Córdoba 17 de Mayo de 2022

PDC32

Bienvenidos a mi mundo, te cuento rápidamente que soy.

Bien, soy una simple computadora hecha en una arquitectura propia en 32bits, en tecnología TTL.
Soy una evolución de la PDC3 y a su vez, de la PDC2, ambas de 8bits.
Sigo, puedo hacer cálculos matemáticos, leer un teclado del tipo AT, enviar y recibir datos a través del puerto serial COM.
Mi memoria principal es de hasta 32Mb, y además, me ayuda una pequeña memoria caché de 128Kb, para almacenar mis variables y arrays de sistema.
Puedo también emitir sonidos simples (monofónico) a través de un pequeño altavoz, para dar avisos de eventos, o bien, reproducir una simple melodía.
Poseo un controlador de tarjetas extraíbles de almacenamiento de hasta 256Kb de capacidad.
Incorporo además, un reloj calendario(RTC)
Y por último, puedo manejar unos 18,432,000 píxeles por segundo, en una pantalla VGA, es lo único rápido que puedo hacer, ya que mi procesador sólo puede trabajar hasta 4Mhz.
En modo texto, mis caracteres se pueden desplegar de hasta 256 colores posibles.
Y como detalle, puedo generar un cursor que parpadea por hardware.
Tengo un modesto sistema operativo nativo, que alcanza, para el manejo de archivos de un sólo directorio.
Cuento con una BIOS, que puedes configurar a través de una interfaz algo colorida!

CONTENIDO

Capítulo 1.

ARQUITECTURA

DESCRIPCIÓN

Dicha arquitectura, posee un BUS de rango de 32bits.
El firmware se ejecuta desde 3 memorias EPROM.
Posee una ALU como todo sistema informático, y se encarga del manejo de cálculos, operadores de bits, desplazamientos de datos en registros y por último, genera los saltos de programa correspondiente.
Cada función del sistema se ha destinado a tarjetas individuales, de esa manera se puede testear y corregir, por separado durante el desarrollo.
Una función vital en el sistema, es la incorporación de TIMERs por hardware. El mismo, se encarga de generar los retardos necesarios, liberando así al sistema para realizar otras tareas.
Otra mejora añadida (sugerida por el prof. Nicolás Wolovick) es la posibilidad de retornar a una línea de programa luego de un salto, de manera automática.
Es decir, si el programa se encontraba en la línea 43, luego de producirse una llamada, pues bien, cuándo retorne, lo hará a la línea 44 para continuar.

CONSTRUCCIÓN DEL HARDWARE

¿Cómo nace la PDC?
Bien, todo comienza desde una idea. Luego esa idea se transfiere a papel, y luego de bastante empeño, pero nunca exento de equivocaciones, se puede plasmar en una tarjeta de cobre, para darle vida!
Pero hay que destacar algo, todo el diseño se aplica bajo la filosofía de; TODO DEBE SER DESDE CERO!
Por lo tanto el desarrollo se realiza en lógica discreta.
Aquí no puede haber ningún microprocesador o controlador afín.
Hay que "arreglárselas" con componentes que pueden conseguirse en tiendas o en reciclaje.
Todos los PCBs se realizaron a mano, con la ayuda de una escuadra de dibujo y una aguja de insulina cortada a medida inyectando tinta indeleble.
Es un trabajo laborioso, ya que debe lograrse simetría en las distancias entre pistas, además de organizar los circuitos integrados en zonas, dónde resulte cómodo la interconexión.
Las tarjetas se insertan a la placa principal, a través de conectores simples o duales del tipo IDC.

Algunas placas poseen a su vez, otras placas conectadas a ella, formando un apilado.
Bien, volviendo al diseño del PCB, una vez terminado el ruteado de pistas, se verifica minuciosamente posibles cortocircuitos entre pistas o pistas discontinuadas.
Las zonas no utilizadas se enmascaran con tinta para formar planos de GND, con el fin de mejorar el rechazo a interferencias, además de prolongar la vida útil del percloruro férrico.

Hasta aquí todo bien, pero... ¿por dónde empezamos?
Para lograr el mínimo funcionamiento del sistema, se debe contar con la memoria del firmware, el decodificador de instrucciones, la ALU, algo de memoria RAM y algún puerto de salida de cualquier tipo, para "ver" los resultados.
Conseguido ésto, se procedió a realizar el primer programa, que consistía en un simple LED destellando.
El programa contaba una variable y al desbordarse, cambiaba de estado un bit, reflejándose en dicho LED.
Éste trabajo invirtió un tiempo de aprox. cuatro meses, y hasta ese momento era trabajar a "ciegas" ya que no había forma de probar el funcionamiento, hasta juntar las piezas necesarias.

DISEÑO DEL SOFTWARE

Bueno, el software se desarrolló en un compilador muy simple de la PDC3.
Sin su ayuda, todo hubiera sido muy tedioso, ya que la única manera de crearlo, hubiese sido escribiendo ceros y unos.
Pero en cambio, la PDC3, permite tipear el código en un lenguaje, un poco más amigable, usando caracteres alfanuméricos.
Por ejem. Si tipeamos la sig. línea:

A12 15
El compilador lo traduce a :11101100 00001111, y podemos darnos cuenta de la ayuda que representa!
Bien, el software sigue la misma ideología que el HARDWARE, ya que se crearon tantas rutinas, como tarjetas hay en el sistema.
De esa forma cada rutina, trabaja de forma individual, y se comunican unas con otras a través de variables específicas, ya sea por referencia o por valor, como lo hace una función en C.

Para organizar el software, existe una rutina principal que se ejecuta en un bucle infinito, y a partir de ahí, se realizan llamados a cada rutina según el evento.
Por ejem. Si pulsamos la tecla “A”, existe una rutina que lee el registro principal llamado REG STATE. A él, concurren todas las señales que indican un cambio o evento.
Bien, volviendo a la tecla pulsada, cuándo la rutina detecta dicha tecla, le indica a otra rutina el evento ocurrido, entonces la nueva rutina advertida, se encarga de decodificar la tecla pulsada y la almacena en un buffer cíclico, de 64 elementos.
Entonces, cuándo se llama a la rutina que lee dicho buffer, libera el espacio del mismo y se encarga de realizar la tarea que se le asignó, que puede ser, por ejem. enviar ese carácter identificado a la tarjeta de video, llevarlo al puerto serial, o simplemente almacenarlo en RAM, etc.
Cada rutina fue desarrollada en función de la necesidad del momento, y cabe recalcar que hay rutinas que necesitan de otra rutina “obligada” para poder funcionar.
Por ejem. una rutina llamada REFRESH, se encarga de intercambiar las dos páginas de video, es decir, una página que está visualizándose, pasa a modo escritura y la que estaba escribiéndose, ahora se muestra en pantalla. Entonces si no hay ninguna rutina que haya actualizado ningún dato en la tarjeta VGA, la rutina REFRESH no advertiríamos alguna diferencia!

COMPATIBILIDAD

Durante el diseño de la PDC, se planteó la idea de utilizar periféricos estándar para facilitar el intercambio de datos.
Entonces, ésta necesidad generaba la simpleza, al no tener que desarrollar periféricos exclusivos, pero a cambio de eso, había que ajustarse al requerimiento de lo estándar.

Empezando por el teclado, había que estudiar el funcionamiento íntegro de la comunicación, que se trata de un protocolo síncrono de dos señales, una es el CLOCK y la otra el DATO que es bidireccional.
Cómo anécdota, la tarjeta de teclado, hubo que realizarla por segunda vez, ya que por descuido al no considerar un simple "detalle" , el teclado NO respondía a los datos enviados. Sólo podía recibirse lo que el teclado enviaba. En principio sonaba algo reconfortante, pero NO era lo que se esperaba!
En fin, luego de realizar de nuevo la tarjeta de teclado, pasamos al diseño del controlador del puerto serial COM.
El cuál, permite una comunicación bidireccional full dúplex, y puede ser configurado desde la BIOS, para funcionar en diferentes velocidades y tamaño (en bits) de cada elemento, como así también si la comunicación se realiza con o sin paridad.

Continuamos con el desarrollo de la tarjeta de video(VGA), sin dudas es la más compleja, pero no sólo por la cantidad de componentes que utiliza, sino por la velocidad a la que trabaja.
Requiere una señal de al menos 25Mhz, para poder alcanzar el formato estándar de 640x480 con un refresco de 60Hz, en barrido de modo progresivo.
Éste requerimiento generó demoras para conseguir las vitales memorias de alta velocidad, que milagrosamente fueron recicladas, de un viejo router.
Además, se utilizaron contadores síncronos del tipo 74LS193, honestamente NO eran los idóneos, para apenas alcanzaban para lograr ese cometido.
Ésta tarjeta requirió, en algunas partes de un simulador como PROTEUS, ya que no había manera de comprobar el comportamiento que tendría, y no podíamos entrar en la idea de prueba y error hasta lograr el funcionamiento, ya que eso acabaría con cualquier grado de paciencia.
Luego, empecé por la carga de los 256 caracteres en una memoria EPROM.
Pixel a pixel se fueron tipeando, en un lapso de una semana aprox. Invirtiendo unas dos horas diarias.
Cómo detalle, hay un caracter, más precisamente el número 255, el cuál se personalizó en una especie de apilado de letras denotadas como B, C, G y W. Ésto fue en honor a las personas que colaboraron de diferentes maneras, proveyendo el hardware tipo scrap entre otras cosas. Además de estar presentes desde el día que fue concebida la idea de armar la PDC32.

Continuando con la sig. etapa, el turno era para: La controladora de memoria extraíble para crear las tarjetas de intercambio de información.
Se utilizó la versión 24LCXXX, que funciona bajo el protocolo I2C, si bien NO es la opción más rápida, pero dado que la densidad de datos que almacena es baja, eso compensa en cierta manera su lentitud.

Por último, se realizó el diseño de la tarjeta controladora de memoria principal. La cuál utiliza la tecnología DRAM, en formato SIMM de 72 pines.
Si bien soporta varias capacidades, la finalidad era poder trabajar con 32Mb, que es más que suficiente!

Bien, concluído el trabajo de periféricos, arrancamos con el desarrollo de cada rutina, según la necesidad del flamante HARDWARE recién implementado esperando funcionar!

Capítulo 2.

CREANDO RUTINAS

Veamos, una vez terminada cada tarjeta del sistema, había que realizar una rutina para poder comprobar su funcionamiento.
Ya que si instalamos la tarjeta en su conector correspondiente, al encender la máquina, NO habría ninguna respuesta de dicha tarjeta.
Por lo tanto, el sistema la ignoraría, ya que como cualquier sistema informático, se requiere de un controlador de hardware, para lograr su funcionamiento.
Dicho ésto, empezamos por lo básico.
Desarrollar rutinas simples, que pueden o no, ser utilizadas por otras rutinas más complejas.
Ésto representa dos ventajas, la primera es que nos ahorra la tan reducida memoria de firmware disponible y la segunda ventaja es la abstracción del funcionamiento de las señales de cada tarjeta. Y ponemos un ejem.
Supongamos que queremos imprimir un caracter en pantalla.
Pues bien, primero hay que verificar que el barrido de la pantalla haya llegado al final para realizar el intercambio de páginas, ésta información la obtenemos de dos bits del REG STATE.
Luego hay que activar, el modo de funcionamiento, ya sea texto o gráfico, luego habilitar las señales de posición en dónde se escribirá un carácter. Luego sincronizar para volver a la página de video principal para realizar nuevamente el intercambio.
Si analizamos todo ésto, se vuelve tedioso si tuviéramos que realizarlo, cada vez que debamos imprimir un simple caracter.
Ahora si delegamos a una rutina, a que haga todo ese trabajo, nos desentendemos completamente del tema señales y demás!
Y eso se resume a: Escribir el caracter en un registro de memoria, luego escribir en otro registro, la ubicación X e Y de la pantalla en dónde se mostrará y por último llamar a la rutina que mencionamos antes.
Y ella "solita" tomará el dato de la memoria en dónde lo habíamos depositado, y se encargará de realizar toda esa tarea de activar todas las señales necesarias y poner a trabajar a la tarjeta para que al final lo muestre en pantalla.
¿Nos facilita la vida verdad?

RUTINAS BÁSICAS

Una rutina básica, utiliza pocos recursos, pero resuelve por sí misma una tarea específica. Listamos a continuación las principales rutinas básicas.

***BOX:**

Ésta rutina, es del tipo gráfica, y se encarga de armar un marco en pantalla, con un área seleccionable, como así también su color, sombreado y formato de marco.

***KBD:**

La misma, lee el bit de interrupción, que indica una tecla pulsada o soltada, y almacena dicha tecla en un buffer cíclico.

***TXT**:

És sin dudas, la más importante, ya que intuitivamente por su descripción, indica que maneja texto, y si!, se encarga de imprimir texto en pantalla, leyendo un buffer y transfiriéndolo a la tarjeta de video en pocos pasos.

***UART:**

Su función es similar a la rutina KBD, pues de igual manera lee el puerto COM y almacena el dato recibido en un buffer específico, destinado para ella.

***SOUND:**

Se encarga de gestionar el altavoz, sólo se requiere de dos datos, la frecuencia del TONO y la duración del mismo.

***TMR2:**

Ésta rutina, maneja uno de los temporizadores asignado a ella,.
Necesita como dato el período de espera y una dirección de retorno.
Lo que hace es leer permanentemente un bit de desborde, indicando que el tiempo ha transcurrido, acto seguido, saltará a la dirección indicada.

***BIN_BCD**:

La misma, tiene una función importante para la ALU, ya que convierte cualquier número binario a BCD.
Es muy útil para mostrar resultados en pantalla, entre otros.

***CONFIG_KBD y CONFIG_COM:**

Ambas rutinas, funcionan sólo en el inicio del sistema, y se encargan de leer los registros específicos de la memoria EEPROM interna, y luego configuran la tarjeta de teclado y la tarjeta UART, según opciones elegidas en la BIOS.

***DEC_KEY:**

Su función es, decodificar la tecla pulsada y asignarle un caracter ASCII.
Puede trabajar con combinación de teclas, por ejem. si pulsamos SHIFT+a, nos devuelve el caracter A, de igual manera detecta si CAPS BLOQ, está activada.

(*) Existen más rutinas que no se han detallado.

RUTINAS COMPUESTAS

La diferencia de éstas rutinas con las básicas, es que utilizan una o más rutinas para poder funcionar.
Entre ellas tenemos:

***RTC**

Ésta rutina, lee la tarjeta del reloj calendario, luego almacena cada dato en memoria RAM y se los transfiere a la rutina TXT, para imprimir en pantalla.

***BCD_ASCII**:

Ésta rutina se apoya a la rutina BIN_BCD. Simplemente, imprime en pantalla los caracteres convertidos, en una secuencia de números. Y se encarga de detectar el final de la cadena. Para ello, requiere de la rutina TXT.

***TURN_OFF:**

Se encarga de apagar al sistema, gobernando los bits que controlan el encendido y apagado de la fuente ATX.
Necesita de la rutina TXT, para mostrar las acciones que está realizando.

***CURSOR:**

Ésta rutina, funciona de diferentes maneras, según sea llamada desde otra función, pero su meta es mostrar en pantalla un caracter como símbolo de cursor, que puede seleccionarse entre, uno de dos posibles formatos, y que además puede parpadear o no, según se configure.

(*) Existen más rutinas que no se han detallado.

Capítulo 3.

SET DE INSTRUCCIONES

El hardware de la PDC, puede manejar hasta 64 microinstrucciones, de las cuales, sólo se incorporaron 48.
Cada microinstrucción, se ejecuta en un ciclo de máquina y requiere un tiempo de:

T = 4 / (Frecuencia reloj)

Dichas microinstrucciones o instrucciones simples, se han dividido en tres grupos.
Y se identifican con las letras A, B y C respectivamente.
A su vez, a cada una de las instrucciones se les asigna un número.
Vale decir que, para identificar una instrucción específica, necesitamos nombrarla con la letra del grupo y su número asignado.
Por ejem.

A12

En la sig. figura observamos una lista completa, de todas las instrucciones incluídas en el sistema.

```
╞========================= GRUPO A ============================
A0 REG DATA DRAM
A1 REG Carry IN  (bit 3)
A2 RETURN
A3 A<=B
A4 A>=B
A5 GOTO sin condición
A6 Ck STEP_UP ADDR DRAM
A7 REG ADDR DRAM
A8 A!=B
A9 REG (A) ALU
A10 REG Carga bit16-bit31 en parte alta
A11 WRITE DRAM
A12 REG Sel 3-S
A13 REG Return Automatico (bit 15)
A14 REG (B) ALU
A15 REG Comparador ALU (bit 0, 1, 2)
```

```
========================== GRUPO B ============================
B0 REG Sel OFV Timer/Speaker
B1 REG Sel OFV UART
B2 REG Config UART
B3 REG DATA Tx UART
B4 - Libre Slot Trasero
B5 REG DATA Tx KBD
B6 DATA/ADDRESS RTC DS1387
B7 CONTROL COMMAND RTC DS1387
B8 A<B
B9 Sel Funct TMR/SPK (bit 4, 5)
B10 A>B
B11 A=B
B12 REG DATA Caché
B13 REG ADDR Caché
B14 REG ON/OFF ATX PWR (bit 11)
B15 WRITE Caché
```

```
============================ GRUPO C ============================
C0 SEL TMR
C1 SEL TIME(timer)
C2 REG DATA DRIVE SERIAL(32bits)
C3 REG ADDR DRIVE SERIAL(19bits OUT ADDR EEP, 10bits IN ADDR
EEP)
C4 SEL FUNCT DRIVE SERIAL(b0-b2) 1=RD IN EEP, 3=WR IN EEP, 4=RD
OUT EEP, 6=WR OUT EEP
C5 -
C6 -
C7  SEL COLOR TEXT
       (b16-b23) 1er. Plano
       (b24-b31) Fondo
C8 WRITE VRAM VGA
C9 SEL FUNC. VGA (b0-b1)
      00=Mode Graphic
      01=Transfer Data at Buffer
      10= Transfer Text
      11= Mode Text
C10 REG BLINK TEXT (b12=1 ON)
C11 REG DATA COLOR PIXEL VGA (b0-b7)
C12 WRITE CMD TEXT
C13 REG SEL ASCII RAM TEXT (b0-b7)
C14 REG ADDR XY PIXEL VGA (b0-b18)
C15 REG LOCATE XY TEXT(b0-b11)
```

Las microinstrucciones, consisten en señales generadas por el decodificador de instrucciones, y se encargan de activar las diferentes secciones de hardware, sin ninguna intervención de por medio.

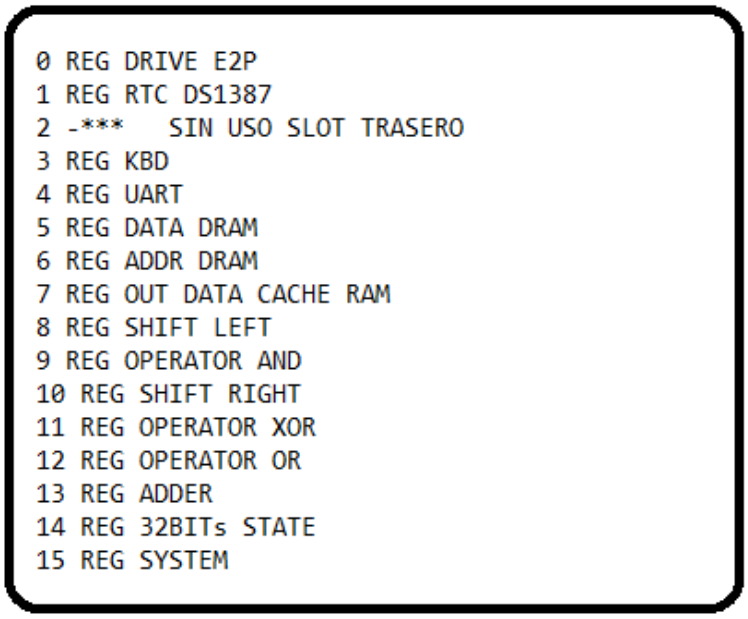
Cada instrucción, va acompañada de un argumento, que puede ser un dato del tamaño de 1 bit hasta 32 bits.
Hay excepciones en los argumentos, por ejem. las instrucciones A6, A11, B15, C8 y C12, no requieren argumento.

Capítulo 4.

REGISTROS

Existen dos tipos de registros, de naturaleza de soft o hard.

En el caso de registros de soft, se trata de una locación en la memoria RAM, que puede ser tanto la memoria caché, cómo la principal.
En cambio, en un registro por hard, hablamos de un FLIP-FLOP concretamente, que puede ser del tipo LATCH-D, RS, JK, etc. y/o también registros del tipo 3S o tercer estado.
Éste último tipo de registros, son muy importantes, ya que devuelven un valor al BUS principal, para ser procesado a posterior; Como por ejem, el resultado de una comparación o una suma.
Cada tarjeta puede poseer uno o más registros de datos, además puede manejar desde 1 bit hasta 32 bits.
Para seleccionar un registro físico, se activan SOLAMENTE de a uno por vez, ya que dos o más registros activados al mismo tiempo, provocarían un cortocircuito en las líneas del BUS principal.
Para evitar ésto, se utilizó un decodificador de direcciones como el 74LS138.
Más precisamente, se requieren dos de éstos, ya que existen 16 registros en toda la PDC.
La decodificación, es gobernada por una instrucción específica y es la A12.
Para aclarar, si deseamos direccionar el registro 15, simplemente debe escribirse el código A12 15.
E inmediatamente, luego de ser decodificada dicha instrucción, será direccionado el registro 15, y permanecerá seleccionado, hasta tanto NO se seleccione otro.
Los registros utilizados en el sistema, son de tipo TTL de alta corriente de salida, como lo es el 74LS245.
A continuación vemos la tabla completa de registros físicos.



El registro 14, es el encargado de recolectar todas las señales de todo el sistema, como antes mencionamos.
Más abajo se detalla la función de cada bit.

INPUT:	OUTPUT:
Bit 0: A>B	Bit 0: A>B
Bit 1: A=B	Bit 1: A=B
Bit 2: A<B	Bit 2: A<B
Bit 3: CARRY IN ADD	Bit 3: CARRY OUT ADD
Bit 4: ENABLE TMR=0	Bit 4: OVERFLOW TMR=1
Bit 5: SEL SPK=0, TMR=1	Bit 5: Tx UART=1
Bit 6: -	Bit 6: Rx UART=1
Bit 7: -	Bit 7: Rx KBD=1
Bit 8: -	Bit 8: TX KBD=1
Bit 9: -	Bit 9: SLOT TRASERO
Bit 10: -	Bit 10: SLOT TRASERO
Bit 11: TURN-ON=1	Bit 11: TURN-OFF=0 (BOTON PRESIONADO)
Bit 12: BLINK TXT ON=1	Bit 12 y 13: RETURN STATE VGA
Bit 13:	0= MODE GRAPH
Bit 14:	1= REC BUFFER
Bit 15: RETURN=1	2= MODE TEXT
	3= MODE LOAD TEXT
	Bit 14: Busy DRIVE E2P SERIAL=1
	Bit 15: Busy TMRs SELECT=1
	Bit 16-
	Bit 17-
	Bit 18-
	Bit 19-
	Bit 20-
	Bit 21-
	Bit 22-
	Bit 23-
	Bit 24 al Bit 31 OVERFLOW=1 TMR0-TMR7

Las señales que se identifican en el grupo INPUT, se desplazan desde el BUS hacia las tarjetas del sistema.
En cambio las señales que pertenecen al grupo OUTPUT, son las entregadas por el registro 14 al BUS principal.

Capítulo 5.

DIAGRAMA DE LA PDC32

Cabe aclarar que existen dos tipos de tarjetas en el sistema.
Pueden ser del tipo COMBINACIONAL o SECUENCIAL.
En el primer caso, los datos obtenidos se generan a partir de datos inyectados desde el BUS, y se obtienen a través del REGISTRO de salida correspondiente a esa tarjeta.
Por ejem. el comparador de la ALU, el SUMADOR, memoria caché, etc.
Por otro lado, en las tarjetas del tipo secuencial, poseen un reloj propio que las hace trabajar a una frecuencia independiente a la del sistema, pero para lograr la sincronización de los datos obtenidos, se requiere de un bit que indica que el resultado está listo, y dicho dato se obtiene del registro REG STATE.
Algunos ejemplos de tarjetas secuenciales son:
Todas las tarjetas controladoras, Timers, etc.

Bien, la Arquitectura del sistema se compone de los sig. bloques que se detallan.

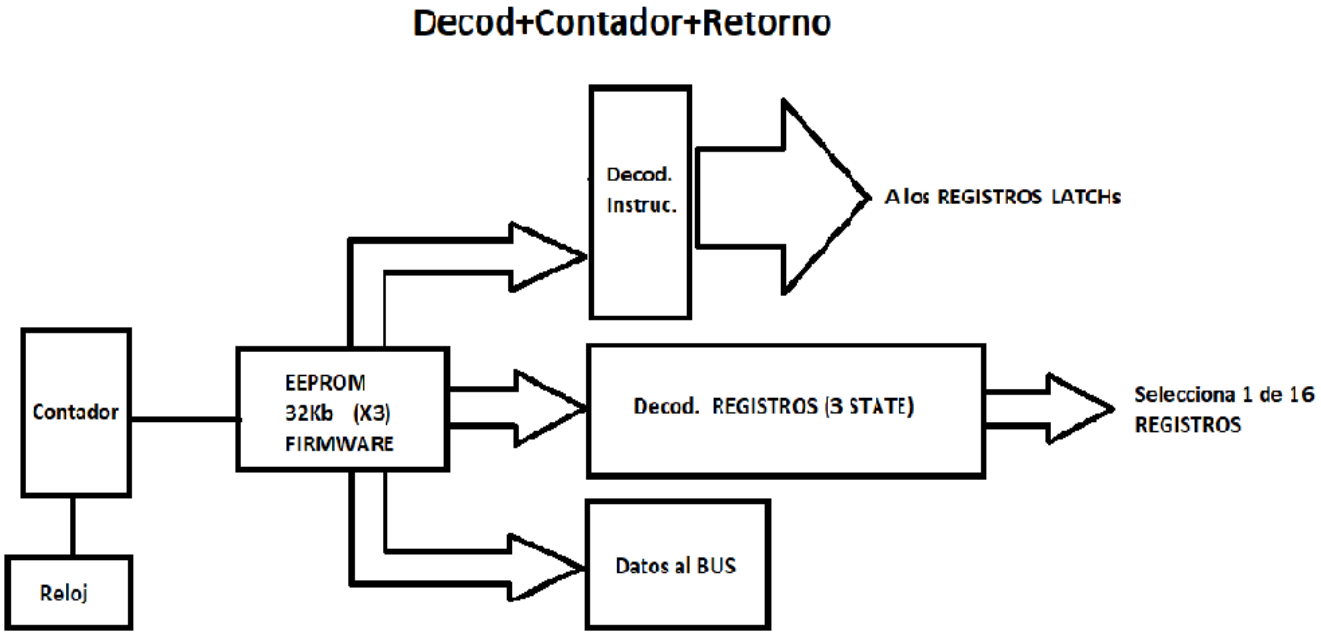
*Reloj de sistema
*Contador de programa
*Decodificador de instrucciones y registros
*ALU
*Memoria Caché
*Memoria principal
*Timers del sistema
*Controlador de Keyboard
*Controlador de puerto COM
*Controlador de Memory CARD
*Controlador de video(VGA)
*Controlador de altavoz
*Controlador de fuente ATX
*Controlador de reloj RTC

RELOJ DE SISTEMA

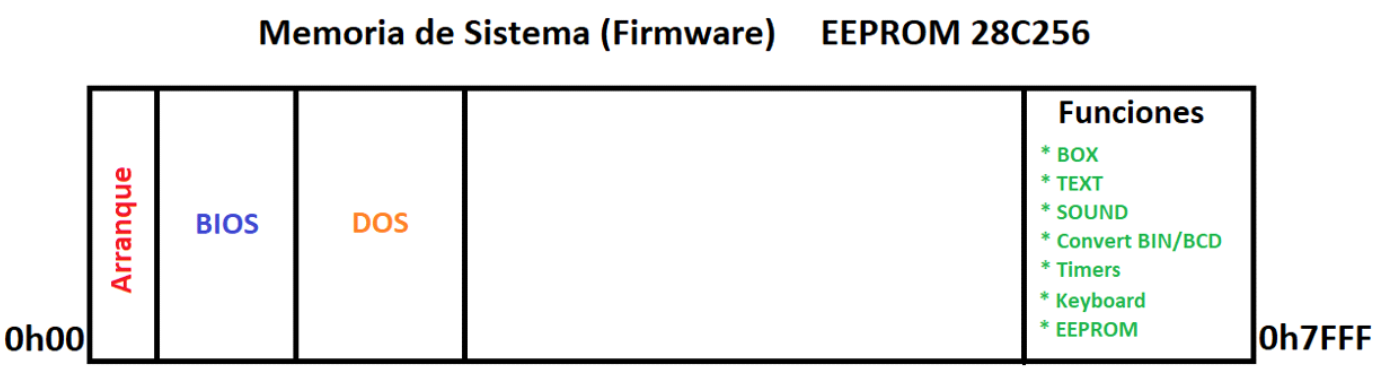
Posee un circuito integrado con un oscilador de cuarzo a 16Mhz, luego la señal producida, es dividida por dos etapas. En la primer etapa, la señal se direcciona a la tarjeta de memoria DRAM, y se utiliza para sincronizar los datos de entrada/salida, y a su vez, se encarga de generar el REFRESCO requerido por éste tipo de tecnología de memoria.
La segunda etapa, genera una señal que se inyecta al contador de programa, en éstas circunstancias la señal tiene una frecuencia de 4Mhz.

CONTADOR DE PROGRAMA

Ésta sección es el corazón del sistema, ya que tiene la función de direccionar e incrementar las direcciones de memoria del firmware del sistema.
A continuación, el diagrama en bloques.



Aquí puede observarse, como se distribuyeron las rutinas, dentro de la memoria del firmware.

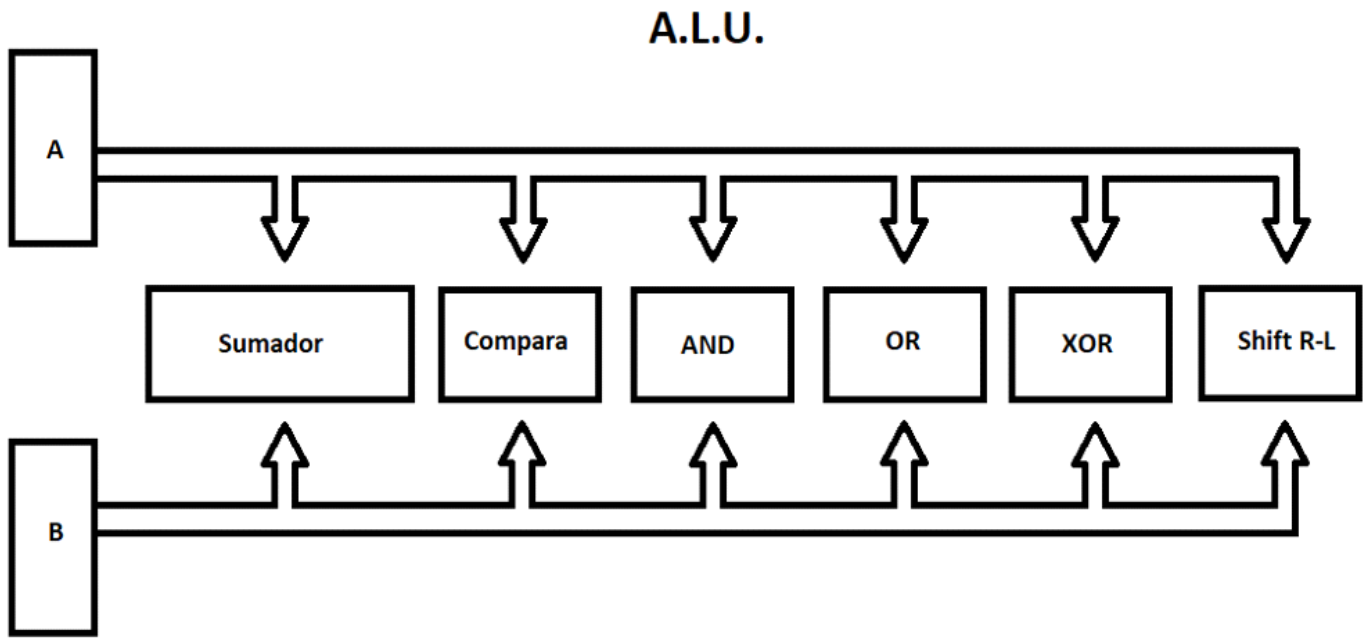


DECODIFICADOR DE INSTRUCCIONES Y REGISTROS

Aquí se decodifican, como su nombre lo indica, las instrucciones tomadas de las memorias que contienen el firmware del sistema.
Y paralelamente a ello, hay un decodificador, que se pone en marcha cuándo el programa comienza a ejecutarse, para direccionar los datos, según la lógica del programa.

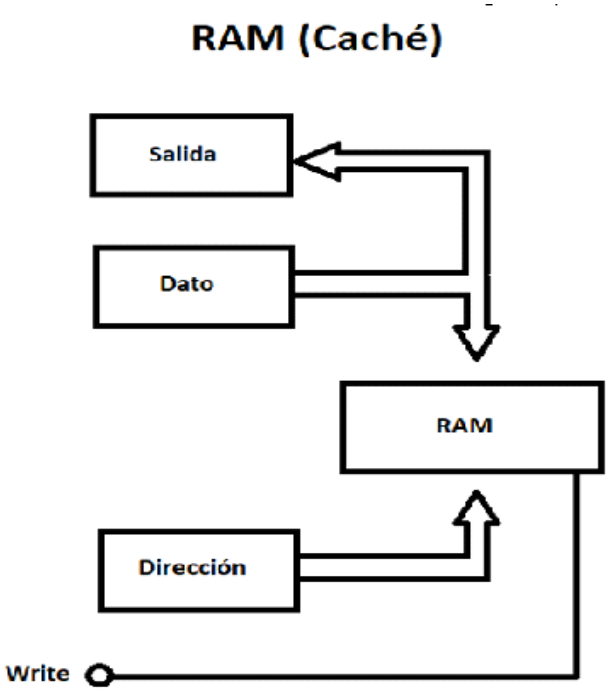
ALU

Bueno, se trata de una sección imprescindible como otras, ya que sin él o ella, no podría existir un sistema computacional como tal.
Aquí los datos pueden procesarse como:
COMPARADOR: =, >, <, >=, <= y !=
SUMADOR
OPERACIONES DE BIT: AND, OR, XOR, SHIFT-R y SHIFT-L.
Además posee otra función, que es la de generar el salto de dirección correspondiente, según el programa indique cuándo se encuentra en una condición de bifurcación.

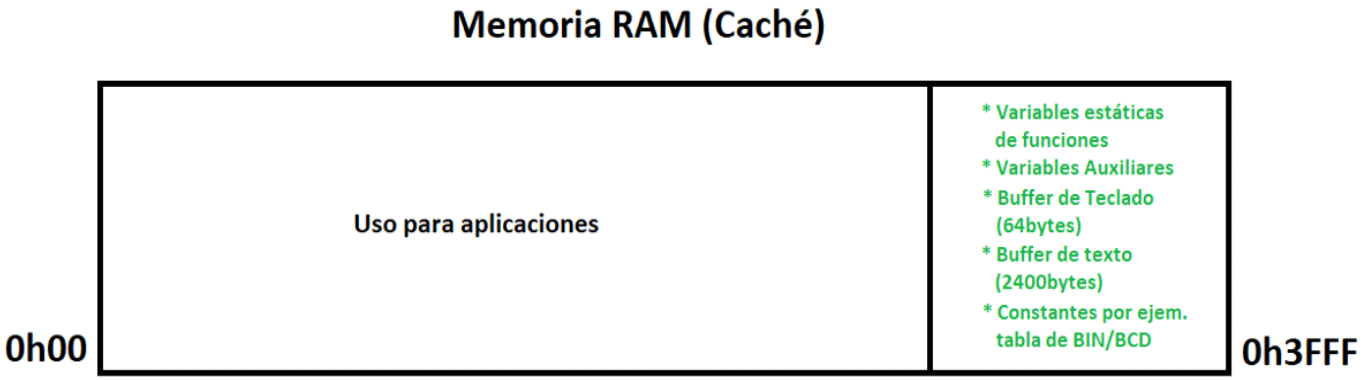


MEMORIA CACHÉ

Bien, aquí se almacenan todas las variables que utiliza el sistema, hay variables de tipo exclusivas, globales y auxiliares, que más adelante se detallan.
Más abajo podemos apreciar su configuración.



Distribución de datos dentro de la misma, como se indica.



MEMORIA PRINCIPAL

La memoria principal, se encarga de almacenar los programas que se cargan de la tarjeta de almacenamiento.
La rutina LOAD, se encarga de gestionar ésta tarea.
Una vez cargado dicho programa, entra en acción otra rutina de ejecución.

TIMERs DEL SISTEMA

Éste apartado, se encarga de gestionar hasta 8 temporizadores independientes.
Pueden configurarse individualmente y cada uno entrega una señal única, indicando que su tiempo se ha completado.

CONTROLADORES

Aquí los controladores, se mencionan con una definición en común, y es que cada uno controla específicamente según el periférico asociado. Y son del tipo SECUENCIAL.

Capítulo 6.

VARIABLES

Las mismas, se organizaron en básicamente cuatro grupos.

* PRIVADAS
* PÚBLICAS
* AUXILIARES
* ARRAYS

PRIVADAS

Son variables utilizadas únicamente por rutinas específicas y NO se comparten con otras rutinas.
Podríamos imaginarlas como variables LOCALES. Pero a diferencias de éstas, NO se destruyen sus valores al salir de las rutinas.

PÚBLICAS

Éstas variables son del tipo "obreras", ya que llevan datos de una rutina a otra, pueden ser de naturaleza puntero o no.

AUXILIARES

Éste último grupo, trata de variables de tipo LOCAL, pueden usarse por cualquier rutina, pero una vez abandonada la misma, su valor queda a la deriva. Ya que al ser usada por otra rutina, se le asignará el valor que corresponda, en la nueva utilidad.

ARRAYS

Aquí las variables se agrupan, para formar una cadena de datos.
Son muy utilizadas en los buffers de cada controladora, cada una con un tamaño específico y bien delimitada para evitar solapamientos con otros buffers.

Resumiendo, todas las variables poseen un rango de 32bits.
Lamentablemente, si sólo utilizamos 1 bit, se desperdician los restantes.
Ésto suena poco óptimo, pero de no ser así, nos vuelve complejo el firmware que se encarga de manejarlas, por otro lado, al tener disponible 32Mb, podemos tener el lujo de ese desperdicio.
Además todas las variables pueden comportarse como variable de dato o puntero.
No necesitan inicializarse como tal.
Para lograr que una variable se comporte como puntero, se requiere de hardware adicional. Y consiste en realimentar la dirección de memoria con la dirección de datos y viceversa.
Ponemos el sig. ejem:
La instrucción B13, nos direcciona a la memoria RAM, luego, le añadimos el argumento 56, que corresponde a la dirección física de memoria.
Luego cargamos en el registro de datos el valor 109.
Y por último guardamos el valor, con la instrucción B15, recordemos que ésta última, NO requiere argumento.
Todo, nos queda así:

```
B13 56
B12 109
B15  --
```

Ésto se traduce a: El dato 109, se almacena en la dirección 56.
Ahora, asignaremos el valor 109 en un puntero, es decir, ese dato debe transformarse en una dirección de memoria. Y para llevarlo a cabo, añadimos ahora la selección de registro número 7, que nos devuelve el dato alojado en la dirección seteada.
Entonces, se escribe como veremos:

```
B13 56
A12 7
B13 --
```

Podemos ver que, una vez seleccionado el selector de registros al número 7, NO es necesario añadir argumento a la instrucción B13, ¿Por qué?
Porque el registro 7, provee ese dato al BUS principal, y B13 lo toma para si!
Entonces ahora, el dato 109, se transformó en una dirección.

CONCLUSIONES

El desarrolo de la PDC32, sólo nos sirve como fin didáctico, podemos considerarla como, un "juguete algo elaborado".
Desde esa perspectiva, podemos decir que, está muy lejos de poder transformarse en algo rentable o de utilidad práctica.
Pero, hay algo que podría cambiar esa premisa, y es que podríamos migrar toda la arquitectura a una FPGA, y en esas circunstancias, se obtendría un sistema más compacto, más veloz y con la posibilidad de añadir funciones.

Autor: Roberto Duberlín Gudiño Barraud.-