Procesador uUTN8B16COP-SF

Capacidades y limitaciones del conjunto de instrucciones

Ing. Gabriel Edgardo Argañaras, Marco Miretti, Lorenzo Depetris, Facundo Busano, Esteban Ceré, Luciano Brezzo

Departamento de Electrónica
UTN Facultad Regional San Francisco
San Francisco, Córdoba, República Argentina

gabriel_arganaras@arnet.com.ar, marco.miretti@gmail.com, depetrislorenzo1@gmail.com
facundobusano@gmail.com, esteban.cere@gmail.com, lucianobrezzo@outlook.com.ar

***Resumen*– el proyecto consistió en el diseño de la arquitectura interna de un procesador didáctico que cumpliese con los requisitos de un procesador real, sin periféricos, y la evaluación de su conjunto de instrucciones a través de programas de aplicación de uso común.**

El mismo fue diseñado por alumnos de la asignatura Técnicas Digitales I año 2015, de la carrera de Ingeniería Electrónica de la Universidad Tecnológica Nacional Facultado Regional San Francisco. Fue denominado uUTN8B16COP-SF. Consta de un repertorio de 16 instrucciones y es una evolución de su antecesor, el uUTN8B4COP, que poseía 4 instrucciones. El grupo de instrucciones se seleccionó de una lista de comandos que permitieran la mayor variedad de acciones. Su desempeño se simuló en el entorno de desarrollo ISIS PROTEUS versión 8.1 a través de la ejecución de diversos programas.

Palabras claves: Microcontroladores – Simulación – Modos de direccionamiento – Capacidad de direccionamiento

I. INTRODUCCIÓN

El procesador didáctico µUTN8B16COP-SF resulta de una nueva versión de los trabajos realizados por los alumnos de la asignatura Técnicas Digitales I del año 2014. En esta ocasión, el objetivo fue ampliar las capacidades del procesador µUTN8B4COP para proveerle un mayor número de instrucciones y simplificar la arquitectura interna. Las premisas de diseño fueron:

- Procesador con arquitectura Harvard.
- Ancho de palabra de datos de 8 bits.
- Conjunto de instrucciones de 16 códigos de operación.
- Bus de direcciones con capacidad de direccionamiento de 256 posiciones de memoria.
- Implementación virtual y simulación de la arquitectura obtenida en el entorno de desarrollo ISIS-Proteus 8.1.

La Fig. 1 muestra la arquitectura de referencia de la CPU usada para el desarrollo del trabajo. El esquema sugerido se basó en la arquitectura del µUTN8B4COP con la que trabajaron los alumnos de Técnicas Digitales I año 2014 [1], y basada a su vez en la máquina sencilla propuesta por Angulo Usategui [2] y en los conceptos básicos de computadoras de John Uyemura [3].

Se propuso una serie de comandos comunes a los microcontroladores de 8 bits de línea baja de mayor uso del mercado local. De la lista inicial se seleccionaron 16, que son los que conforman el conjunto de instrucciones del procesador diseñado. El criterio de selección utilizado fue el de lograr un equilibrio entre brindar la más amplia variedad de acciones, y los modos de direccionamiento. La Tabla 1 muestra el repertorio de instrucciones.

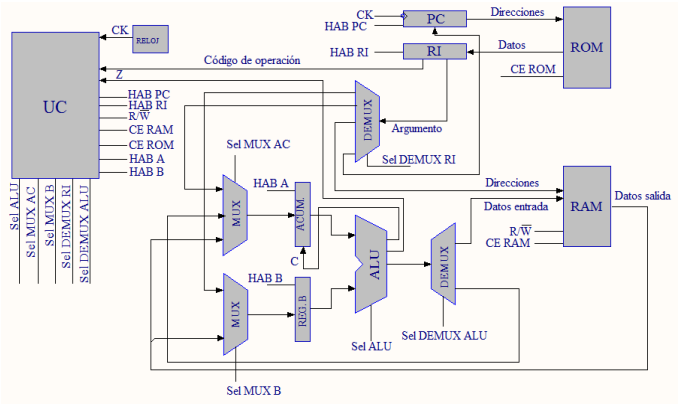

Fig. 1. Procesador µUTN8B16COP-SF, arquitectura interna sugerida.

Tabla 1: repertorio de instrucciones

Instrucciones del µUTN8B16COP-SF		
Nemónico	***Sintaxis***	***Acción***
CargaM	CargaM,@	Carga en el registro acumulador A el contenido de la posición de memoria especificada en el argumento por @.
CargaA	CargaA,#	Carga en el registro acumulador A el literal contenido en el argumento #
GuardaM	GuardaM,@	Guarda el dato actualmente almacenado en el acumulador A en la posición de memoria especificada en el argumento @.
AndA	AndA,#	Realiza la operación lógica AND entre el contenido del acumulador A y el argumento #.
OrA	OrA,#	Realiza la operación lógica OR entre el contenido del acumulador A y el argumento #.
NotA	NotA	Realiza la operación lógica NOT al contenido del acumulador A. Esta operación no tiene argumento.
OrExA	OrExA,#	Realiza la operación lógica OR EXCLUSIVA entre el contenido del acumulador A y el argumento #.
SumaA	SumaA,#	Suma aritmética del contenido actual del acumulador A más el argumento # más el bit de acarreo.
RestaA	RestaA,#	Resta al contenido actual del acumulador A el argumento #.
SumaM	SumaM,@	Suma aritmética del contenido actual del acumulador A de la posición de memoria especificada en el argumento @ más el bit de acarreo.
RestaM	RestaM,@	Resta al contenido actual del acumulador A el contenido de la posición de memoria especificada en el argumento @.
RotAI	RotAI	Desplazamiento hacia la izquierda del contenido actual del acumulador A. El bit entrante proviene del bit de acarreo C. El bit saliente se guarda en el bit C. Esta operación no tiene argumento.
RotAD	RotAD	Desplazamiento hacia la derecha del contenido actual del acumulador A. El bit entrante proviene del bit de acarreo C. El bit saliente se guarda en el bit C. Esta operación no tiene argumento.
BitA0	BitA0,&	Pone a 0 el bit del actual contenido del registro acumulador A especificado en el argumento &. El rango permitido es 0 a 7.
BitA1	BitA1,&	Pone a 1 el bit del actual contenido del registro acumulador A especificado en el argumento &. El rango permitido es 0 a 7.
Salto0	Salto0,@	Salta condicional a la posición de memoria especificada en el argumento @, si el resultado de la última operación fue cero.

La instrucción se divide en dos campos, indicados en la Fig. 2: el Código de operación y el Argumento. El ancho del campo Argumento es de 8 bits, lo que limita su rango de

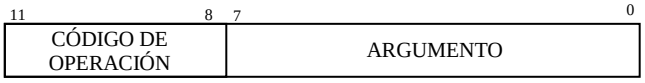

Fig. 2. Formato del código de operación.

direccionamiento a 256 posiciones de memoria o a procesar datos de 00_H a FF_H.

II. EVALUACIÓN DE DESEMPEÑO DEL PROCESADOR

Cada grupo de alumnos utilizó en forma diferente los recursos de diseño para el desarrollo del procesador. Algunos emplearon el criterio de crear componentes a medida mientras que otros utilizaron componentes comerciales o de librería. El modo de trabajo y las diferencias en las arquitecturas internas logradas se vio reflejado durante la simulación. Los distintos procesadores creados no tuvieron la misma velocidad de ejecución en el software de simulación ISIS Proteus 8.1. A modo de ejemplo, el procesador Brezzo-Ceré, no pudo funcionar a una frecuencia de reloj mayor a 1KHz, mientras que el procesador de Miretti-Depetris-Busano logró correr el mismo programa de prueba a una frecuencia de reloj de 2,5MHz.

Una vez verificado el funcionamiento del procesador, se pasó a la siguiente etapa: la evaluación de las capacidades del conjunto de instrucciones para crear rutinas de uso común en procesos de control. Las rutinas consistieron en cálculos matemáticos, movimientos y reordenamiento de datos dentro de una memoria.

Las tareas se repartieron individualmente entre los alumnos creadores de los procesadores antes mencionados. Cada uno desarrolló su programa a partir del enunciado entregado, traduciendo en algunos casos, algoritmos generales de cálculos matemáticos al lenguaje nemónico del procesador µUTN8B16COP-SF.

En los siguientes apartados se presentan tres de los programas que se corrieron en los dos prototipos virtuales seleccionados, con el fin de evaluar el conjunto de instrucciones.

A. Multiplicación binaria

Se planteó el problema de crear un programa que realizara el producto de dos números binarios de 8 bits. Para lograrlo, se descompuso la operación aritmética en tareas que el procesador sea capaz de realizar. Se utilizó el algoritmo iterativo de 8 etapas consistente en desplazamientos y sumas, que se deriva del método de multiplicación decimal en forma manual. El proceso se ve simplificado por el hecho de trabajar con números binarios: solo hay dos tablas de multiplicación que conocer: la tabla del 1 y la del 0. De esta manera, los resultados de las multiplicaciones parciales son el mismo número o cero respectivamente.

En el método iterativo cada iteración comienza desplazando al multiplicador hacia la derecha y evaluando el bit saliente. En caso que el bit analizado sea "1", se suma al resultado parcial, el multiplicando con su orden de magnitud correspondiente. En caso que el bit evaluado sea "0", el

producto parcial es “0”, por lo que no se realiza la suma. La iteración finaliza desplazando el multiplicando hacia la izquierda, aumentando su orden de magnitud para la eventual suma de la próxima iteración.

El programa hace uso de las operaciones de bit y los desplazamientos para evaluar los bits del multiplicador, y de las instrucciones de suma y de desplazamientos para obtener el resultado. Además, se usa la instrucción de resta para decrementar el registro que lleva la cuenta de los ciclos. Las iteraciones se repiten ocho veces, debido al ancho de palabra de los números. Una vez que el contador llega a cero, se vale de la operación de salto para finalizar el programa.

La Tabla 2 presenta el código del programa que realiza la multiplicación de dos números binarios de 8 bits.

La Tabla 3 muestra el mapa de memoria RAM de las variables utilizadas por el programa donde se almacenan los números a operar, resultados parciales de cálculo y el resultado final, para facilitar su entendimiento.

El diagrama de flujo de la Fig. 3 ilustra el programa, pudiéndose observar la lógica planteada anteriormente.

Al finalizar las simulaciones se concluyó que se cumplió el objetivo planteado: el conjunto de instrucciones permitió obtener el programa de multiplicación. En la simulación, se cargaron en el programa números seleccionados para la operación, cuyo producto da un resultado particular, y este fue el esperado. Luego, para continuar probando el programa, se multiplicaron diversos números binarios de 8 bits, siendo nuevamente sus resultados exactos en todas las oportunidades poniendo en evidencia el correcto funcionamiento, tanto del programa como del procesador.

Tabla 2: Código del programa multiplicador

Dir	Código de operación	Dir.	Código de operación
1	CargaA,#Multiplicando	21	GuardaM,@AcB
2	GuardaM,@MB	22	RotaAI
3	CargaA,#multiplicador	23	AndA,#01H
4	GuardaM,@m	24	SumaM,@AcA
5	CargaA,#00H	25	SumaM,@MA
6	GuardaM,@MA	26	GuardaM,@AcA
7	GuardaM,@AcB	27	CargaM,@MB ← SALTO1
8	GuardaM,@AcA	28	SumaA,#00H
9	CargaA,#08H	29	RotaAI
10	GuardaM,@Cont	30	GuardaM,@MB
11	CargaM,@m← SALTO2	31	CargaM,@MA
12	GuardaM,@cm	32	RotaAI
13	CargaM,@m	33	GuardaM,@MA
14	RotaAD	34	CargaM,@Cont
15	GuardaM,@m	35	RestaA,#01H
16	CargaM,@cm	36	GuardaM,@Cont
17	AndA,#01H	37	Salto0,@SALTO3 ←SALTO3
18	Salto0,@SALTO1	38	CargaA,#00H
19	CargaM,@MB	39	Salto0,@SALTO2
20	SumaM,@AcB		

Tabla 3: mapa de variables en RAM

Dirección	Contenido
00_H	Multiplicando parte baja (MB)
01_H	Multiplicando parte alta (MA)
02_H	Multiplicador (m)
03_H	Resultado bajo (AcB)
04_H	Resultado alto (AcA)
05_H	Contador (Cont)
06_H	Copia multiplicando (cm)

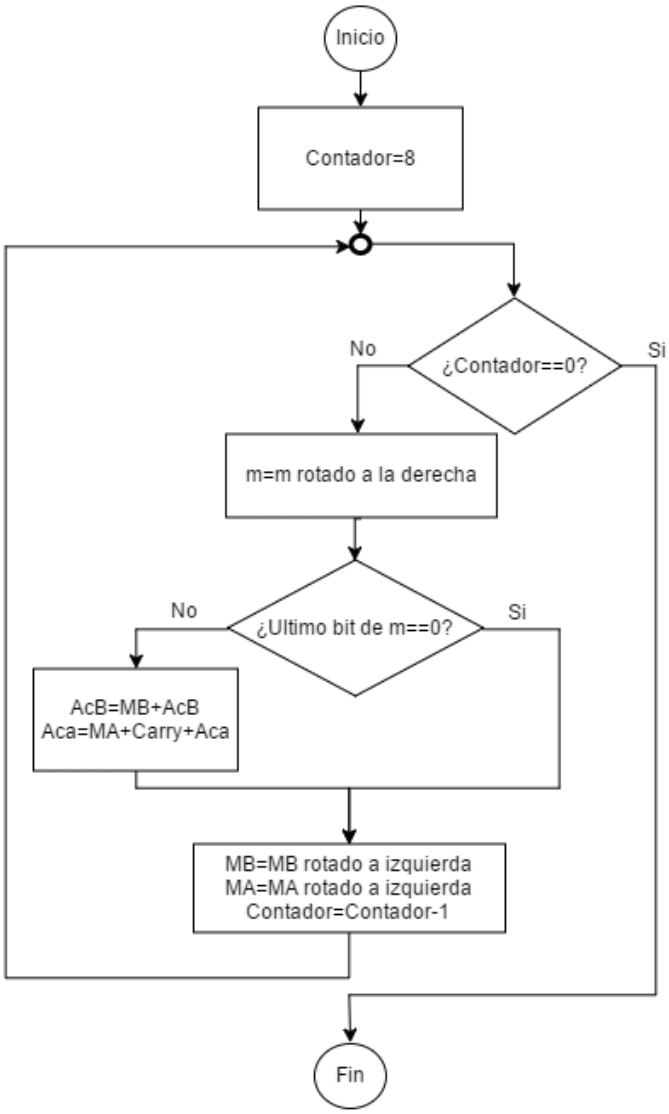

Fig. 3. Multiplicación binaria. Diagrama de fluljos.

B. Mayor, menor y promedio:

El problema propuesto fue hallar el número mayor, el menor y el promedio de un grupo de 16 números binarios de 8 bits. Nuevamente se debió reducir el programa a operaciones que se puedan realizar con el conjunto de instrucciones del µUTN8B16COP-SF.

Durante el desarrollo del programa se presentó el inconveniente de la imposibilidad de realizar saltos indexados. Esto impidió la utilización de estructuras repetitivas o bucles, motivo por el cual las comparaciones se realizaron una a continuación de la otra, dando como resultado un programa extenso. A causa de la limitada capacidad de la memoria de programa, se debió reducir la cantidad de números a procesar a un valor muy inferior a lo propuesto.

La lógica usada para encontrar el máximo es comparar un número con el mayor obtenido hasta el momento. Si el número en consideración supera al mayor, entonces lo reemplazará como nuevo máximo. Si en la comparación resulta ser menor, no se realizan cambios y se pasa a evaluar el siguiente número. Como la lógica es comparativa, se toma el primer número como el mayor y luego se lo compara con el resto, como se describió. La misma lógica es aplicada para hallar menor, pero en este caso se conserva el menor de los números evaluados.

La comparación se planteó como una resta. El signo del resultado es reflejado por el bit "carry" del registro A. Cuando se busca el mayor, al máximo actual se le resta el candidato. Si el resultado es negativo, entonces se está en presencia de un nuevo número mayor y deberá ejecutarse una rutina de reemplazo. En cambio, si el resultado es positivo, no habrá cambios y no se debe entrar en la rutina de reemplazo. Una vez hecha esta evaluación, si no hay un nuevo mayor, se verificará si es el menor, utilizando la misma lógica. Si el menor actual menos el número candidato es negativo, no se efectuarán cambios. Si la resta es positiva, se ejecutará la rutina de reemplazo correspondiente al número menor.

En la rutina de reemplazo, la evaluación del bit carry se realizó a través de operaciones de bits y de desplazamiento al no disponer de comandos de evaluación de bits. Se ejecuta una operación con resultado 0, que no afecte el bit de carry, para borrar el acumulador. Luego se lo rota en cualquier dirección. Si el resultado sigue siendo 0 entonces la resta anterior fue positiva, en caso contrario el resultado fue negativo.

La segunda parte del programa se encarga de obtener el promedio de los números. Se logró de manera simple con las operaciones de suma y desplazamiento a la derecha. Debido a que al momento de las pruebas no se dispone de una rutina de división, la cantidad de números debe ser siempre una potencia de 2. De esta manera la división se logra simplemente desplazando hacia la derecha tantas veces como indique el exponente. La cantidad de números se redujo a 8 por las limitaciones de la capacidad de memoria.

Se pudo observar que este programa llevo al procesador al límite de sus capacidades tanto en las acciones que se pueden realizar con el conjunto de instrucciones, como en la capacidad de direccionamiento de la memoria de programa. Queda en evidencia la necesidad de mejorar los modos de direccionamiento y capacidad de ROM. El programa se evaluó con diferentes grupos de números binarios de 8 bits obteniendo resultados correctos en todas las oportunidades. La Tabla 4 muestra un fragmento del programa. La Fig. 4 muestra el diagrama de flujo de la determinación de los números mayor y menor.

Tabla 4: Rutina de comparación y reemplazo

Código de op	Dir.	Código de op	Dir.
CargaM,12	1A	GuardaM,12	21
RestaM,01	1B	CargaM,11	22
AndA,00	1C	RestaM,01	23
RotaI	1D	AndA,00	24
Salto0,21	1F	RotaI	25
CargaM,01	20	Salto0,D0	26

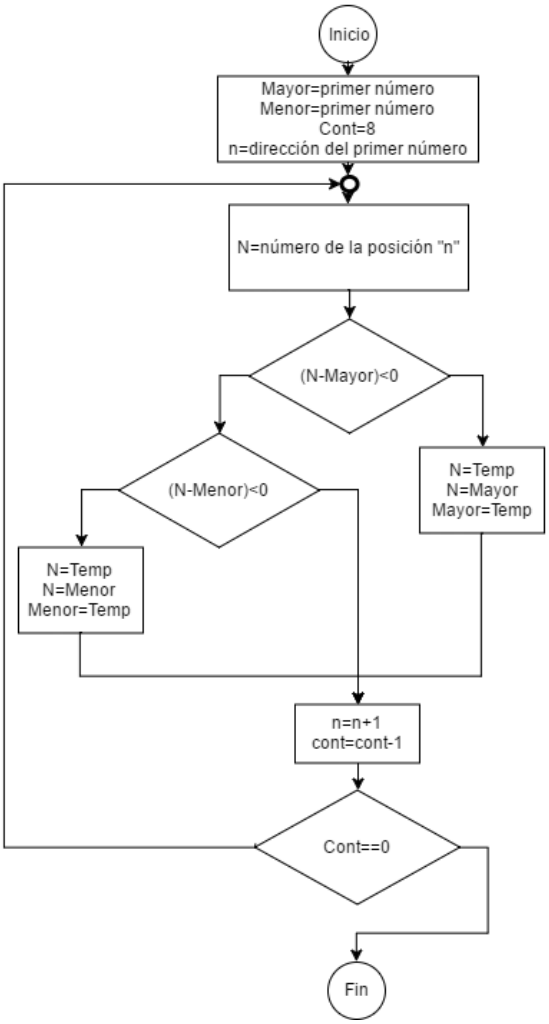

Fig. 4. Mayor, menor y promedio. Diagrama de flujo.

C. Conversor de binario a BCD:

El tercer problema propuesto fue el de convertir números binarios de 16 bits a Decimal Codificado Binario (BCD). En este caso se utilizó el método iterativo "double dabble", que se basa en desplazamientos y sumas.

Para implementar el método "double dabble" se utilizaron siete registros: dos para guardar el número inicial y cinco para el resultado final. Los registros se ubican de manera que el número inicial quede a la derecha de los registros de resultado. En cada iteración todos los registros se desplazan una posición hacia la izquierda. El bit saliente del número a convertir ingresa a los registros que guardarán el resultado de la conversión. En cada vez se observa si el contenido de alguno de los registros del resultado es mayor a 4. En este caso se deberá sumar 3 a dicho registro. Las iteraciones continúan con esta regla de procedimiento hasta que se hayan efectuado 15 desplazamientos y se vacían los registros del número inicial. Luego del último desplazamiento el resultado queda definido, sin tener que realizar correcciones si es mayor a 4. La Tabla 5 muestra un ejemplo de 8 bits del mismo procedimiento.

En el problema planteado, el número a convertir es de 16 bits. El byte bajo se guarda en la posición de memoria "BL", mientras que el byte alto en "BA". El programa se ejecuta como si el número binario fuese de 8 bits. Se carga el byte alto en un registro "BT", luego cuando ya se han realizado 8 desplazamientos se reemplazan los bits altos por los bajos en el registro BT.

A continuación, se verifica si el contenido de cada registro de resultado es mayor que 4. Esta evaluación se realiza a través de la operación de resta de dicho registro menos 5, y la observación del bit de carry. Si el resultado es positivo o cero, el número fue mayor a 4 y por lo tanto se sumará 3. En caso contrario se continuará sin realizar modificaciones. En la evaluación del bit de carry se utilizó el mismo mecanismo que en el caso del programa mayor y menor.

El último desplazamiento se realiza sin tener en cuenta el contenido de los registros resultados, dando por finalizado el programa.

La Tabla 6 muestra las direcciones de la memoria RAM y su contenido.

Tabla 5: Ejemplo de double dabble de 8 bits

Centenas	Decenas	Unidades	Binario	Operación
0000	0000	0000	11110011	
0000	0000	0001	11100110	RotAI
0000	0000	0011	11001100	RotAI
0000	0000	0111	10011000	RotAI
0000	0000	1010	10011000	Suma 3 a Unidades porque fue 7
0000	0001	0101	01100000	RotAI
0000	0001	1000	01100000	Suma 3 a Unidades porque fue 5
0000	0011	0000	01100000	RotAI
0000	0110	0000	11000000	RotaI
0000	1001	0000	11000000	Suma 3 a Decenas porque fue 6
0001	0010	0001	10000000	RotAI
0010	0100	0011	00000000	RotAI
2	4	3		

Tabla 6: mapa de variables en la memoria RAM

Dirección	Contenido
00_H	Bits binarios bajos (BB)
01_H	Bits binarios altos (BA)
02_H	Unidades BCD (U)
03_H	Decenas BCD (D)
04_H	Centenas BCD (C)
05_H	Unidades de mil BCD (UM)
06_H	Decenas de mil BCD (DM)
07_H	Contador 1
08_H	Contador 2

El programa se probó con gran cantidad de números binarios de 16 bits obteniéndose la conversión correcta a BCD en todas las oportunidades.

En el diagrama de flujo del programa que se muestra en la Fig. 5, se puede apreciar de una manera simple la lógica utilizada.

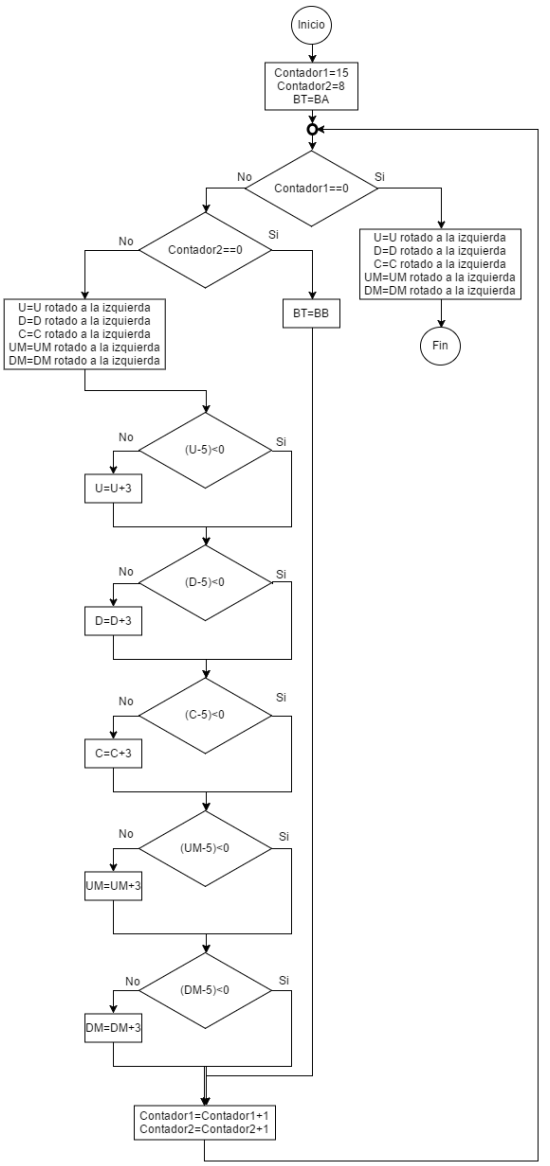

Fig. 5. Conversor binario 16 bits a BCD. Diagrama de flujos.

D. Otros programas:

Además de los programas mostrados en esta presentación, se ha avanzado mucho en nuevos programas para continuar evaluando el procesador. La rutina de división entera de números de 16 bits se encuentra en mayor grado de avance. El método usado en este caso se basa en 16 iteraciones donde se realizan desplazamientos y restas. Se están evaluando otros métodos numéricos de obtención del resultado.

III. CONCLUSIONES

Las pruebas demostraron que las diferentes arquitecturas de los procesadores son transparentes a los programas: los resultados de las operaciones matemáticas fueron iguales y correctos, independientemente del procesador usado.

El procesador Miretti-Depetris-Busano demostró el mejor desempeño de todos los modelos logrados, alcanzando una frecuencia de trabajo de 2,5MHz en las pruebas del simulador. Se adoptará esta arquitectura como base para la futura evolución del procesador.

Durante las pruebas se evidenciaron los límites que posee este procesador didáctico: un repertorio de instrucciones y una capacidad de direccionamiento limitados. La primera característica condiciona el diseño del camino de datos de manera que solo es capaz de proporcionar dos modos de direccionamiento: directo e implícito. No es posible realizar operaciones de salto indexados, quedando reducida la capacidad de escribir programas con ciertas estructuras repetitivas.

La segunda de las características adversas condiciona la longitud de los programas, más teniendo en cuenta las limitaciones en la construcción de bucles o lazos.

Los problemas enumerados fijan las pautas de diseño para la nueva evolución del procesador:

- Ampliación del set de instrucciones a 32 comandos.
- Incorporación de instrucciones de llamado y retorno de subrutinas.
- Incorporación de instrucciones con direccionamiento indexado e indirecto.
- Mecanismo de interrupción.
- Ampliación de la capacidad de direccionamiento de la memoria de programa.
- Conexión al exterior a través de puertos de entrada y salida.

Los resultados obtenidos con el procesador didáctico permitieron abrir dos nuevas líneas de trabajo, actualmente en desarrollo. La primera de ellas es la descripción de la arquitectura interna del µUTN8B16COP-SF en lenguaje VHDL. Esta línea de trabajo tiene como objetivo la adquisición de conocimientos para facilitar la transición hacia el nuevo temario de la cátedra de Técnicas Digitales I de la carrera de Ingeniería Electrónica de la Universidad Tecnológica Nacional Facultado Regional San Francisco.

La segunda línea de trabajo consiste en el diseño de un entorno de desarrollo para escribir y simular programas en lenguaje ensamblador propio del procesador. Este entorno de desarrollo será de uso corriente durante el año lectivo de la asignatura. El objetivo es proporcionar al alumno conocimientos básicos de programación en lenguaje ensamblador.

IV. REFERENCIAS

[1] G. Argañaras, G. Cervetti, J. Ramírez, A. Yoaquino, V. Boasso, "Procesador uUTN8B4COP Diseño y simulación de un procesador didáctico" , artículo 4 FPGA y lógica programable, libro del VI Congreso de Microelectrónica Aplicada, Universidad de La Matanza, San Justo, Buenos Aires, Argentina - mayo 2015.

[2] José María Angulo Usategui, Javier García Zubía, "Sistemas Digitales y Tecnología de Computadores, capítulo 12: la Máquina sencilla" Editorial Paraninfo, 2002.

[3] John P. Uyemura, , "Diseño de Sistemas Digitales, un enfoque integrado, capítulo 11: Conceptos básicos de computadoras" International Thomson Editores, 2000.