Hacia una Especificación Formal de un Microcontrolador usado en Marcapasos

Carlos Luna[1], Luis Sierra[1], y Paula Echenique[2]

[1] Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay
{cluna, sierra}@fing.edu.uy

[2] Dirección de Formación y Perfeccionamiento Docente, ANEP, Uruguay
pechenique@anep.edu.uy

Resumen. Un microcontrolador es un circuito integrado que contempla las funcionalidades de un computador. Los avances tecnológicos de las últimas décadas han permitido que los microcontroladores se usen hoy en diferentes ámbitos; por ejemplo, en dispositivos de comunicación como los celulares o de control médico como los marcapasos. En este trabajo analizamos una familia de microcontroladores programables llamada PIC. El microprocesador cuenta con un programa modificable, que lo convierte, esencialmente, en una computadora de uso general, pequeña, barata, y versátil. Nuestro interés se centra en la verificación de programas críticos desarrollados en este entorno. En particular, nuestro objetivo final es el estudio de programas para dispositivos biomédicos implantables. En este artículo presentamos la formalización de un PIC usando máquinas de estados. Especificamos el comportamiento de las instrucciones de la máquina y certificamos, usando el asistente de pruebas Coq, el funcionamiento de la misma, expresado a través de una semántica operacional.

Palabras Claves: Microcontroladores (PICs), Especificaciones Formales, Coq.

1 Introducción

A principios de los setenta la tecnología permitió ensamblar todos los componentes de la unidad central de un procesador en un único chip, al que llamamos microprocesador. Los microcontroladores constituyen un paso más en el proceso de miniaturización; todas las funcionalidades de un computador, incluyendo la entrada/salida y los relojes, se encuentran alojados en el mismo circuito integrado. Los avances tecnológicos de las últimas décadas han permitido que los microcontroladores se hayan vuelto cada vez más pequeños, más resistentes a las condiciones ambientales, más potentes y más baratos. En consecuencia, la difusión de esta tecnología ha sido explosiva [1].

Hoy encontramos microcontroladores en todos los ámbitos de la producción industrial, así como en la vida social de los individuos; dispositivos de comunicación como los celulares, de control médico como los marcapasos, son sólo algunos ejemplos que nos hablan de esta omnipresencia. De allí el nombre de computación penetrante (*pervasive computing*).

En este trabajo analizamos una familia de microcontroladores programables llamada PIC [2,3]. Esto quiere decir que el microprocesador cuenta con un programa

modificable, que lo convierte, esencialmente, en una computadora de uso general, pequeña, barata, y versátil. Nuestro interés se centra en la verificación de programas críticos desarrollados en este entorno.

Los programas implementados sobre los microcontroladores se comportan de la siguiente manera: observan la situación del ambiente o entorno, y actúan de forma de preservar cierta condición de coherencia en el ambiente. La interacción se realiza mediante dos interfaces; los sensores que proporcionan información sobre el medio, y los actuadotes que realizan acciones sobre éste. Por ejemplo, si el PIC está destinado a controlar un marcapasos, tendrá sensores que detecten la actividad del corazón del paciente; cuando la misma no sea la adecuada, estimulará al corazón de forma que retome su actividad normal, registrando la situación para posterior análisis por el médico.

Llamamos sistemas reactivos a estos sistemas informáticos que se caracterizan por reaccionar a los cambios del entorno de forma de preservar cierta situación. Estos sistemas no responden a un comportamiento funcional simple; no podemos representarlos mediante el cómputo de una función en su sentido matemático, como es habitual en las rutinas de C o Java. Para comprender el programa cargado en el PIC precisamos otro modelo computacional, donde el programa se represente sino como un mecanismo de observación y acción sobre el medio.

El modelo elegido es el de una máquina de estados. En cada momento el sistema se encuentra en un determinado estado, y dependiendo de las acciones que se detecten o realicen, el estado irá modificándose. Es necesario definir claramente tanto el estado como las reglas que lo modifican; la máquina representará al PIC, y las reglas al lenguaje ensamblador del PIC. En este trabajo presentamos estos modelos, que constituyen lo que se conoce como semántica operacional.

Nuestro interés en el modelado del PIC se justifica por la necesidad de verificar propiedades sobre los programas. En particular, estamos interesados en mostrar la viabilidad de probar formalmente la seguridad de ciertos programas usados en dispositivos biomédicos implantables. Las propiedades de interés pueden caracterizarse como extensionales o intensionales. Las propiedades extensionales refieren a elementos que aparecen explícitos en el PIC; por ejemplo, determinar si cierto sector protegido de la memoria es modificado de forma ilegal. Las propiedades intensionales refieren a elementos que no aparecen en el PIC, pero que pueden llegar a modelarse; por ejemplo, si el tiempo transcurrido entre la lectura de los sensores es suficiente para garantizar le corrección de esa lectura.

Un criterio de sanidad de los programas es la verificación de que el programa cumple determinadas propiedades. Un modelo formal de la semántica operacional del PIC, del programa y de la propiedad que deseamos que se cumpla, nos permiten probar o refutar la corrección del programa. La especificación del modelo se construirá usando el sistema Coq [4,5], de forma de asegurarnos su corrección. De esta manera obtenemos demostraciones verificadas por una computadora de la corrección de programas ejecutables en el PIC. Una ventaja adicional de usar un asistente de pruebas basado en teoría de tipos como Coq radica en la posibilidad de extraer prototipos ejecutables certificados de ciertas pruebas. Si bien no abordaremos esto último, como tema central en este artículo, esta posibilidad justifica aún más el interés de nuestro encare.

La organización del resto de este artículo es como sigue. En la sección 2 describimos las características fundamentales del PIC considerado. En la sección 3 incluimos formalizaciones de los componentes y el conjunto de instrucciones del PIC. Luego, especificamos el comportamiento de las instrucciones, en la sección 4, y definimos una semántica operacional de la máquina formalizada, en la sección 5. Finalmente, analizamos la corrección del prototipo de la máquina, representada por la semántica operacional, respecto al comportamiento de las instrucciones especificado en la sección 4. En la sección 6 presentamos las conclusiones y los trabajos futuros.

2 Descripción del Microcontrolador

En este trabajo consideramos el PIC 16F84 [2,3], usado por ejemplo en dispositivos biomédicos implantables. Como otros microcontroladores, el PIC 16F84 (PIC, de aquí en más) es un circuito integrado que incluye en su interior las tres unidades funcionales de una computadora: procesador, memoria y, unidades de entrada y salida. El juego de instrucciones del PIC es pequeño, conforme a las arquitecturas RISC habituales.

Los microcontroladores se clasifican en tres gamas: alta, media y baja, según cantidad y largo de las instrucciones, número de puertos, funcionalidades, cantidad y tipo de memoria. El PIC considerado pertenece a la gama media. Posee un conjunto de 35 instrucciones de 14 bits, 2 puertos, ambos de entrada/salida, y 256 bytes de memoria RAM. Al utilizar arquitectura Harvard dispone de dos memorias independientes, una para datos y otra para el programa. De esta manera el procesador puede acceder en forma independiente y simultánea a ambas.

El PIC tiene un registro particular, denominado registro de trabajo (W), que puede ser usado en las operaciones aritméticas. La única forma de direccionarlo es mediante instrucciones específicas. Por ejemplo, se usa la instrucción *movf* para mover la información de un registro, pero *movwf* si el registro en cuestión es W. Este registro W puede utilizarse para almacenar el resultado de las operaciones al igual que cualquier registro de la memoria de datos.

La memoria está organizada en tres bloques: (1) la memoria de programa de 1k x 14 bits; (2) la memoria de datos, que cuenta con dos bancos de 128 registros de un byte cada uno, denominados banco0 y banco1; y (3) una pequeña memoria EEPROM de datos, que cuenta con 64 registros de un byte para almacenar información en caso de corte de alimentación. Cada banco de la memoria de datos se divide en dos partes: la que contiene los registros de funciones especiales, conformada por los 12 primeros registros; y la que abarca a los registros de propósito general. Son 68 registros diferentes que se usan para almacenar los datos temporales del programa en ejecución. Los registros de propósito general de ambos bancos mapean al banco 0. El resto de los registros de los dos bancos de la memoria de datos no están implementados para escritura y devuelven 0 en caso de lectura.

El PIC posee una pila para la ejecución de subrutinas. La pila tiene capacidad para 8 elementos y no hay ningún mecanismo de hardware que indique desbordamiento de la misma. Si un programa causa el desbordamiento de la pila, se perderá automáticamente el primer elemento ingresado. Esto es, sólo se conservan las últimas 8 entradas a la pila.

3 Formalización del Microcontrolador

En esta sección presentamos una formalización de los componentes básicos del PIC y detallamos un subconjunto de sus instrucciones. La idea es describir el funcionamiento del PIC como una máquina de estados, donde un estado de la máquina representa el estado de la memoria, y se modifica de acuerdo a las instrucciones ejecutadas.

3.1 Componentes del Sistema

Memoria de Datos. La memoria de datos es una función, que dada una dirección devuelve el registro contenido en dicha posición.

Llamamos `DirMD` al espacio de direcciones de la memoria de datos. Cada registro es una función que dada una posición devuelve su contenido; formalmente, `Reg:Pos→Bit`. `MD` es el tipo de la memoria de datos. Esto es, `MD:DirMD→Reg`.

Memoria de Programa. La memoria de programa es una función que dada una dirección devuelve la instrucción contenida en dicha posición.

Llamamos `DirMP` al espacio de direcciones de la memoria de programa. Cada instrucción contiene un código de operación y sus correspondientes operandos. Llamamos `MP` a la memoria de programa, `MP:DirMP→I`. El contador del programa, `PC`, es un elemento de `MP`.

Pila de Ejecuciones. Es una estructura de pila (stack), usada para la ejecución de subrutinas, que contiene direcciones de la memoria de programa. Consideraremos las operaciones habituales para manejo de pilas, teniendo en cuenta el comportamiento de la inserción (push) detallado en la sección 2.

Estados. Un estado es una tupla compuesta por: una configuración de la memoria de datos, un valor del registro `W`, el contenido del `PC` y una configuración de la pila de ejecuciones. Cada estado es un registro `e` = ⟨`md`, `W`, `PC`, `s`⟩; y sus proyecciones son `e.md` = `md`, `e.W` = `W`, `e.PC` = `PC` y `e.s` = `s`, respectivamente.

3.2 Instrucciones

El PIC tiene 35 instrucciones, que pueden clasificarse en cuatro grandes categorías, según el tipo de operandos y la operación que realizan: instrucciones de literal (o de valor constante), instrucciones de byte, instrucciones de bit e instrucciones de control. En este trabajo consideramos, por razones de espacio, un subconjunto de las instrucciones del PIC. La elección incluye instrucciones en cada una de la categorías mencionadas, según muestra la tabla 1.

Tabla 1. *Instrucciones consideradas del PIC.*

Instrucción	Semántica informal
	Instrucción de literal
addlw l	Suma el literal `l` con `W`. Almacena el resultado en `W`.
	Instrucciones de byte
addwf f el	Suma `W` con el contenido del registro de dirección `f` (registro f de aquí en más) y almacena el resultado en `W` si el parámetro booleano `el` contiene el valor false, ó en `f` en caso contrario.
decfsz f el	Decrementa el contenido del registro `f` en 1 y almacena el resultado en `W`, si el parámetro booleano `el` contiene el valor false, ó en `f` en caso contrario. Asimismo, si el valor inicial de f es 1, se saltea la siguiente instrucción.
movf f el	Copia el contenido del registro `f` en `W`, si el parámetro booleano `el` contiene el valor false, ó en `f` en caso contrario.
clrf f	Pone en cero el registro `f`.
	Instrucciones de bit
bcf f p	Pone en 0 el bit `p` del registro `f`.
btfss f p	Saltea la siguiente instrucción si el bit `p` del registro `f` es 0.
	Instrucciones de control
goto k	Salta a la dirección `k`.
call k	Llama a una subrutina en la dirección `k`.
return	Retorno de una subrutina.

4 Especificación de la Máquina de Alto Nivel

En esta sección presentamos una semántica axiomática de las instrucciones del PIC, usando pre y postcondiciones. Esta es una abstracción de la máquina real, donde se define el comportamiento del sistema mediante condiciones sobre los estados previo y posterior a la ejecución de una instrucción. Llamaremos Máquina de Alto Nivel (MAN) a la especificación del PIC en base a la semántica axiomática referida.

Antes de presentar la semántica formal de las instrucciones, necesitamos realizar un par de observaciones sobre los registros especiales y de uso general.

- Hay un registro especial, llamado STATUS (`ST`), cuyos bits señalan condiciones resultantes de ciertas operaciones. Las operaciones que presentamos usan los bits 0 y 2 de ese registro, que indican que la última operación aritmética produjo acarreos o dio cero como resultado. Usamos las siguientes condiciones para referirnos a esta situación.
  - CARRY exp e $=_{def}$ exp > 215 $\leftrightarrow$ (e.md ST 0) = 1.
  - ZERO exp e $=_{def}$ exp = 0 $\leftrightarrow$ (e.md ST 2) = 1.
- Es importante asegurar que se está accediendo el banco correcto para realizar una operación sobre un registro especial. El bit 5 del registro `ST` indica el

banco al que se tiene acceso en un momento dado. La condición de seguridad necesaria es: bancoOK e f $=_{def}$ (e.md f) $\in$ banco0 $\Leftrightarrow$ (e.md ST 5) = 0.

- Si se desea acceder para lectura o escritura a un registro de propósito general, hay que asegurar que éste se encuentre en el rango de los registros de propósito general implementados, es decir que sea modificable.
- Existen algunos bits de registros especiales que no pueden ser modificados por un programa.

La tabla 2 incluye operaciones que serán utilizadas para describir, en la tabla 3, las pre y postcondiciones de las instrucciones presentadas en la tabla 1.

Tabla 2. *Operaciones auxiliares*. Se usará la pareja de bits (f2 f0), de tipo F, cuyos elementos representan, respectivamente, las flags cero y acarreo.

Operación	Semántica Informal
$actF : MD \to F \to MD$	*actF* actualiza las flags en la memoria según el contenido de un elemento de F. Si, por ejemplo, la flag de acarreo no debe ser modificada, la notación usada será: (f2, _).
$\downarrow : Reg \to N$	$\downarrow$ retorna el valor decimal de un registro. N representa al conjunto de los números naturales.
$\uparrow : N \to Reg$	$\uparrow$ devuelve un registro con el valor de un número entre 0 y 255.
$e \equiv e'[a, b \ldots]$	$\equiv$ es una relación usada para establecer que dos estados e y e' difieren a lo sumo en los componentes listados entre corchetes. Estos componentes pueden ser registros de la memoria de datos, bits de estos registros, o componentes de un estado.
$modifReg : Reg \to bool$	*modifReg* indica si un registro está en el rango de direcciones implementadas de la memoria.
$modifBit : Reg \to Posición \to bool$	*modifBit* indica si la posición de un registro es modificable por el programa.
$updReg : MD \to Reg \to Reg \to MD$	*updReg* actualiza el contenido del primer registro con el del segundo en la memoria dada.
$updBit : MD \to Reg \to Pos \to Bit \to MD$	*updBit* actualiza una posición de un registro en la memoria con un bit dado.

Tabla 3. *Precondiciones de las instrucciones de la tabla 1*. Denotamos con e al estado sobre el cual se establece la precondición.

Instrucción	Precondición
addlw l	`true`
addwf f el	$el \to bancoOk\, e\, f \wedge modifReg(e.md\ f)$
movwf f el	$el \to bancoOk\, e\, f \wedge modifReg(e.md\ f)$
decfsz f el	$el \to bancoOk\, e\, f \wedge modifReg(e.md\ f)$
clrf f	$bancoOk\ e\, f \wedge modifReg\ (e.md\ f)$
bcf f p	$bancoOk\, e\, f \wedge modifReg\,(e.md\, f) \wedge modifBit\,(e.md\ f\ p)$
btfss f p	`true`
goto k	`true`

call k	$\lvert e.s \rvert < 8$
return	$0 < \lvert e.s \rvert$

Tabla 4. *Postcondiciones de las instrucciones de la tabla 1*. Denotamos con `e` al estado previo a la ejecución de una instrucción y con `e'` al estado resultante. La variable `destino` refiere al destino del resultado de la ejecución de una instrucción; según el valor del parámetro booleano de la instrucción, `destino` puede ser el registro W ó algún registro de la memoria de datos.

Instrucción	Postcondición
addlw l	$e'.W = \uparrow(\downarrow e.W + \downarrow l) \wedge \text{CARRY}(\downarrow e.W + \downarrow l)e' \wedge \text{ZERO}(\downarrow e.W + \downarrow l)e' \wedge$ $e'.PC = e.PC + 1 \wedge e \equiv e'[(md\ ST\ 0), (md\ ST\ 2), W, PC]$
addwf f el	$(el \rightarrow (e'.md\ f) = \uparrow(\downarrow e.W + \downarrow (e.md\ f))) \wedge (\neg el \rightarrow e'.W = \uparrow(\downarrow e.W + \downarrow (e.md\ f))) \wedge$ $\text{CARRY}(\downarrow e.W + \downarrow (e.md\ f))e' \wedge \text{ZERO}(\downarrow e.W + \downarrow (e.md\ f))e' \wedge$ $e'.PC = e.PC + 1 \wedge e \equiv e'[destino, (md\ ST\ 0), (md\ ST\ 2), PC]$
movf f el	$(el \rightarrow (e'.md\ f) = (e.md\ f)) \wedge (\neg el \rightarrow e'.W = (e.md\ f)) \wedge$ $\text{ZERO}(\downarrow (e.md\ f))e' \wedge e'.PC = e.PC + 1 \wedge e \equiv e'[destino, (md\ ST\ 2), PC]$
decfsz f el	$(el \rightarrow (e'.md\ f) = \uparrow(\downarrow (e.md\ f) - 1)) \wedge (\neg el \rightarrow e'.W = \uparrow(\downarrow (e.md\ f) - 1)) \wedge$ $(\uparrow(\downarrow (e.md\ f) - 1) \neq 0 \rightarrow e'.PC = e.PC + 2) \wedge$ $(\uparrow(\downarrow (e.md\ f) - 1) = 1 \rightarrow e'.PC = e.PC + 1) \wedge e \equiv e'[destino, PC]$
clrf f	$(e'.md\ f) = \uparrow 0 \wedge \text{ZERO}\, 0\, e' \wedge e'.PC = e.PC + 1 \wedge e \equiv e'[(md\ f), (md\ ST\ 2), PC]$
bcf f p	$(e'.md\ f\ p) = 0 \wedge e'.PC = e.PC + 1 \wedge e \equiv e'[(e'.md\ f\ p), PC]$
btfss f p	$((e.md\ f\ p) = 1 \rightarrow e'.PC = e.PC + 2) \wedge ((e.md\ f\ p) = 0 \rightarrow e'.PC = e.PC + 1) \wedge e \equiv e'[PC]$
goto k	$e'.PC = k \wedge e \equiv e'[PC]$
call k	$e'.s = (push\ e.PC + 1\ e.s) \wedge e'.PC = k \wedge 0 \leq \lvert e'.s \rvert \leq 8 \wedge e \equiv e'[PC, s]$
return	$e'.PC = (top\ e.s) \wedge e'.s = (pop\ e.s) \wedge 0 \leq \lvert e'.s \rvert \leq 8 \wedge e \equiv e'[PC, s]$

5 Semántica de la Máquina de Bajo Nivel

En esta sección definimos una semántica operacional de la máquina. Llamaremos Máquina de Bajo Nivel (MBN) a la formalización en base a esta semántica, que nos provee un modelo ejecutable del sistema y describe el comportamiento del PIC. En esta máquina la ejecución de una instrucción está representada por una relación funcional que vincula un estado `e`, una instrucción `i` y un estado `e'`, si `e'` es el estado resultante de la ejecución de la instrucción `i` a partir del estado `e`, que notaremos $(e, i) \Rightarrow e'$.

El estado inicial es el estado en que se encuentra el PIC al conectarlo a la alimentación. Un programa es una secuencia de instrucciones. La ejecución de un programa genera una secuencia de estados, donde cada uno de éstos, salvo el estado inicial, se obtiene a partir del estado anterior ejecutando el paso semántico de la

instrucción correspondiente. La semántica operacional, para cada instrucción, se presenta a continuación.

addlw l

$$\frac{\downarrow e.W = x, \downarrow l = y, f2 \leftrightarrow x + y = 0, f0 \leftrightarrow x + y > 255}{(e,\ addlw\ l) \Rightarrow \langle (actF\ e.md\ (f2\ f0)),\ \uparrow (x + y),\ e.PC + 1,\ e.s \rangle} \quad (1)$$

addwf f el

$$\frac{el = true, \downarrow e.W = x, \downarrow (e.md\ f) = y, f2 \leftrightarrow x + y = 0, f0 \leftrightarrow x + y > 255}{(e,\ addwf\ f\ el) \Rightarrow \langle (actF\ (updReg\ e.md\ f\ \uparrow (x + y))\ (f2\ f0)),\ e.W,\ e.PC + 1,\ e.s \rangle} \quad (2)$$

$$\frac{el = false, \downarrow e.W = x, \downarrow (e.md\ f) = y, f2 \leftrightarrow x + y = 0, f0 \leftrightarrow x + y > 255}{(e, addwf\ f\ el) \Rightarrow \langle (actF\ e.md\ (f2\ f0)),\ \uparrow (x + y),\ e.PC + 1,\ e.s \rangle} \quad (3)$$

movf f el

$$\frac{el = true, f2 \leftrightarrow \downarrow (e.md\ f) = 0}{(e,\ movf\ f\ el) \Rightarrow \langle (actF\ e.md\ (f2\ \_)), e.W,\ e.PC + 1,\ e.s \rangle} \quad (4)$$

$$\frac{el = false, f2 \leftrightarrow \downarrow (e.md\ f) = 0}{(e,\ movf\ f\ el) \Rightarrow \langle (actF\ e.md\ (f2\ \_)),\ e.md\ f,\ e.PC + 1,\ e.s \rangle} \quad (5)$$

decfsz f el

$$\frac{el = true, \downarrow (e.md\ f) = 1}{(e, decfsz f\ e1) \Rightarrow \langle (updReg\ e.md\ f\ \uparrow 0),\ e.W,\ e.PC + 2,\ e.s \rangle} \quad (6)$$

$$\frac{el = true, \downarrow (e.md\ f) = x,\ x \neq 1}{(e, decfsz f\ e1) \Rightarrow \langle (updReg\ e.md\ f\ \uparrow (x - 1)),\ e.W,\ e.PC + 1,\ e.s \rangle} \quad (7)$$

$$\frac{el = false, \downarrow (e.md\ f) = 1}{(e, decfsz\ f\ e1) \Rightarrow \langle e.md, \uparrow 0,\ e.PC + 2,\ e.s \rangle} \quad (8)$$

$$\frac{el = false, \downarrow (e.md\ f) = x,\ x \neq 1}{(e, decfsz f\ e1) \Rightarrow \langle e.md, \uparrow (x - 1),\ e.PC + 1,\ e.s \rangle} \quad (9)$$

clrf f

$$\frac{f2 \leftrightarrow \downarrow (e.md\ f) = 0}{(e, clrf\ f) \Rightarrow \langle (actF\ (updReg\ e.md\ f\ \uparrow 0)\ (f2\ \_)),\ e.W,\ e.PC + 1,\ e.s \rangle} \quad (10)$$

bcf f p

$$\frac{}{(e, \text{bcf } f\ p) \Rightarrow \langle (\text{updBit e.md } f\ p\ 0),\ e.W,\ e.PC+1,\ e.s\rangle} \quad (11)$$

btfss f p

$$\frac{(e.md\ f\ p)=1}{(e, \text{btfss } f\ p) \Rightarrow \langle e.md,\ e.W,\ e.PC+2,\ e.s\rangle} \quad (12)$$

$$\frac{(e.md\ f\ p)\neq 1}{(e, \text{btfss } f\ p) \Rightarrow \langle e.md,\ e.W,\ e.PC+1,\ e.s\rangle} \quad (13)$$

goto k

$$\frac{}{(e, \text{goto } k) \Rightarrow \langle e.md,\ e.W,\ k,\ e.s\rangle} \quad (14)$$

call k

$$\frac{}{(e,\ \text{call } k) \Rightarrow \langle\ e.md,\ e.W,\ k,\ \text{push e.s } e.PC+1\rangle} \quad (15)$$

return

$$\frac{}{(e,\ \text{return}) \Rightarrow \langle e.md,\ e.W,\ \text{top e.s},\ \text{pop e.s}\rangle} \quad (16)$$

6 Análisis de Propiedades

Todas las formalizaciones presentadas en este trabajo están escritas en el Cálculo de Construcciones Inductivas, usando Coq [4,5]. En particular, empleando el asistente de pruebas Coq hemos demostrado que MBN es una implementación MAN. Esto es:

Teorema ***Correctness****: ∀e, e': Estado ∀i: Instrucción,* (17)
Precondición(i, e) $\wedge$ *(e, i)* $\Rightarrow$ *e'* $\rightarrow$ *Postcondición(i, e, e')*

Donde, *Precondición* y *Postcondición* son las familias de precondiciones y postcondiciones de las instrucciones, respectivamente, detalladas en la sección 4.

La prueba del teorema *Correctness* se realizó por inducción sobre el conjunto de las instrucciones. Asimismo, se demostraron propiedades de seguridad relacionadas con las instrucciones que operan sobre la pila de ejecuciones. Por razones de espacio omitimos un análisis detallado de las propiedades analizadas y de sus pruebas.

Un aspecto importante a destacar del asistente Coq es que a partir de (17), y en general de pruebas de teoremas de este tipo, es posible extraer un prototipo certificado en un lenguaje funcional de alto nivel, como Haskell o ML [4,5].

7 Conclusiones y Trabajos Futuros

Los avances tecnológicos de las últimas décadas han permitido que los microcontroladores se usen hoy en diferentes ámbitos; por ejemplo, en dispositivos de comunicación como los celulares o de control médico como los marcapasos. En este trabajo analizamos una familia de microcontroladores programables llamada PIC, en particular el PIC 16F84. El microprocesador cuenta con un programa modificable, que lo convierte, esencialmente, en una computadora de uso general, pequeña, barata, y versátil. Nuestro interés se centra en la verificación de programas críticos desarrollados en este entorno. En particular, nuestro objetivo final es el estudio de programas para dispositivos biomédicos implantables. No obstante en este artículo damos un paso previo imprescindible en dicha dirección.

El aporte de nuestro trabajo reside en la formalización del PIC considerado, a través de una máquina de estados. Especificamos el comportamiento de las instrucciones de la máquina y certificamos, usando Coq, el funcionamiento de la misma, expresado a través de una semántica operacional. Por razones de espacio debimos omitir muchos detalles; algunas instrucciones del PIC, y en particular las formalizaciones y pruebas realizadas en Coq. Hasta nuestro conocimiento, no existen formalizaciones alternativas del PIC 16F84, que es usado por ejemplo por la empresa CCC S.A. (Uruguay) para la programación de marcapasos.

Como trabajos futuros nos proponemos representar y probar propiedades de programas para el PIC considerado, y enriquecer el sistema con la incorporación de relojes y un conjunto completo de instrucciones. Asimismo, nos planteamos desarrollar un framework para verificar propiedades de seguridad sobre la clase de sistemas críticos que se embarcan en el PIC. Para esto planeamos usar un asistente de pruebas como Coq y herramientas de verificación de modelos, como Kronos [6], siguiendo por ejemplo la metodología propuesta por uno de los autores de este trabajo en [7] y conceptos de [8]. En particular, estamos interesados en mostrar la viabilidad de probar formalmente la seguridad de ciertos programas usados en marcapasos, analizando propiedades tanto extensionales como intensionales.

Referencias

[1] Stallings, W. Computer Organization and Architecture. Prentice Hall (1999)
[2] Palacios, E. et al.: Microcontrolador PIC16F84. Desarrollo de proyectos. Alfaomega (2004)
[3] Data Sheet. Microchip Technology Inc. (1998), http://ww1.microchip.com/ downloads/en/DeviceDoc/30430c.pdf, último acceso: Mayo de 2008.
[4] The Coq Development Team: The Coq Proof Assistant Reference Manual, Version 8.1. (2006). Materiales disponibles en http://coq.inria.fr/, último acceso: Mayo de 2008.
[5] Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science. Springer-Verlag (2004)
[6] Yovine, S. Kronos: A verification tool for real-time systems. International Journal of Software Tools for Technology Transfer, Vol. 1, Issue 1/2, pages 123-133, October 1997.
[7] Luna, C. Especificación y análisis de sistemas de tiempo real en teoría de tipos. Vol 3 No. 1, Revista Iberoamericana Computación y Sistemas, CIC, IPN, México (2004)
[8] Betarte, G. et al. Specification of a Smart Card Operating System. In Proceedings of the 1999 Workshop on Types for Proofs and Programs, LNCS 1956 (2000)