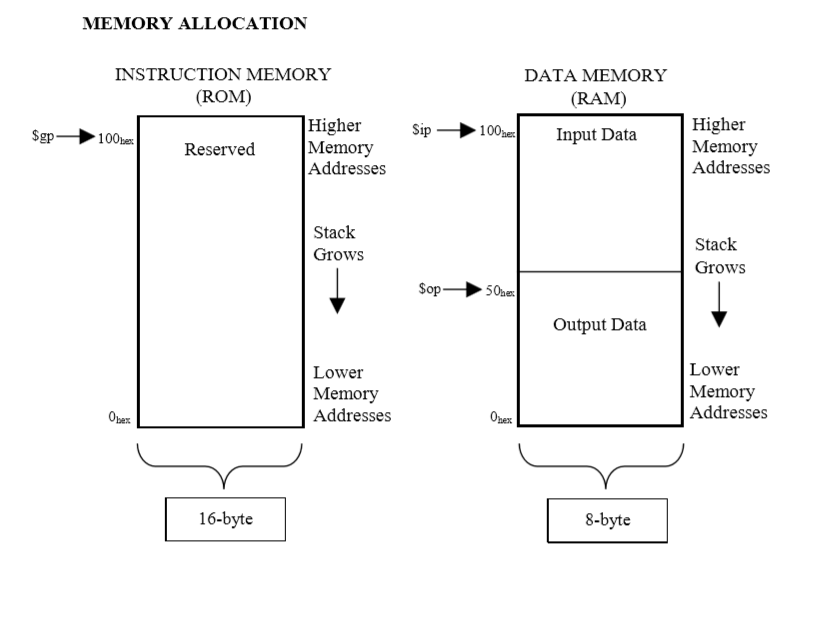
La arquitectura a implementar en el sistema de cómputo se basa en el funcionamiento de la arquitectura multiciclo de MIPS pero con ligeros cambios en el hardware ya que omitieron algunos componentes y buses que consideramos no relevantes a la hora de ejecutar nuestro programa. Por ende, se podría considerar un hibrido con otras arquitecturas antes vistas en el curso donde podremos encontrar nuevo hardware como un bus principal y un sumador de uno en uno encargado de aumentar el Program Counter (PC). Para el manejo de la interfaz con el usuario y las diferentes salidas del programa se tendrá en cuenta 4 displays 7 segmentos ubicados en la tarjeta FPGA en el cual se mostrarán los resultados de las diferentes funciones a realizar.

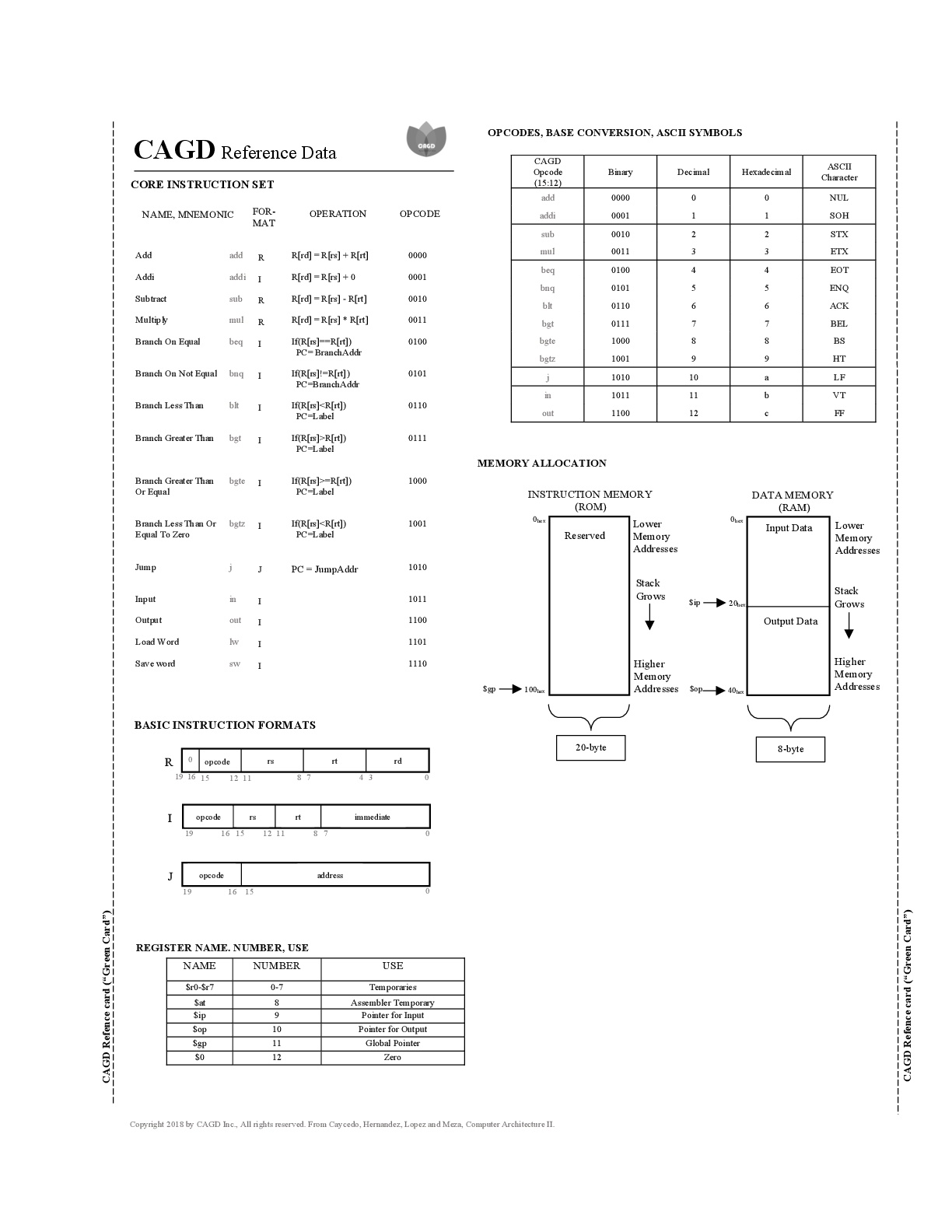
****

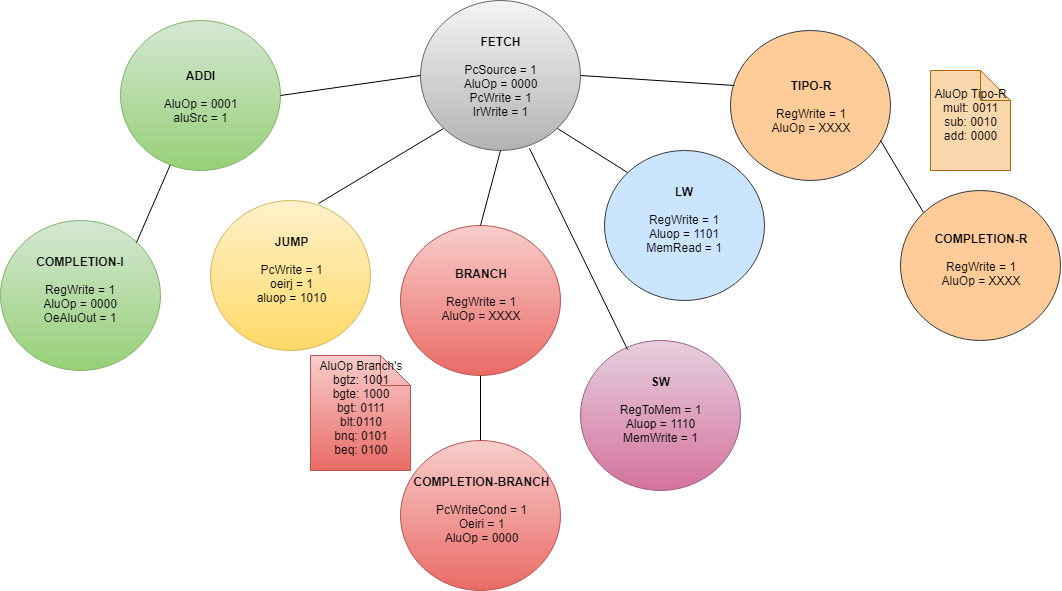
**CRITERIO DE DISEÑO PARA EL MANEJO DE MEMORIA**

Para el manejo de la memoria dentro de la arquitectura se implementarán y/o usarán dos memorias (ROM y RAM). Esto facilitará la manera en que se acceden a las instrucciones y a los datos al momento de ejecutar el programa. Para la ROM se usó un tamaño de 20 bits para cada instrucción y una profundidad de 256 bloques. Debido a que nuestras instrucciones son de tres tipos generales: tipo J, tipo I y tipo R fueron necesario definir el tamaño para cada parte de la instrucción (opcode, registros, constantes, etc) esta información se puede analizar más detalladamente en la ISA aunque por lo general se estableció que el opcode es de tamaño 4 bits, así mismo cada registro (rd, rs, rt) y las constantes de 8 bits dependiendo de la instrucción.

Por otro lado, en la RAM se estableció un ancho de 8 bits puesto que el máximo valor necesario en las constantes requeridas para nuestra aplicación oscila entre 100 y 180 como máximo. Y también se utilizan una cantidad de bloques de 256 porque se prefirió que sobre un poco de espacio a que falte ya que puede causar graves fallas en el programa.





**Máquina de estados**

**Ejecución de instrucciones**

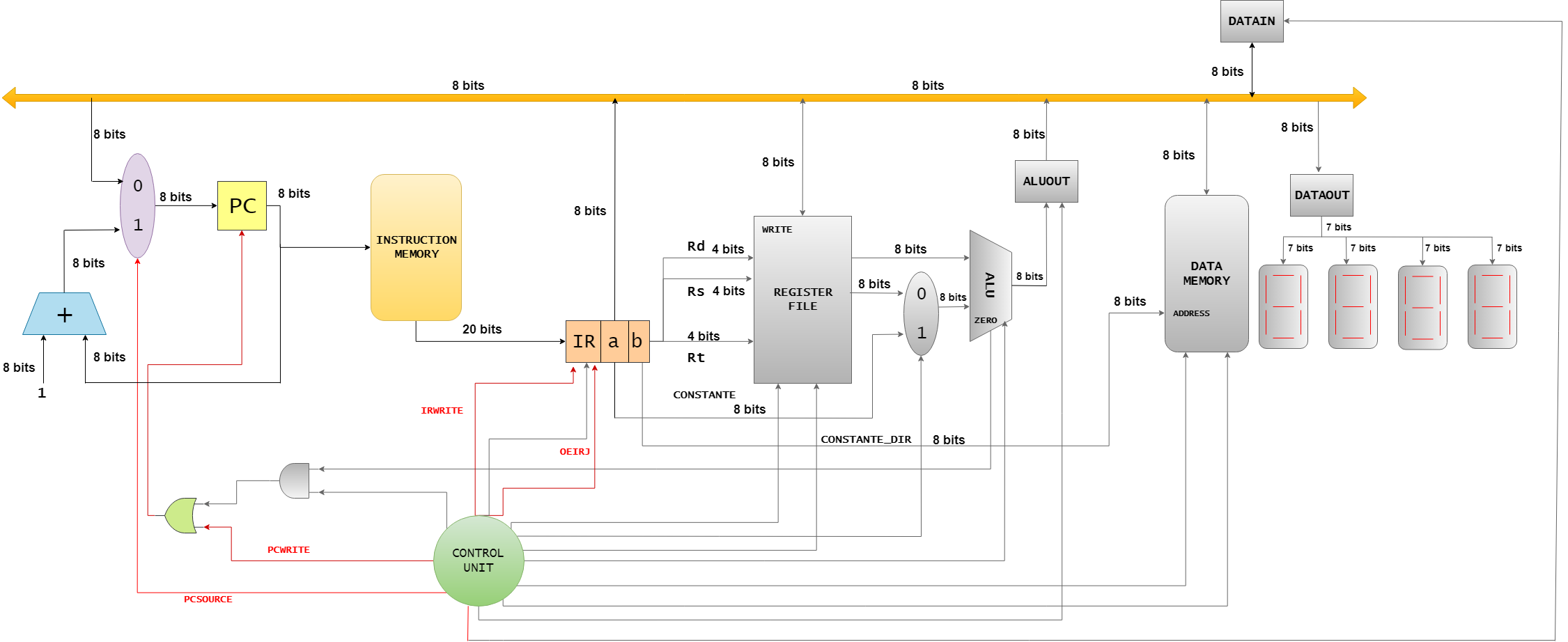
En esta parte del documento se detallará la ejecución de los tipos de instrucciones que se usaran en la aplicación. Elementos importantes como señales, componentes y buses están diferenciados por el color. Además, se especificará lo que sucede en cada etapa ya que es de relevancia para lograr un mayor entendimiento del sistema de cómputo.

**Etapa de Fetch:**

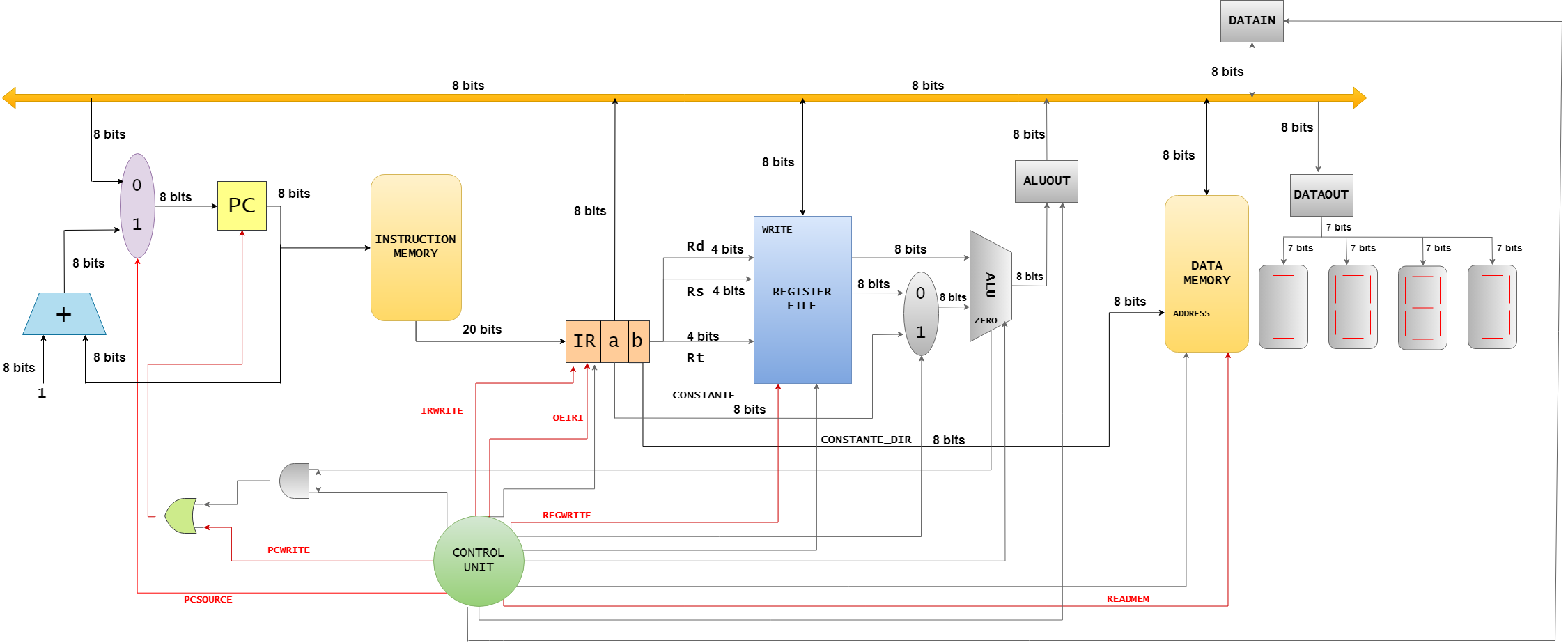
Es la primera etapa que se ejecuta en cada instrucción y es la que actualiza el program counter mediante un adder que suma de una en una unidad, asimismo es donde se transmite cada instrucción de la memoria de instrucciones al registro de instrucciones. Esta etapa es común a todos los tipos de instrucciones.

**Jump:**

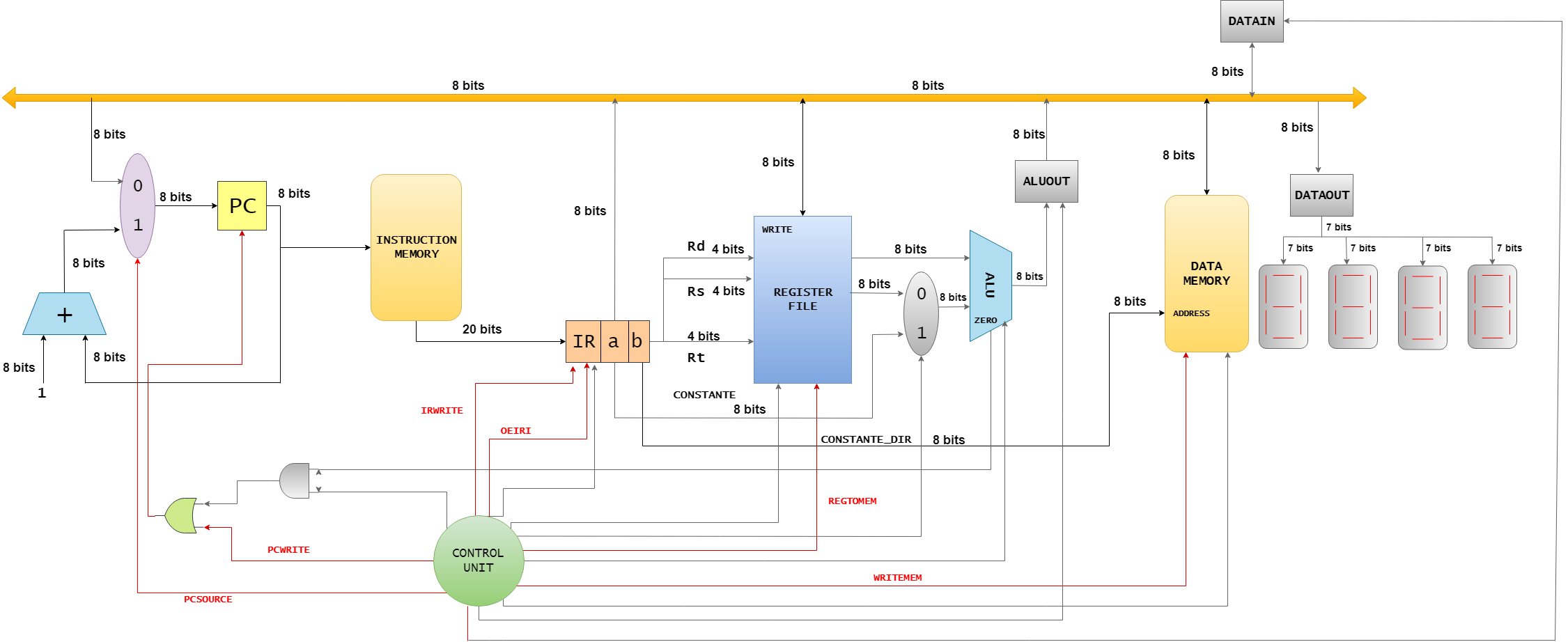
Después de haber realizado la etapa de fetch si la instrucción es de salto no condicional lo que se sucede es que en el IR se encontrará la instrucción en binario por lo que solo se necesitará de mandar los bits que conforman la dirección de donde se desea saltar al PC. Para ello se deben tener habilitadas todas las señales necesarias para un fetch(PcWrite,PcSource,IrWrite) y en la etapa de Jump se debe modificar el pcSource en 0 y activar el OeIrJ.



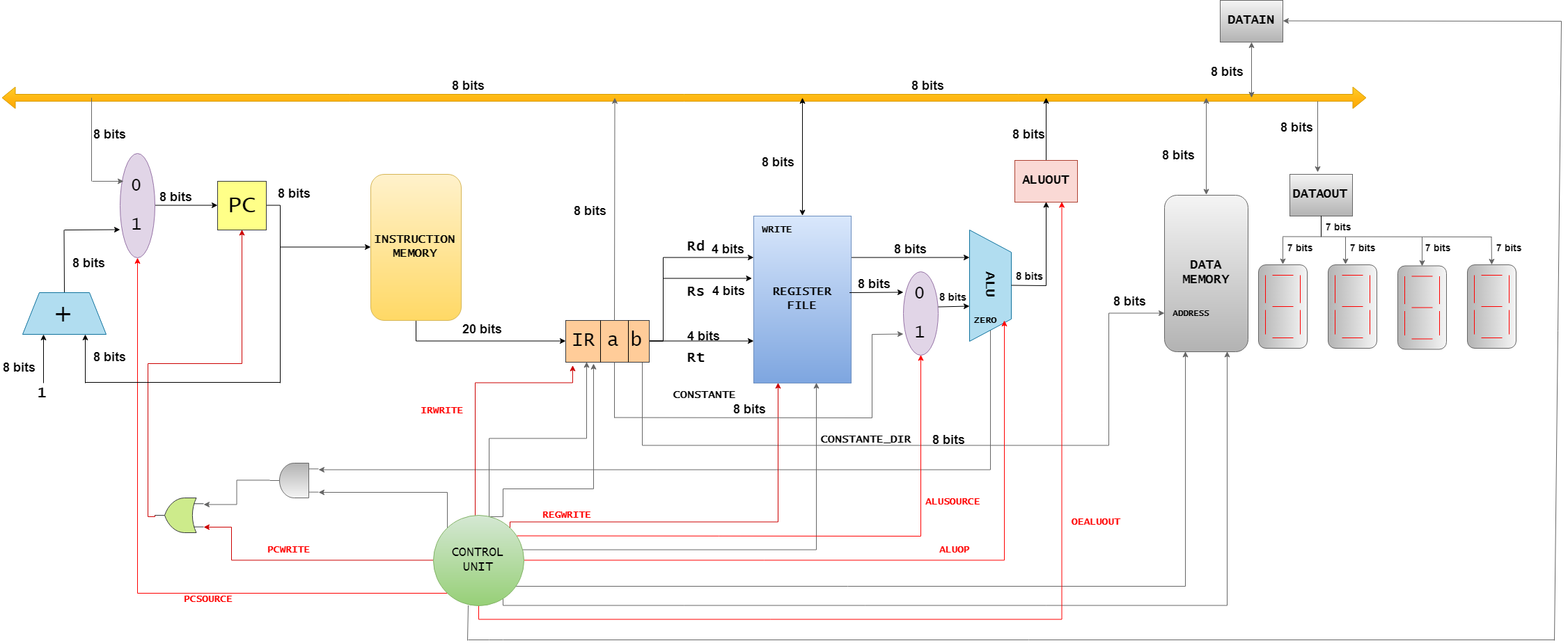
**Load Word:**

Cuando se va a cargar de memoria un dato se tendrán habilitas las señales OeIrI que permite mandar del IR a la memory data(RAM) la dirección en donde se encuentra alojado el dato que se desea cargar, posteriormente con la señal ReadMem se da paso a mandar por el bus principal el dato al register file y con RegWrite se escribirá en el registro destino.

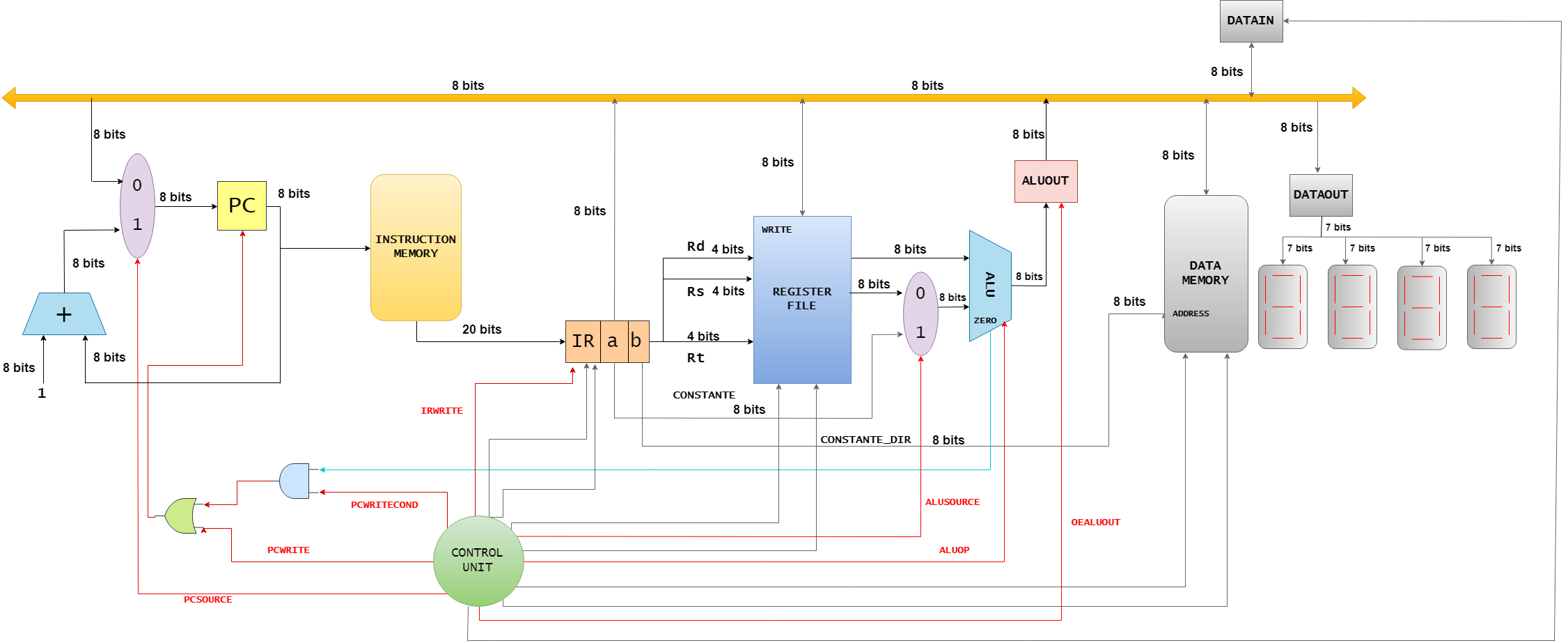
**Store Word:**

En esta etapa se realiza el proceso contrario al lw puesto que se necesita pasar un dato que está en un registro a la memoria, entonces para ello después de la etapa de fetch se manda a través del bus principal el dato desde el register file hasta la memoria, es necesario tener habilitadas las señales RegToMem, WriteMem y OeIrI para tener la dirección donde se va a guardar.

**Tipo R (Incluye Tipo R Y completion-R):**

En esta etapa se hace uso de componentes como el register File, Alu y AluOut ya que es necesario operar los valores dentro de los dos registros rs y rt y guardarlo en rd, teniendo en cuenta eso, la señal Alusource debe estar en 0 y dependiendo del Aluop se le diría a la Alu que operación se debe hacer entre Add,Sub,Mult y el OeAluout debe estar en 1 para mandar el dato al Register File y guardarlo en Rd.

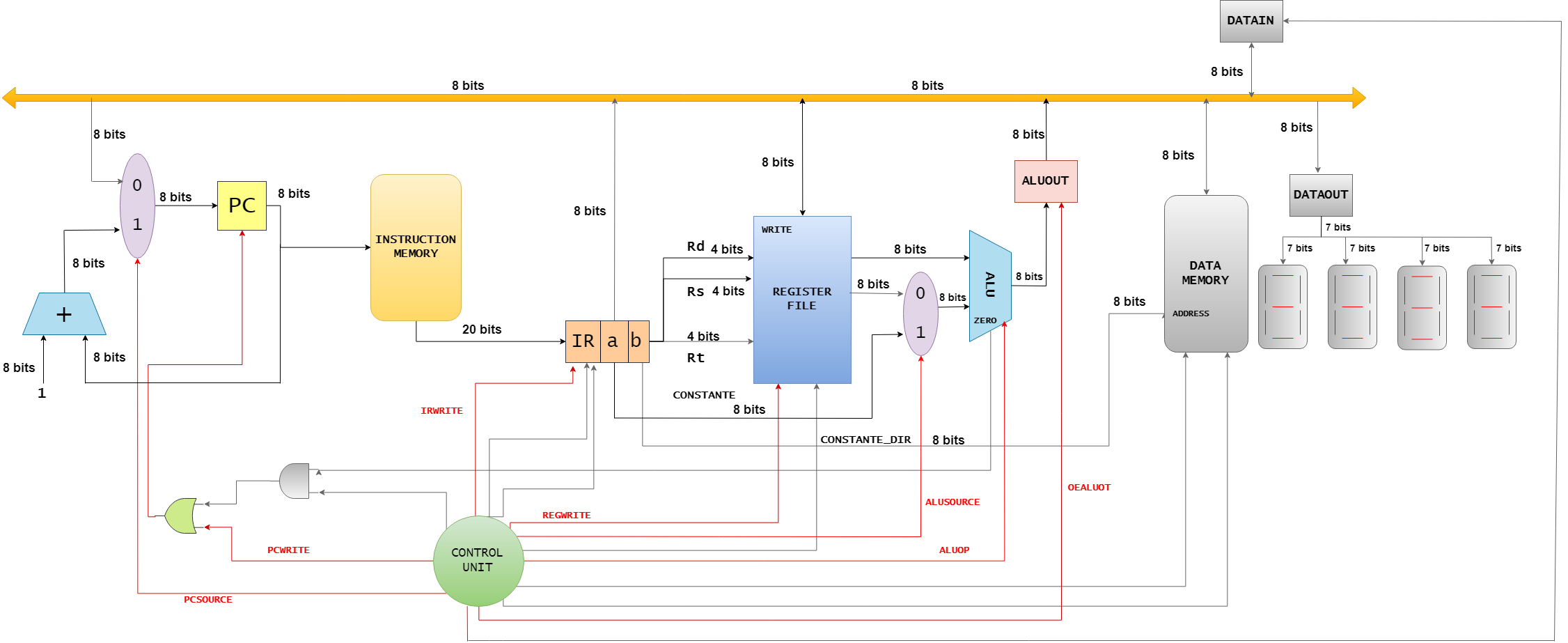
**Tipo Branch (incluye Tipo Branch y Completion Branch):**

En este tipo de instrucción se va tomar dos casos posibles, cuando el brach se cumple se va a usar la señal AluSource en 0 para comparar en la Alu los registros rs y rt, luego se guardará en Aluout terminando el ciclo. Cuando comienza el otro ciclo (Completion Branch) se procede a mandar el dato por el bus príncipal y escribir en el PC la dirección donde se ****va a saltar utilizando el OeAluot, el PcSource en 0 y PcWriteCond habilitado.

En el caso de que la condición del branch no se hubiera cumplido cambiarían las señales de la última etapa de Completion Branch ya que el OeAluot = 0, el PcSource en 1, PcWriteCond = 0 y el PcWrite estaría habilitado

**Addi:**

En la aplicación se hizo uso de una instrucción con manejo de inmediatos o constantes la cual es addi, el procedimiento que tiene que suceder en la arquitectura seria el siguiente, una vez ubicados en el IR se debe colocar el AluSource en 1 para dejar pasar la constante y el registro rs que se sumaran más tarde en el componente Alu y después se guardara en Aluout en donde se acabara el ciclo. Lo que sigue seria mandar el dato con OeAluout al Register File en donde se tiene que activar el RegWrite para almacenar el dato en rd.



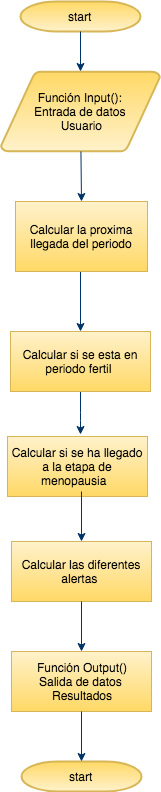
**ALGORITMIA**

El desarrollo de algoritmos en bajo nivel (.asm) dependen en gran parte de la invención de estos en alto nivel, que son los comúnmente realizados en el ámbito laboral y académico. Por lo tanto, la creación del programa a ejecutar por el procesador en lenguaje ensamblador se dio inicio con la creación del programa en lenguaje de alto nivel. En este caso, se hizo uso del lenguaje de programación C. Por lo tanto, la creación del algoritmo en bajo nivel se vio facilitada por el programa ya creado en alto nivel. Durante el proceso de desarrollo se tomó la posibilidad de asignarle una función específica dentro del programa a cada salida de este por medio de las diferentes entradas antes mencionadas (APLICACIÓN).

**DIAGRAMAS DE FLUJO**

Se crearon diagramas de flujo para el programa en alto y bajo nivel (programa a ejecutar por el procesador). Al igual que en los códigos enunciados, los diagramas de flujo también están divididos según salidas específicas.

**Diagrama de flujo aplicación**

****

El anterior diagrama explica superficialmente el orden de ejecución de cada una de las funciones en la aplicación para poder brindar los resultados esperados. Sin embargo, es importante saber cómo se realiza el proceso y la manipulación de la FPGA, por lo cual, a continuación, se explicará un poco sobre estos aspectos.

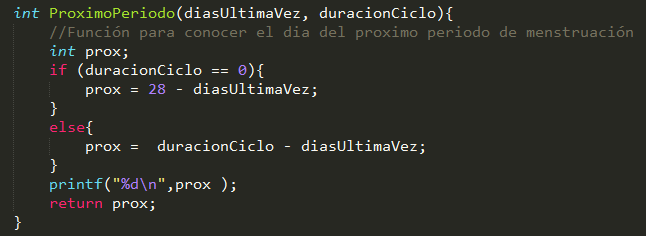
**INPUT():**

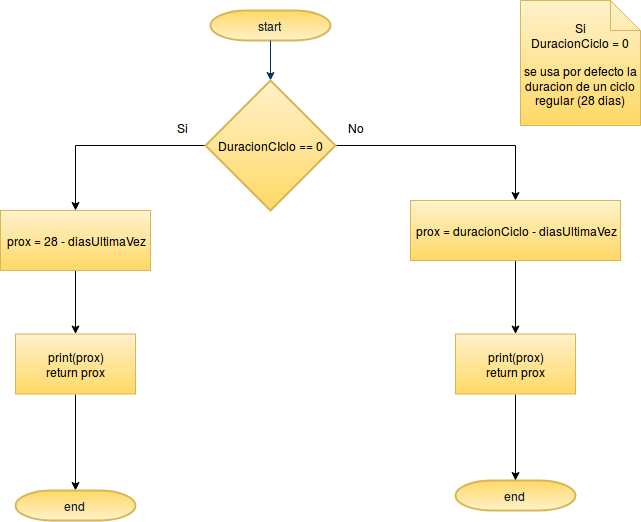
Esta Instrucción junto con Output() fueron creadas con el objetivo de tener una interacción con el usuario, son las encargadas de controlar y manejar la entrada y salida de datos. Si nos ubicamos en la FPGA cada vez que se ingresen los datos que son binarios y son representados con los 10 switches, después de esto se usará un botón especifico que cuando sea oprimido mandara una señal de “enter” que tiene como objetivo capturar el dato, en pocas palabras, lo que se hace es mandar el dato del registro dataIn a la RAM para su posterior uso en la aplicación.

Una vez entendido el proceso que se realiza por debajo en un INPUT() se entiende fácilmente que en la primera fase de la aplicación usa varias veces esta instrucción para alojar en RAM los datos correspondientes de cada mujer tales como la edad, el día en que tuvo su última relación, si planifica o no y todos los datos pertinentes para la realización de cálculos y posteriores resultados.

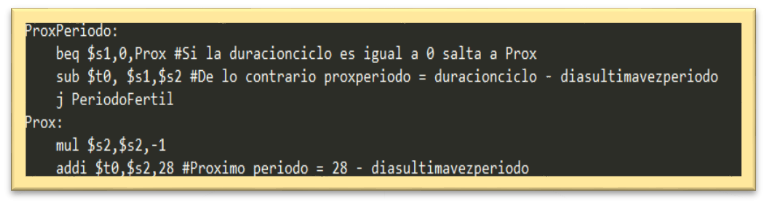
**Calcular la próxima llegada del periodo:**

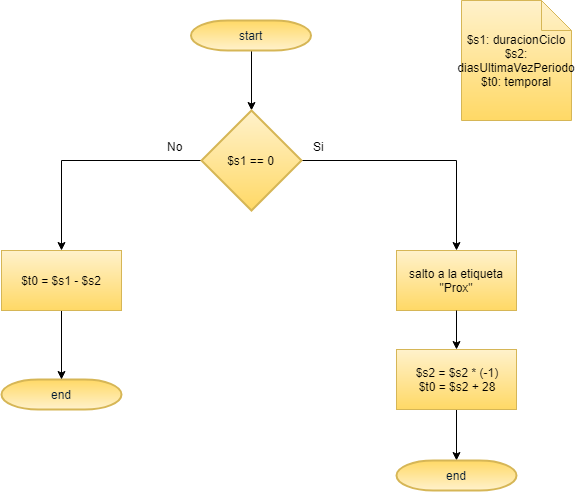
Como su nombre lo indica en esta fase de calcula la próxima llegada del periodo de la mujer, ¿cómo se calcula esto en la aplicación? En el siguiente diagrama de flujo se puede observar que es muy sencillo con los datos de entrada.

**Alto nivel:**

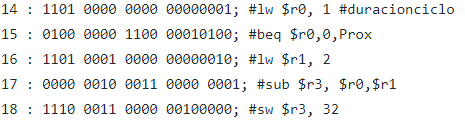
****

**Bajo nivel:**

****



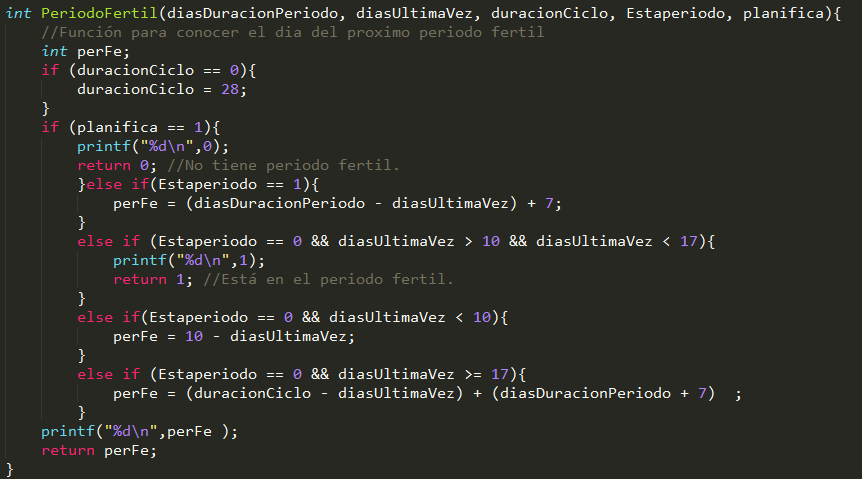
Para poder realizar esto en el sistema de cómputo se utilizó instrucciones para cargar datos (LW) de la RAM a registros como r0 y r1 y realizar una resta que será el resultado que se almacena en alguna posición de la RAM para luego ser visualizado en los 7 segmentos. También se realiza un branch (beq) en el caso de que no se tengan los datos y para ello se realiza otro procedimiento que es asignar un dato por defecto en las variables y calcular ahora si la próxima llega del periodo. Cuando ya estemos ubicados en la FPGA algo importante de aclarar es que es necesario tener las instrucciones en **binario** en la memoria ROM previamente para que el sistema de cómputo sepa que se debe hacer.

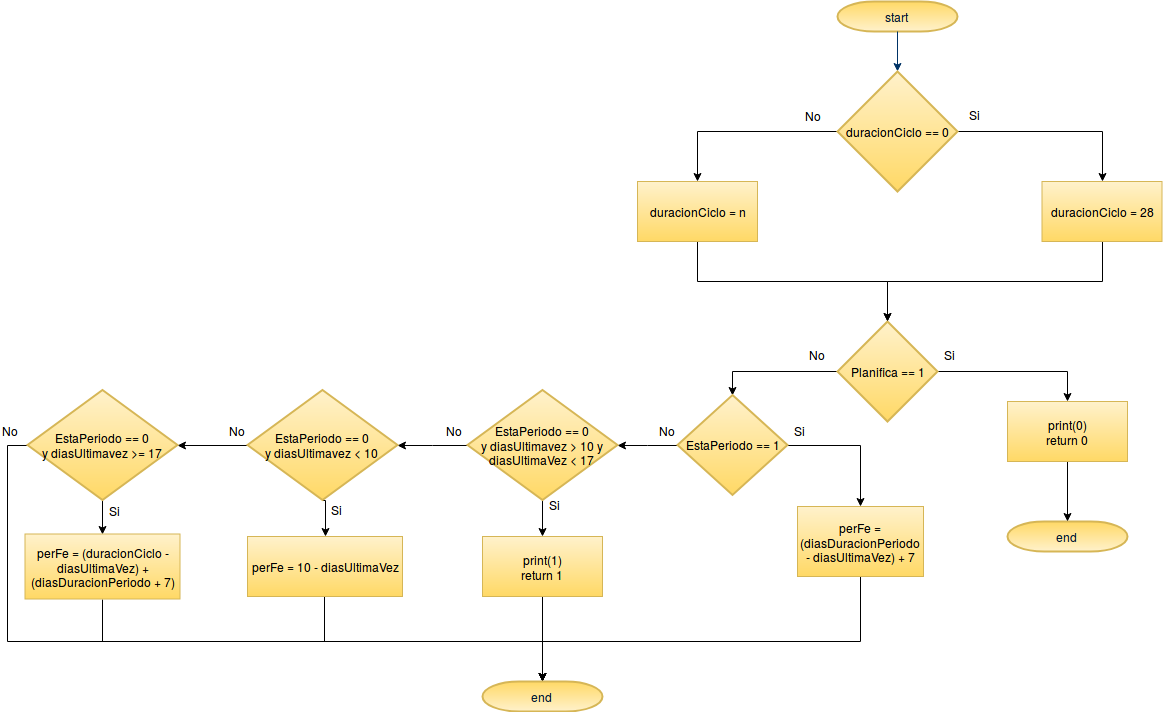


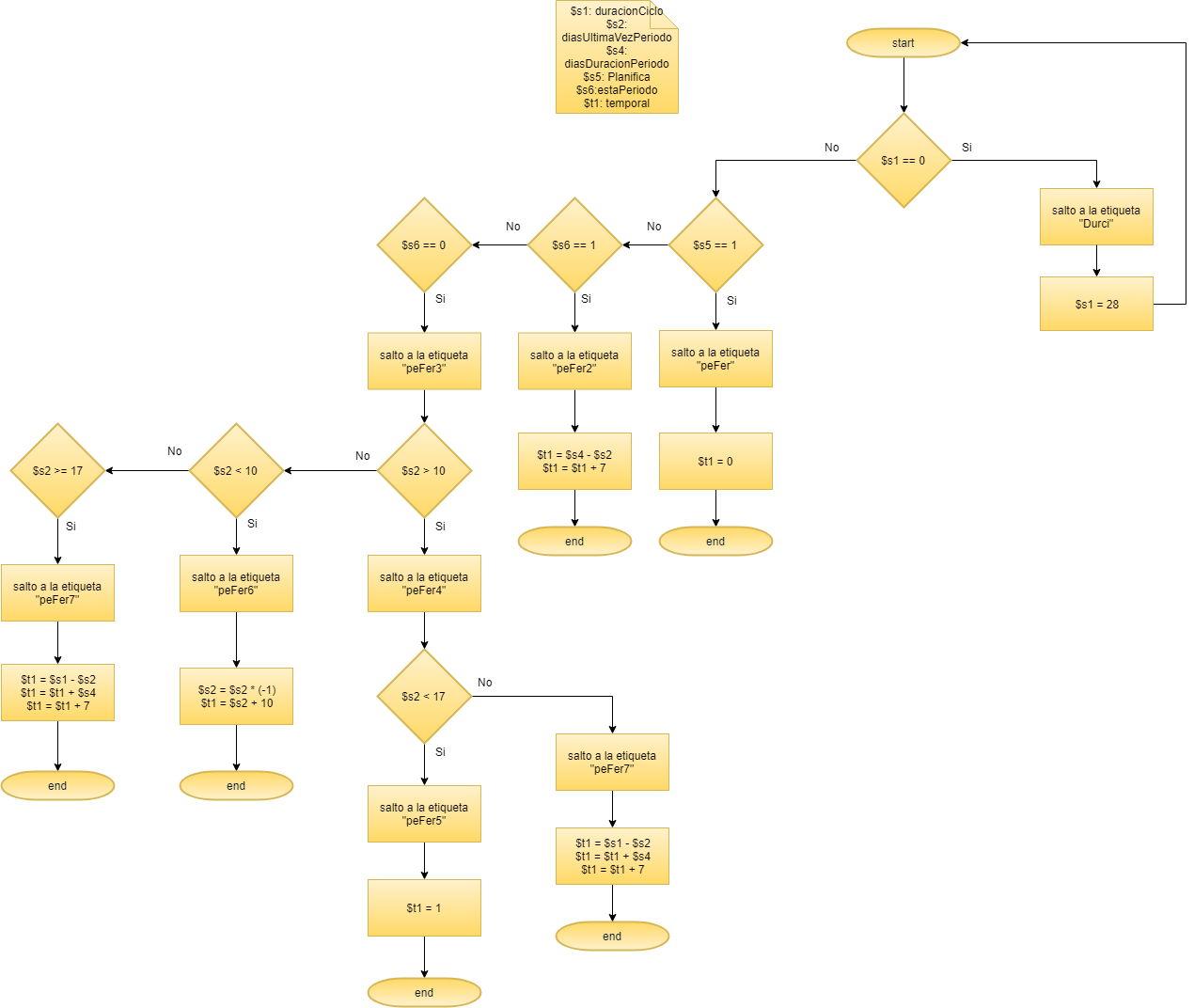
**Calcular si se está en periodo fértil:**

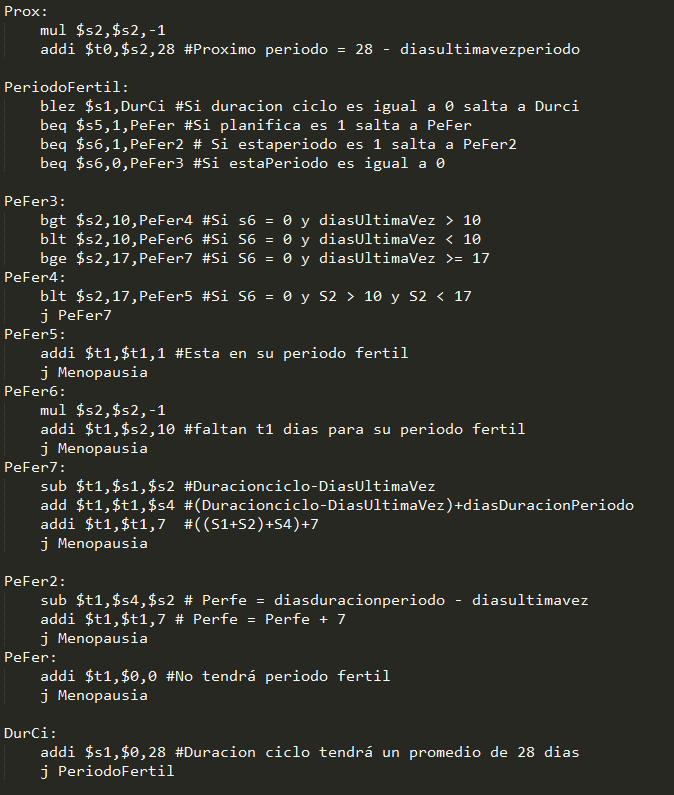
En esta fase del programa se realizan cálculos muchos más complejos sobre comparaciones y se evidencia en el siguiente diagrama de flujo:

**Alto nivel:**

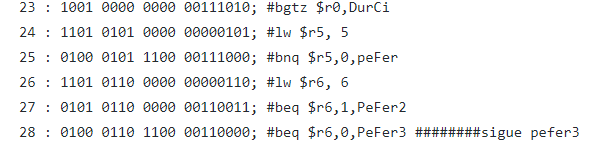
****



**Bajo nivel:**



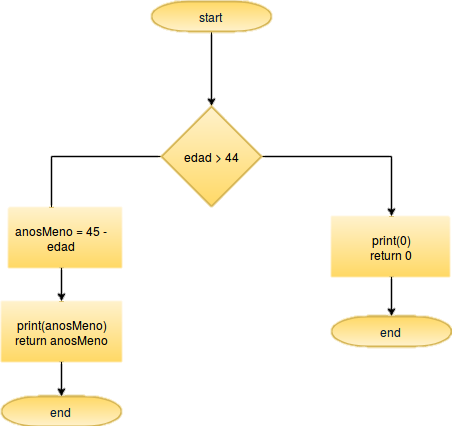
Esta vez no se colocara todas las instrucciones del .mif puesto que son demasiadas y por simplicidad omitimos esa parte, aunque si se desea verificar el .mif completo está ubicado en la carpeta de Rom\_v1 del proyecto VHDL.

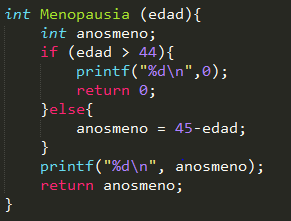


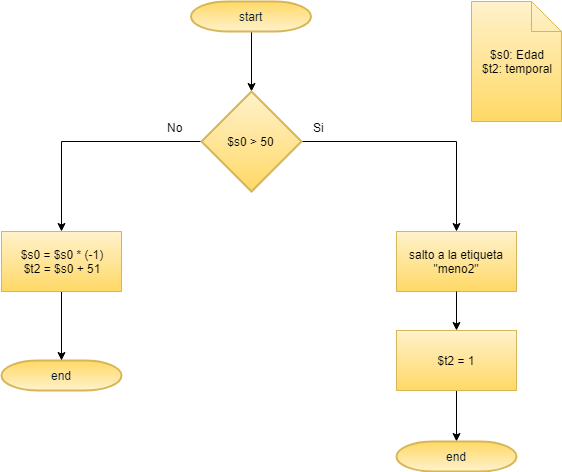
Como se dijo antes aquí se usaron muchos más comparadores como bgtz,bnq,beq, entre otros puesto que es necesario verificar cada una de las variables de entrada como por ejemplo si se está planificando pues se sabría que la mujer no tiene periodo fértil en dicho caso. Como en todas las etapas se hizo uso de lw’s y también de addi’s sub’s y sw’s, se considera esta fase junto con la de Alertas las más complejas.

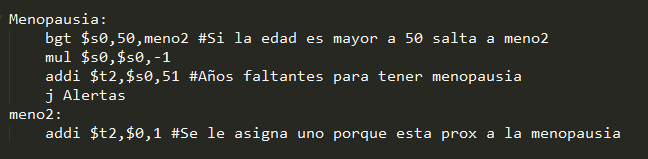
**Calcular si se ha llegado a la etapa de la menopausia:**

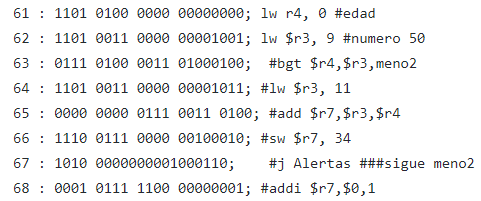
En esta fase se utilizó solo una variable de entrada y es claro que fue la edad y que se considera que después de cierta edad la mujer entra en etapa de menopausia, en el caso de que la mujer no esté en dicha etapa se le calcula un estimado de edad en donde podría estar presentado esa etapa:

**Alto nivel:**

****

**Bajo nivel:**



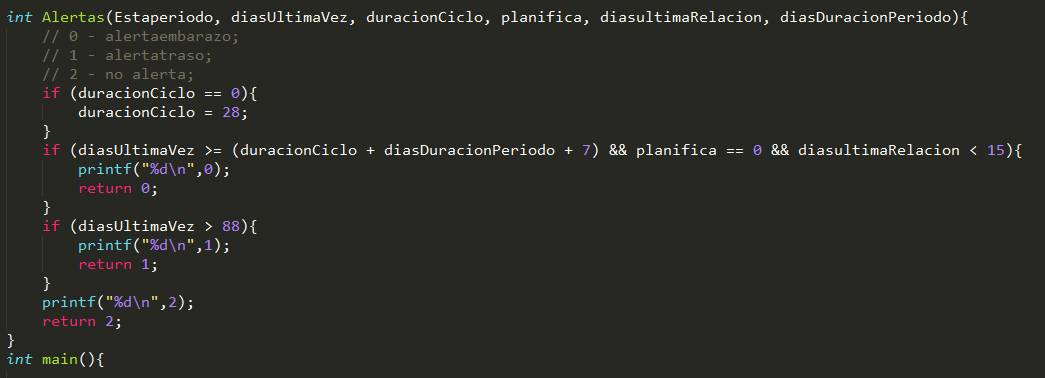


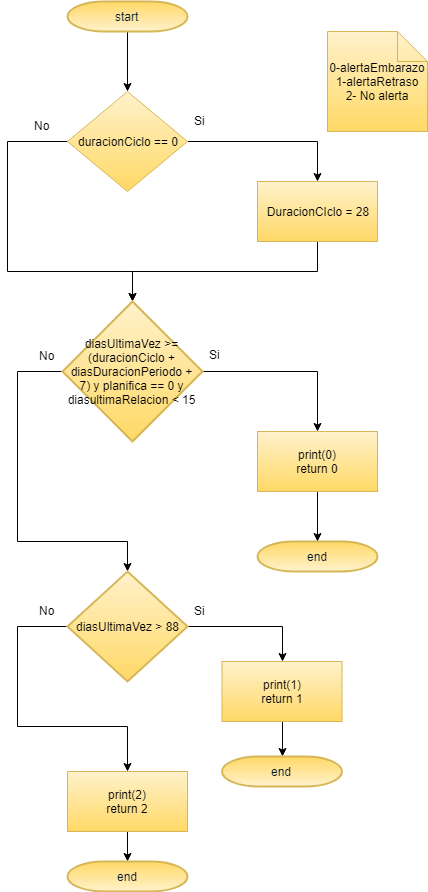
En este caso se realizan dos Lw’s uno para la edad y otro para una constante que fue guardad previamente y se realiza un brach (branch greater than) de comparación para determinar si la mujer está o no en la menopausia y en dado caso que no mediante una suma y una multiplicación se determina en cuanto tiempo le puede llegar a dar la menopausia.

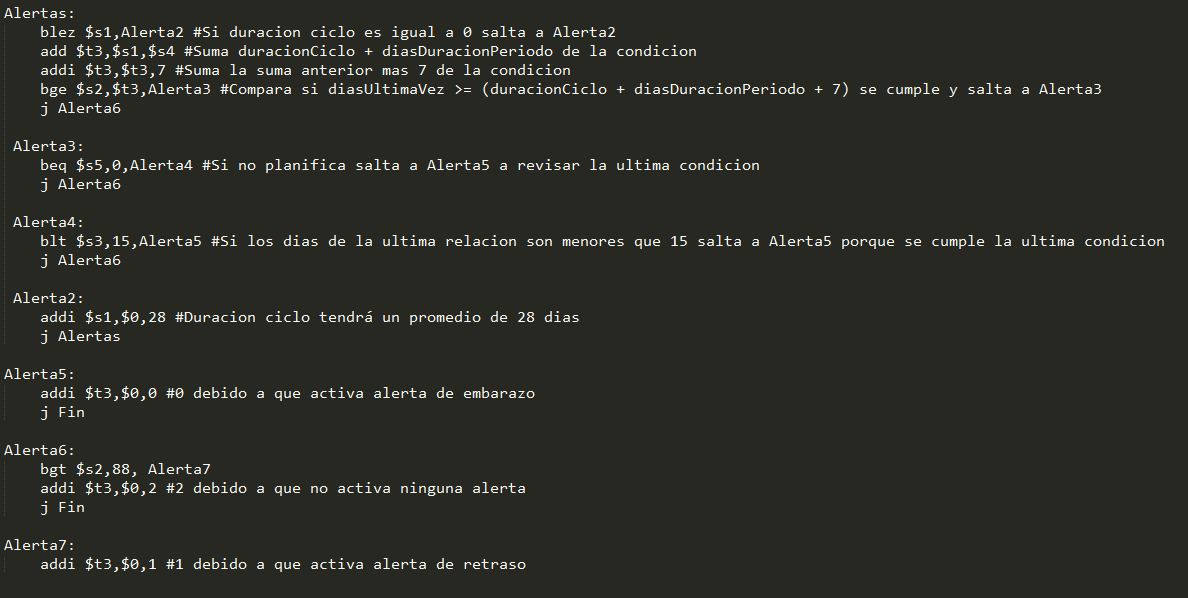
**Calcular las diferentes alertas:**

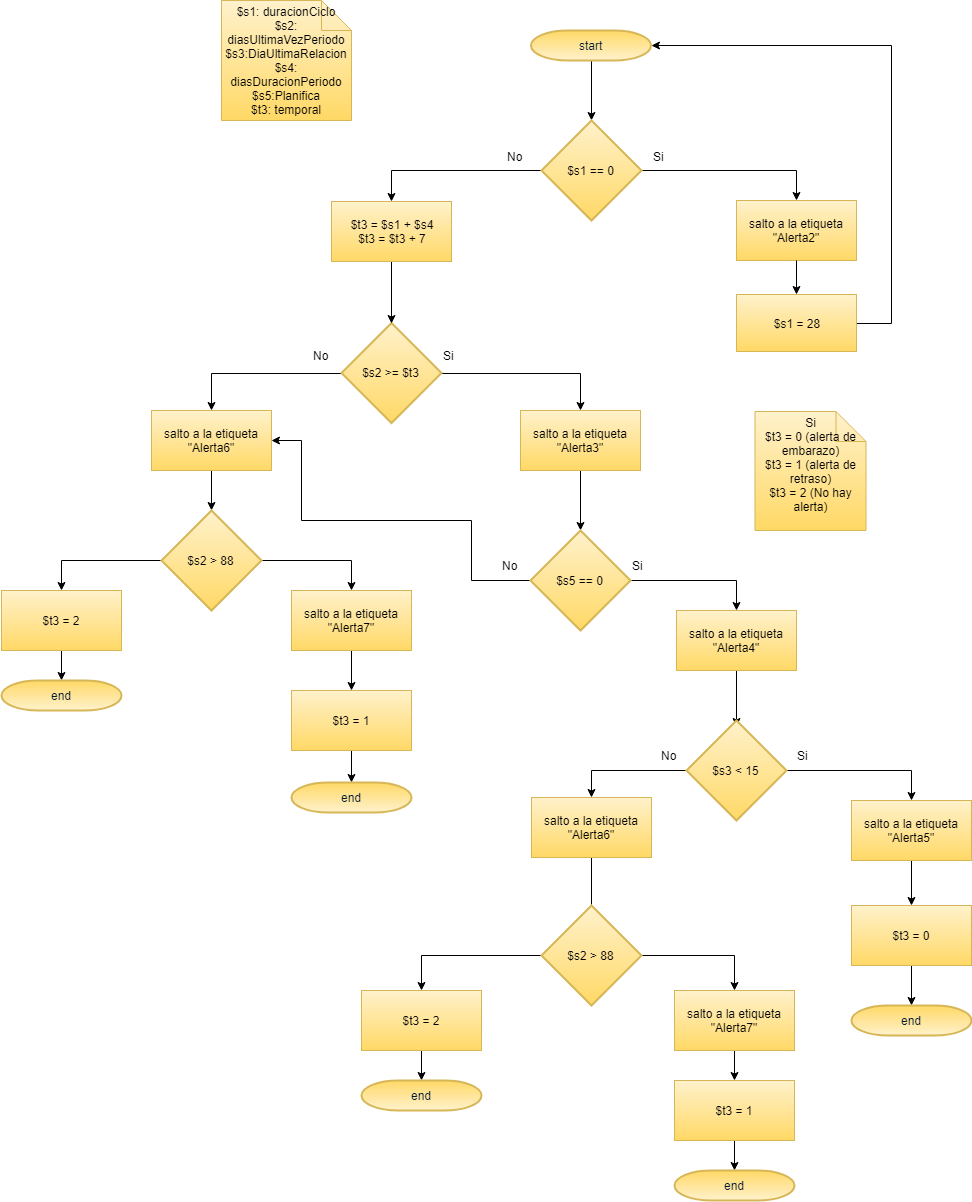
En esta parte del programa se pueden presentar algunas alertas con respecto a los datos ingresados tales como alerta de embarazo, alerta de retraso del periodo, entre otras. Se realizaron varias comparaciones para determinar qué alerta puede tener mujer o si no tiene nada porque preocuparse. El siguiente diagrama explica cómo funciona nuestro algoritmo:

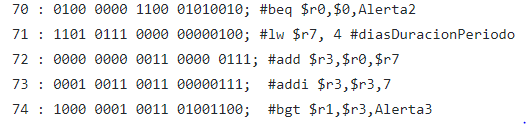
**Alto Nivel:**

****



**Bajo nivel:**

****

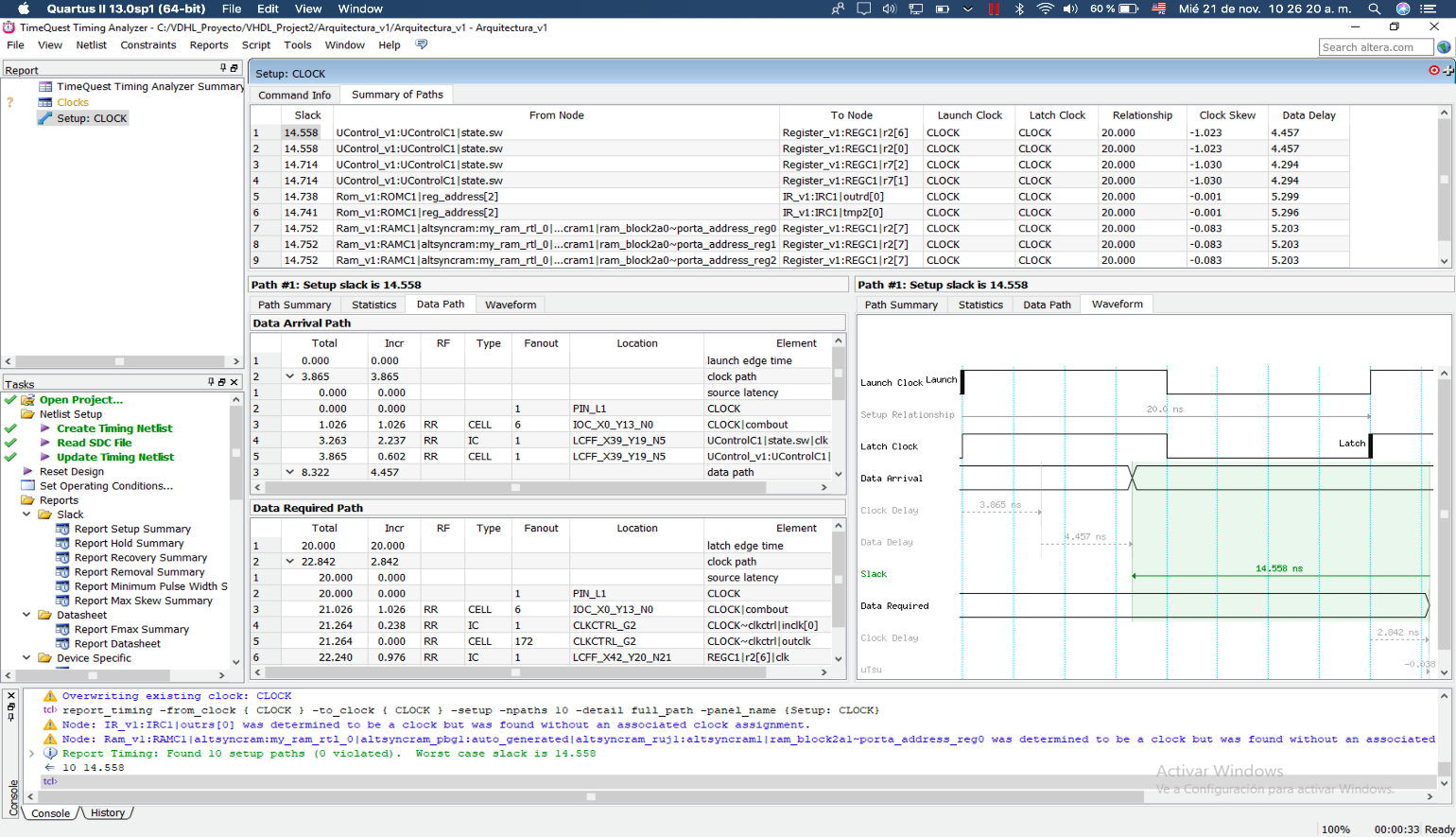


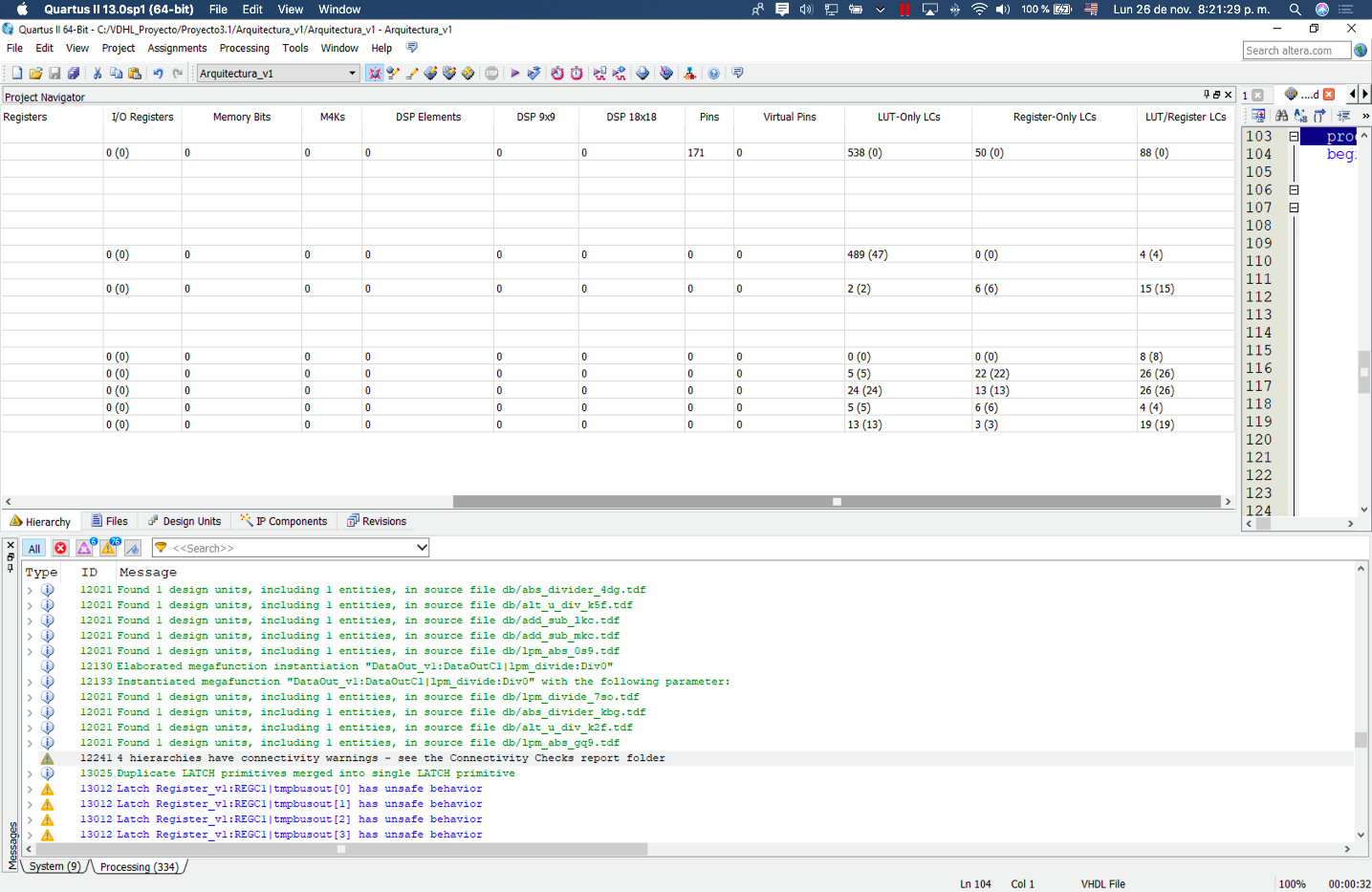
Esta vez no se colocará todas las instrucciones del archivo de inicialización. mif puesto que son demasiadas y por simplicidad omitimos esa parte. Como vemos aquí se realiza una carga de dato y 2 branchs que conducen a otras partes donde hay más branchs por lo que básicamente lo más complejo de este algoritmo es la múltiple comparación de registros.

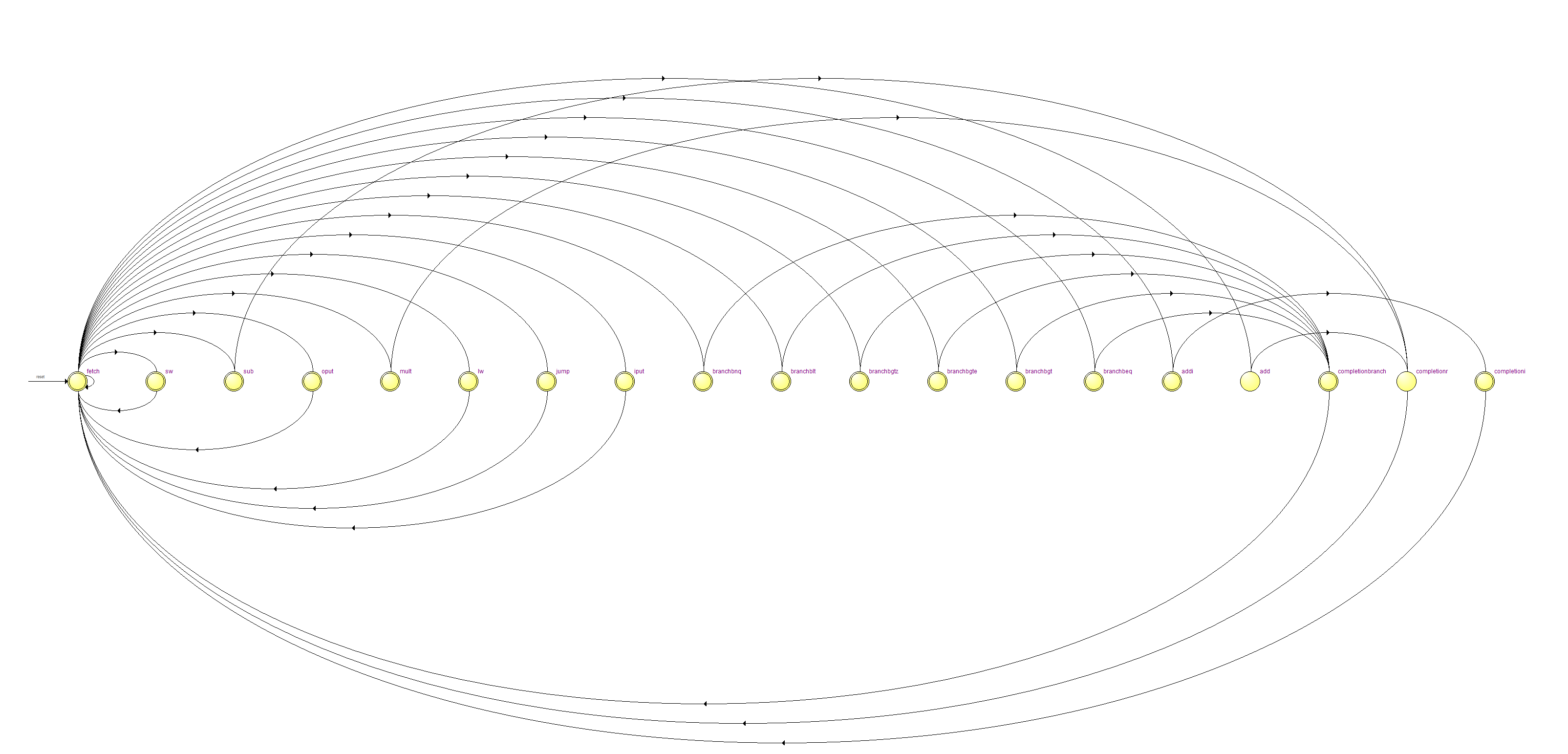
**OUTPUT():**

Esta Instrucción es la encargada de transmitir los resultados que ya se encuentran alojados en la memoria RAM al registro llamado DataOut en donde tomaran medidas para generar los bits correspondientes para que los 7 segmentos de la FPGA puedan visualizar cada uno de los resultados deseados. Por lo que al final de nuestra aplicación es relevante usar esta instrucción con cada uno de los resultados.

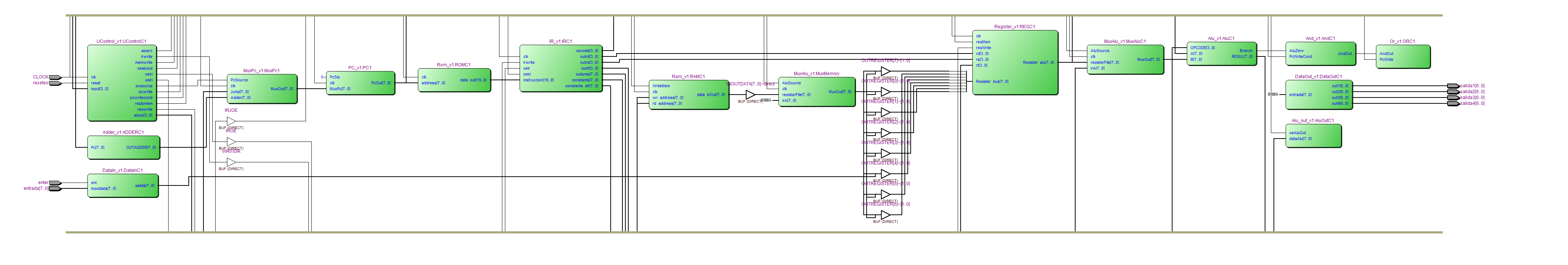
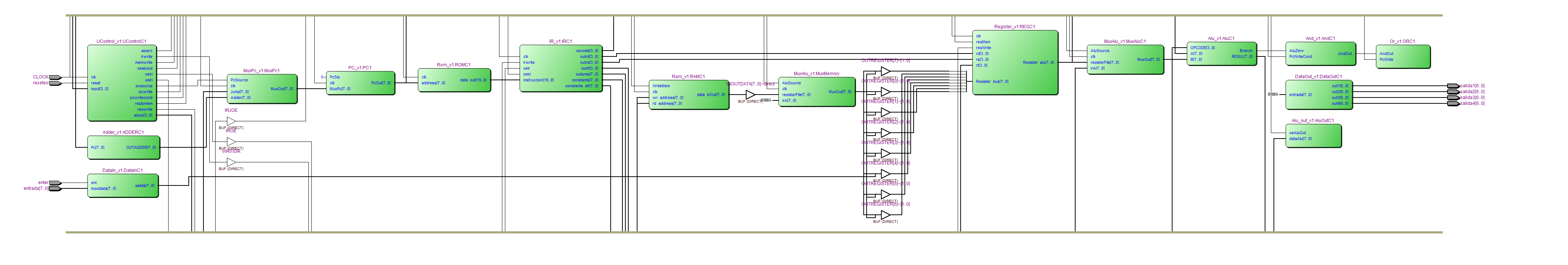
**Patrón de pruebas y revisión de diseño en Quartus II**

****Análisis del Clock:

****Jerarquía de conectividad:

  
Máquina de estados de la arquitectura:

Circuito en bloques de la arquitectura diseñada:



**BIBLIOGRAFIA**

# Lady-Comp (2018). El ciclo menstrual y la ovulación. Recuperado de <http://www.lady-comp.es/ciclo-menstrual>

# Glow. (Año 2011). Eve (Versión 2.12.1) [Aplicación Móvil].

MIPS Reference Data (Green Card). Elsevier, Inc.