Proyecto Final: Implementación de Procesador ARMv4

Kenneth Hernández, Marco Herrera, and Jasson Rodríguez

Ingeniería en Computadores, Instituto Tecnológico de Costa Rica

Noviembre 2019

1. Comprensión del problema

El problema en desarrollo consiste en la implementación de un procesador completo capaz de interactuar con diversos periféricos. Además se debe diseñar un programa en lenguaje ensamblador que demuestre el correcto funcionamiento del procesador. En este caso se trata del juego flappy bird implementado en el set de instrucciones de arquitectura ARMv4.

2. Investigación

Es esencial conocer las características de la arquitectura que se desea implementar; en este caso se utilizará la arquitectura ARMv4. El procesador de arquitectura ARMv4 posee 15 registros de uso general así como el Program Counter (PC). De los 15 registros disponibles, por estandar, uno de estos corresponde al *stack pointer* y otro de estos al *link register*.

La arquitectura de ARMv4 posee los tipos de instrucciones predefinidos que se muestran a continuación, cada uno de estos se caracteriza por tener una encondificación particular según el formato al que pertenecen. En la figura 1 se muestra la encodificación de las instrucciones de proceso de datos.

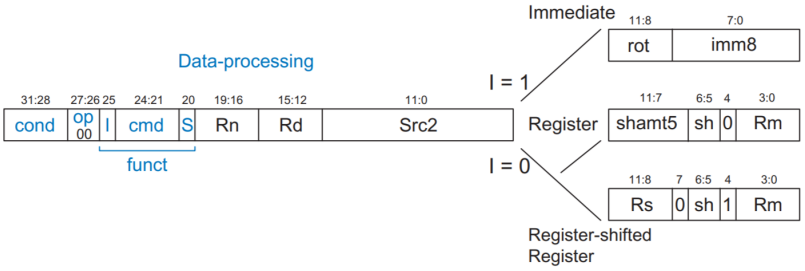

Figura 1: Formato de la arquitectura ARMv4 para las instrucciones que procesan datos. Fuente: Harris y Harris [2]

En la siguiente imagen se muestra el conjunto de instrucciones que pertenecen a esta familia.

cmd	Name	Description	Operation
0000	`AND Rd, Rn, Src2`	Bitwise AND	`Rd ← Rn & Src2`
0001	`EOR Rd, Rn, Src2`	Bitwise XOR	`Rd ← Rn ^ Src2`
0010	`SUB Rd, Rn, Src2`	Subtract	`Rd ← Rn - Src2`
0011	`RSB Rd, Rn, Src2`	Reverse Subtract	`Rd ← Src2 - Rn`
0100	`ADD Rd, Rn, Src2`	Add	`Rd ← Rn+Src2`
0101	`ADC Rd, Rn, Src2`	Add with Carry	`Rd ← Rn+Src2+C`
0110	`SBC Rd, Rn, Src2`	Subtract with Carry	`Rd ← Rn - Src2 -` $\overline{C}$
0111	`RSC Rd, Rn, Src2`	Reverse Sub w/ Carry	`Rd ← Src2 - Rn -` $\overline{C}$
1000 ($S = 1$)	`TST Rd, Rn, Src2`	Test	`Set flags based on Rn & Src2`
1001 ($S = 1$)	`TEQ Rd, Rn, Src2`	Test Equivalence	`Set flags based on Rn ^ Src2`
1010 ($S = 1$)	`CMP Rn, Src2`	Compare	`Set flags based on Rn - Src2`
1011 ($S = 1$)	`CMN Rn, Src2`	Compare Negative	`Set flags based on Rn+Src2`
1100	`ORR Rd, Rn, Src2`	Bitwise OR	`Rd ← Rn | Src2`
1101	Shifts:		
$I = 1$ OR ($\text{instr}_{11:4} = 0$)	`MOV Rd, Src2`	Move	`Rd ← Src2`
$I = 0$ AND ($sh = 00$; $\text{instr}_{11:4} \neq 0$)	`LSL Rd, Rm, Rs/shamt5`	Logical Shift Left	`Rd ← Rm << Src2`
$I = 0$ AND ($sh = 01$)	`LSR Rd, Rm, Rs/shamt5`	Logical Shift Right	`Rd ← Rm >> Src2`
$I = 0$ AND ($sh = 10$)	`ASR Rd, Rm, Rs/shamt5`	Arithmetic Shift Right	`Rd ← Rm>>>Src2`
$I = 0$ AND ($sh = 11$; $\text{instr}_{11:7,\ 4} = 0$)	`RRX Rd, Rm, Rs/shamt5`	Rotate Right Extend	`{Rd, C} ← {C, Rd}`
$I = 0$ AND ($sh = 11$; $\text{instr}_{11:7} \neq 0$)	`ROR Rd, Rm, Rs/shamt5`	Rotate Right	`Rd ← Rn ror Src2`
1110	`BIC Rd, Rn, Src2`	Bitwise Clear	`Rd ← Rn & ~Src2`
1111	`MVN Rd, Rn, Src2`	Bitwise NOT	`Rd ← ~Rn`

Figura 2: Instrucciones de la arquitectura ARMv4 encargadas de procesar datos. Fuente: Harris y Harris [2]

Además de las instrucciones de procesamiento de datos, cabe resaltar las instrucciones dedicadas a la multiplicación, que cumplen con el formato de la figura 3. En la figura 4 se muestra un listado de las instrucciones que corresponden a este formato.

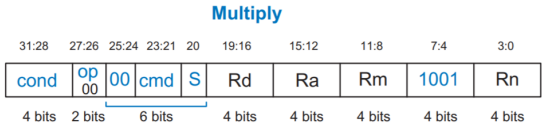

Figura 3: Formato de la arquitectura ARMv4 para las instrucciones encargadas de multiplicar. Fuente: Harris y Harris [2]

cmd	Name	Description	Operation
000	`MUL Rd, Rn, Rm`	Multiply	`Rd ← Rn × Rm (low 32 bits)`
001	`MLA Rd, Rn, Rm, Ra`	Multiply Accumulate	`Rd ← (Rn × Rm)+Ra (low 32 bits)`
100	`UMULL Rd, Rn, Rm, Ra`	Unsigned Multiply Long	`{Rd, Ra} ← Rn × Rm (all 64 bits, Rm/Rn unsigned)`
101	`UMLAL Rd, Rn, Rm, Ra`	Unsigned Multiply Accumulate Long	`{Rd, Ra} ← (Rn × Rm)+{Rd, Ra} (all 64 bits, Rm/Rn unsigned)`
110	`SMULL Rd, Rn, Rm, Ra`	Signed Multiply Long	`{Rd, Ra} ← Rn × Rm (all 64 bits, Rm/Rn signed)`
111	`SMLAL Rd, Rn, Rm, Ra`	Signed Multiply Accumulate Long	`{Rd, Ra} ← (Rn × Rm)+{Rd, Ra} (all 64 bits, Rm/Rn signed)`

Figura 4: Instrucciones de la arquitectura ARMv4 encargadas de multiplicar datos. Fuente: Harris y Harris [2]

Las instrucciones que involucran acceso a la memoria de datos se muestran en la figura 7 y cumplen con el formato de la figura 5 o bien el formato de la figura 6.

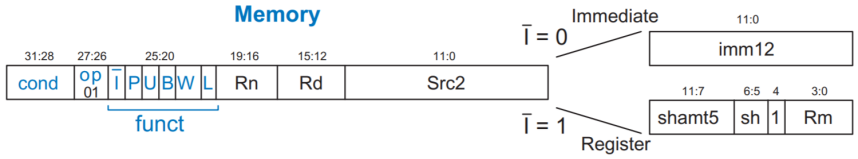

Figura 5: Formato de la arquitectura ARMv4 para las instrucciones que manejan la memoria. Fuente: Harris y Harris [2]

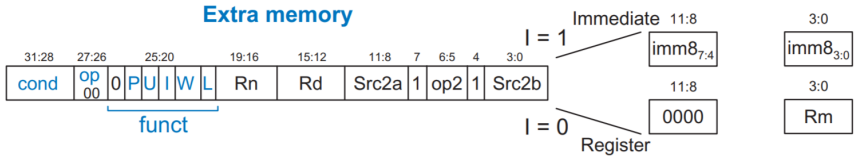

Figura 6: Formato de la arquitectura ARMv4 para las instrucciones alternativas que manejan la memoria. Fuente: Harris y Harris [2]

op	B	op2	L	Name	Description	Operation
01	0	N/A	0	STR Rd, [Rn, ±Src2]	Store Register	Mem[Adr] ← Rd
01	0	N/A	1	LDR Rd, [Rn, ±Src2]	Load Register	Rd ← Mem[Adr]
01	1	N/A	0	STRB Rd, [Rn, ±Src2]	Store Byte	Mem[Adr] ← $Rd_{7:0}$
01	1	N/A	1	LDRB Rd, [Rn, ±Src2]	Load Byte	Rd ← $Mem[Adr]_{7:0}$
00	N/A	01	0	STRH Rd, [Rn, ±Src2]	Store Halfword	Mem[Adr] ← $Rd_{15:0}$
00	N/A	01	1	LDRH Rd, [Rn, ±Src2]	Load Halfword	Rd ← $Mem[Adr]_{15:0}$
00	N/A	10	1	LDRSB Rd, [Rn, ±Src2]	Load Signed Byte	Rd ← $Mem[Adr]_{7:0}$
00	N/A	11	1	LDRSH Rd, [Rn, ±Src2]	Load Signed Half	Rd ← $Mem[Adr]_{15:0}$

Figura 7: Instrucciones de la arquitectura ARMv4 encargadas de manejar los accesos a memoria. Fuente: Harris y Harris [2]

Por último se encuentran las instrucciones de salto o ramificación, estas se muestran en la figura 9 y cumplen el formato de la figura 8.

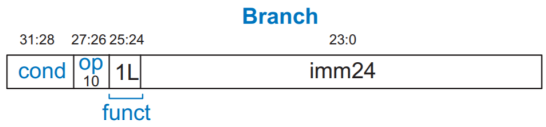

Figura 8: Formato de la arquitectura ARMv4 para las instrucciones encargadas de saltos. Fuente: Harris y Harris [2]

L	Name	Description	Operation
0	B label	Branch	PC ← (PC+8)+imm24 << 2
1	BL label	Branch with Link	LR ← (PC+8) - 4; PC ← (PC+8)+imm24 << 2

Figura 9: Instrucciones de la arquitectura ARMv4 encargadas de manejar los saltos entre instrucciones. Fuente: Harris y Harris [2]

Una vez definidas las características del procesador que se pretende diseñar, es importante planear y conocer las propiedades de los periféricos que interactúan con el procesador. En este caso los protocolos utilizados son el de PS/2 para utilizar un teclado de entrada, y el protocolo VGA para mostrar la salida en un monitor.

En la figura 10 se muestran las señales necesarias para el protocolo ps2 y en la figura 12 se muestran las señales respectivas al protocolo VGA. Por otra parte en las figuras 11 y 13 se muestra la sincronización de dichas señales.

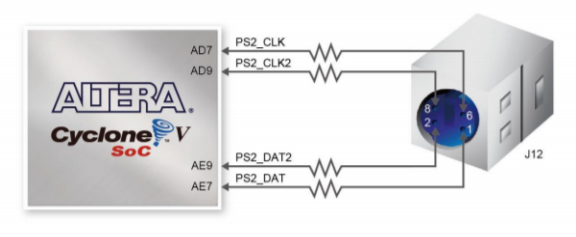

Figure 3-27 Connection between FPGA and PS/2

Figura 10: Señales de sincronización y datos para el protocolo Ps/2. Fuente: Manual Altera [1]

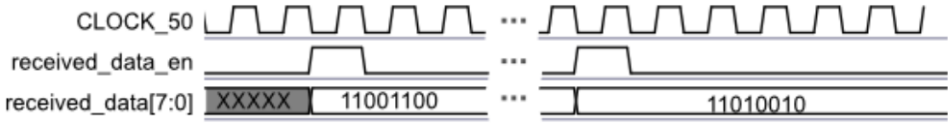

Figura 11: Señales de sincronización y datos para el protocolo VGA. Fuente: Manual Altera [1]

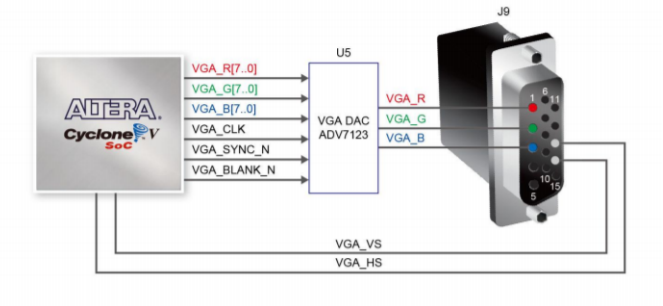

Figure 3-21 VGA Connections between FPGA and VGA

Figura 12: Señales de sincronización y datos para el protocolo VGA. Fuente: Manual Altera [1]

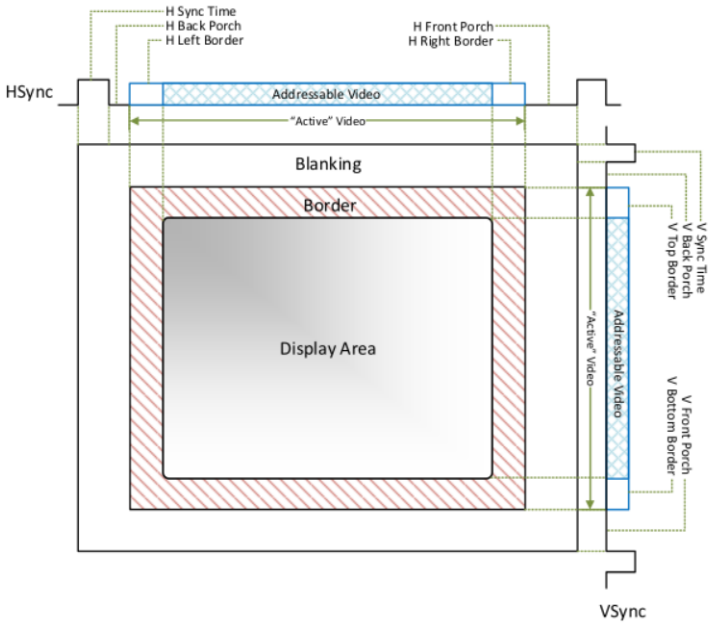

Figura 13: Señales de sincronización y datos para el protocolo VGA. Fuente: Manual Altera [1]

3. Estimación de la solución

A partir de un lenguaje de descripción de hardware, describir e implementar una versión funcional de un procesador completo que siga las especificaciones del set de instrucciones ARMv4, junto con el manejo de un periférico de entrada, en este caso el teclado a través del protocolo PS/2; además se pretende escribir una versión del juego Flappy bird y realizar una interfaz de este con un periférico de salida, el cual será un monitor controlado a través del protocolo VGA.

4. Objetivos de la solución

- Implementar una versión funcional de un procesador completo que cumpla con el estándar ARMv4.
- Crear en lenguaje ensamblador una versión del juego Flappy bird.
- Diseñar e implementar un controlador para teclado que siga los requerimientos del protocolo PS/2.
- Diseñar e implementar un controlador funcional de VGA que cumpla con los requerimientos estándar.
- Utilizar la menor cantidad posible de unidades básicas de la FPGA.

5. Diseño del hardware del sistema

A continuación se presenta una propuesta para la implementación de la solución, se clasifican según su nivel de detalles. Además contienen una breve explicación de cada una de sus entradas y salidas; junto con su funcionamiento general.

5.1. Primer Nivel

Figura 14: Diagrama de primer Nivel

Objetivo: Implementar un computador mínimo, con un procesador ARMv4 que sea capaz de interactuar con el exterior a través de periféricos, con entrada de un teclado y salida para un monitor.

Entradas:

- Teclado: Señal de entrada mediante el protocolo PS/2 para interactuar con el sistema.

Salidas

- Monitor: Salida de video tipo VGA para ser mostrada en un monitor con resolución 640x480 pixeles.

Explicación General: Este modulo representa la funcionalidad total del computador mínimo, en este caso será utilizado para ejecutar una versión del juego Flappy Bird. Las entradas del computador controlarán el juego a través del teclado, y la toda la interfaz del juego y los resultados serán mostrados en un monitor conectado a la salida de video del sistema.

5.2. Segundo Nivel

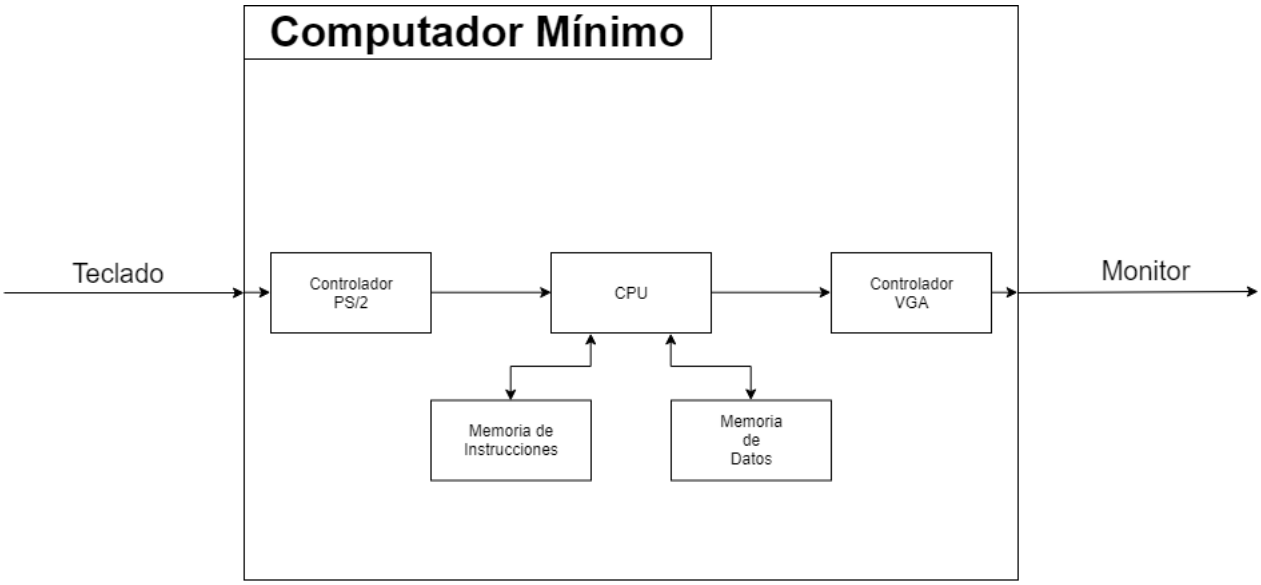

Figura 15: Diagrama de segundo Nivel

5.2.1. Controlador PS/2

Objetivo: Interpretar las señales provenientes del teclado para que puedan ser utilizadas por la unidad de procesamiento.

Entradas:

- Teclado: Señal serial lógica que define cuando se presionó una tecla y cual de estas fue.

Salidas:

- Tecla_presionada: valor lógico estabilizado que indica cual tecla se presionó.
- Interrupción: señal lógica que indica cuando se presionó una tecla.

Explicación General: Este módulo representa una interfaz de las señales provenientes del teclado, acoplándolas y poniéndolas a disposición del procesador; esto se logra pues el procesador escucha la señal de interrupción y se detiene para leer el valor de la tecla presionada. La salida de este módulo es esencial para la interacción del procesador con el exterior pues maneja los periféricos de entrada.

5.2.2. Controlador VGA

Objetivo: Adaptar y sincronizar las señales provenientes del procesador para mostrar una imagen a través del protocolo VGA.

Entradas:

-

- Select Button: Botón utilizado para seleccionar la posición actual del tablero

Salidas

- RGB: Color del pixel que se va a dibujar

- Hsync:Señal de sincronización horizontal
- Vsync:Señal de sincronización vertical
- Blank: Señal para mostrar o no el color del pixel

Explicación General: Este módulo recibe sus entradas del procesador, el procesador le específica lo que se debe mostrar en el monitor al controlador a través de una sección de memoria destinada para esto. A partir de esto, el controlador calcula el color necesario, además si cada uno de los sprites es visible o no. Además genera las señales de sincronización necesarias para la comunicación con el periférico respectivo.

5.2.3. Memoria de instrucciones

Objetivo: Almacenar el conjunto de instrucciones que el programa ejecutará Entradas:

- PC: Dirección de lectura, de donde se obtendrá la siguiente instrucción.

Salidas

- Instrucción: representación binaria de la instrucción que se encuentra en la dirección de PC.

Explicación General: Corresponde a una memoria de lectura que contiene las instrucciones para un programa específico; estas se encuentran ya decodificadas en binario para que sean ejecutadas por el procesador. Se acceden a través de la dirección del PC.

5.2.4. Memoria de Datos

Objetivo: Almacenar los datos que utilizará el procesador para la ejecución de los programas Entradas:

- address: dirección de lectura o escritura
- data_in: dato que se escribirá en la posición definida cuando se realiza una escritura
- we: señal lógica que determina si se realiza una escritura o una lectura

Salidas

- data_out: dato que se encuentra en la posición que especifíca la dirección en el caso de una lectura

Explicación General: Este módulo es el encargado de almacenar los datos que el procesador necesita, así como resultados finales o parciales, además le permite al procesador comunicarse con otros dispositivos a través de este módulo.

5.2.5. CPU

Objetivo: Interpretar, procesar y ejecutar una serie de instrucciones para la obtención de un propósito final
Entradas:

- Instr: Instrucción que se ejecutará en el ciclo respectivo. Obtenida de la lectura de la memoria de instrucciones.
- Dato: Valor obtenido de la lectura de la memoria de datos

Salidas

- PC: Dirección de lectura para la memoria de instrucciones

- Addr: Dirección de escritura para la memoria de datos
- data: Dato para escritura de la memoria de datos

Explicación General: Este módulo es el componente principal del computador mínimo, se encarga de todo el procesamiento y ejecución del programa, parte de las instrucciones almacenadas en la memoria de instrucciones, para posteriormente ejecutar el proceso necesario y finalmente almacenar el resultado para que sea utilizado por un periférico.

5.3. Tercer Nivel

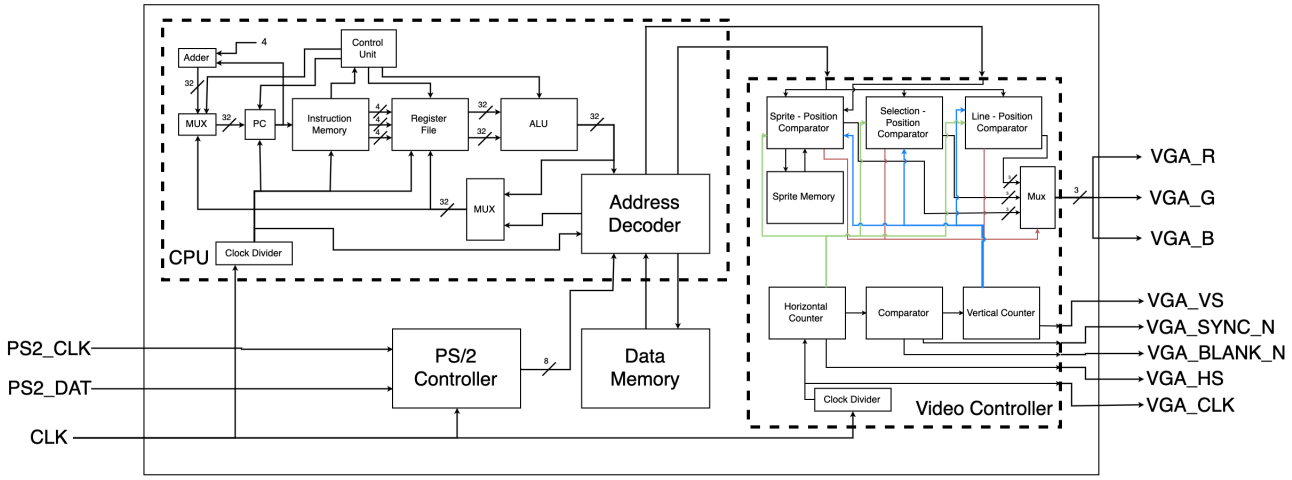

Figura 16: Diagrama de tercer Nivel

5.3.1. Memoria de instrucciones

Objetivo: El objetivo de este modulo es el contener todas las instrucciones que ejecutara el procesador en el programa realizado.

Entradas:

- PC: Esta es una posición de memoria de aproximadamente 32 bits, utilizada para cargar una instrucción en especifico.

Salidas:

- Instruccion: Es un numero de 32 bits que representa la instruccion a ejecutar por el procesador.

Explicación General: Este modulo como su nombre lo dice es una memoria la cual contiene todas las instrucciones de un programa en especifico, en este caso del programa de flappy bird,este recibe una posición de memoria PC de 32 bits y genera de salida la instrucción de 32 bits contenida en dicha posición.

5.3.2. Memoria de Registros

Objetivo: Almacena toda la informacion que se encuentra en los diferentes registros del procesador(16 nen total).

Entradas:

- RA1:Es la posición de memoria de alguno de los 16 registros.

- RA2:Es la posición de memoria de alguno de los 16 registros.
- Instruccion: Esto son 4 bits de la instrucción que contienen.
- Resultado: este es un dato que proviene de un resultado de la ALU o de la memoria
- CLK: Este es el relog del procesador, esta memoria escribe en el ciclo negativo.
- RegWrite: Esta es una señal que le indica a la memoria escribir un dato en el registro que entra a WD3

Salidas:

- SalB: Es un dato de 32 bits contenido en la posición de memoria que indica RA2.
- SrcA: Es un dato de 32 bits contenido en la posición de memoria que indica RA1.

Explicación General: Este modulo ejecuta el manejo de los registros, tanto como lectura como guardado de datos, en cada ciclo se escribe PC+8 en la posición numero 16 de la memoria y en caso de que se requiera se escribe el dato Resultado en el registro indicado por Instruccion.

5.3.3. Memoria de Datos

Objetivo: Permite almacenar diferentes datos necesarios para el programa en ejecución.
Entradas:

- ResultadoALU: Esta es la posición de memoria del dato que se quiere leer.
- Clk: esta señal representa el reloj de la memoria, la cual escribe en su ciclo negativo.
- Dato: Es un dato de 32 bits que se requiere escribir en la memoria.
- MemWrite: Esta es una señal que le indica a la memoria si escribir el dato recibido o no.

Salidas:

- DatoLeido: Es el dato extraído de la posición de memoria indicada por ResultadoALU.

Explicación General: Este modulo es una memoria en la cual se escriben diferentes datos requeridos por el programa en ejecución en el procesador.

5.3.4. Mux

Objetivo: Permite seleccionar entre los datos de un registro y un inmediato para la operacion en la ALU
Entradas:

- ExtImm: Este es un inmediato proveniente de la instrucción a ejecutar.
- SalB: Este es un dato que sale de la memoria de registros.
- ALUSrc: Señal de un bit que permite elegir entre el inmediato y el dato de un registro dependiendo de la instrucción.

Salidas:

- SrcB: Un operando de 32 de la ALU.

Explicación General: Este multiplexor simple nos permite elegir entre un inmediato y un dato de un registro dependiendo del tipo de instruccion que se este ejecutando.

5.3.5. ALU

Objetivo: Permite realizar las diferentes operaciones lógicas como aritméticas.
Entradas:

- SrcA y SrcB: Son los dos operandos a los cuales se les va a aplicar una operacion.
- ALUControl, esta es una señalde 2 bits indicando que operacion realizar,10 para and,01 resta,00 suma y 11 es or.

Salidas:

- ResultadoALU: Este es el resultado de haber aplicado operaciones a SrcA y SrcB
- ALUFlags: Estas son banderas generadas por las operaciones de la ALU

Explicación General: Este es el modulo que ejecuta todas las operaciones logicas y aritmeticas del procesador, teniendo dos entradas SrcA y SrcB como operandos y el resultado como salida.

5.3.6. Sumadores

Objetivo: Permite ejecutar la suma de la dirección de instrucciones en cada ciclo.
Entradas:

- PC: Dirección en la memoria de direcciones de la instrucción en ejecución.
- 4: Es el numero 4 en binario como entrada para realizar la suma.

Señales intermedias:

- PC+4: Dirección de la siguiente instrucción a ejecutar.

Salidas:

- PC+8: es una posición de la memoria de instrucciones que sera guardada en el registro R15 y sera usado para realizar operaciones de salto.

Explicación General: Estos dos modulos son sumadores simples utilizados para realizar la sumas necesarias para el manejo de las instrucciones, tanto para cargar la siguiente instrucion como para realizar saltos.

5.3.7. Mux Decodificacion

Objetivo: Estos multiplexores permiten seleccionar entre los diferentes registros a decodificar.
Entradas:

- Instrucción[19:16],Instrucción[3:0],Instrucción[15:12]: Registros a seleccionar para decodificación provenientes de la introducción.
- 15: Este es una constante hecha para seleccionar el registro 15.
- RegSrc: Esta es la señal de selección para el mux.

Salidas:

- RA2 y RA1: Son un numero de 4 bits seleccionado por el mux dependiendo de la instrucción a decodificar.

Explicación General: Estos dos modulo como se menciono anteriormente se utilizan para seleccionar entre las diferentes opciones de posiciones de memoria de la memoria de registros.

5.3.8. Extendedor

Objetivo: Este modulo permite extender el inmediato proveniente de la instruccion.
Entradas:

- Instrucción[23:0]: Inmediato proveniente de la instrucción.
- ImmSrc:Decide entre extender el numero con 1 o con 0.

Salidas:

- ExtImm: Es el inmediato proveniente de la instrucción luego de ser extendido por el modulo.

Explicación General: Este modulo es el que se encarga de extender el numero proveniente de una instrucción a un numero de 32 bits con una extensión definida por ImmSrc.

5.3.9. Unidad de control

Objetivo: Este modulo es el cerebro del procesador ya que a partir de las señales que entra genera todas las señales de control necesarias para ejecutar las diferentes instrucciones.
Entradas:

- Instrucción: Esto encapsula todas las entradas de la instrucción, estas son las que deciden que salidas se activan o no, todo esto depende de la instrucción que entra a la unidad de control.
- Flags: Estas son las flags generadas por la ALU, esta es utilizada en el caso de que se necesite realizar un salto en las instrucciones.

Salidas:

- PcSrc: Esta señal se encarga de la seleccion de donde proviene el siguiente PC.
- MemToReg: Se encarga de la seleccion del dato que sera escrito en un registro del banco de registros.
- MemWrite:Permite que se escriba en memoria.
- AluControl: Permite la seleccion de que operacion tanto aritmetica como logica que se aplicara en la ALU.
- AluSrc: Permite la seleccion de uno de los operandos de la ALU.
- ImmSrc: Este s el que permite seleccionar que tipo de extension se debe aplicar al inmediato de la instruccion a ejecutar.
- RegWrite: Es el que permite escribir en la memoria de registros
- RegSrc:Permite la seleccion de las direcciones de los registros a decodificar en el banco de registros.

Explicación General: Como se menciono anteriormente este es el cerebro del procesador, ya que genera todas las señales de control para la ejecución de instrucciones, incluido las decisiones de la realización de datos.

5.3.10. Controlador PS/2

Objetivo: Interpretar las señales provenientes del teclado para que puedan ser utilizadas por la unidad de procesamiento.
Entradas:

- PS2_CLK: Señal de reloj para la sincronización de lectura.
- PS2_DAT: Flujo de datos que define el inicio y fin de una tecla presionada además de la tecla que fue presionada.

Salidas:

- Recieved: valor lógico estabilizado que indica cuando un valor es recibido del teclado;
- Data: valor lógico que especifica cual tecla fue presionada;

Explicación General: Este módulo representa una interfaz de las señales provenientes del teclado, acoplándolas y poniéndolas a disposición del procesador; esto se logra pues el procesador escucha la señal de interrupción y se detiene para leer el valor de la tecla presionada. La salida de este módulo es esencial para la interacción del procesador con el exterior pues maneja los periféricos de entrada.

A continuación se muestran todos los componentes que forman parte del módulo presentado en el diagrama de segundo nivel como Renderer.

5.3.11. Sprite Memory

Objetivo: Permite almacenar la representación del Sprite de los jugadores.
Entradas:

- Addr: esta señal lógica de 10 bits corresponde a la dirección de la memoria que se desea acceder.

Salidas:

- Data: esta señal corresponde al dato almacenado en la posición seleccionada en addr.

Explicación General: Esta memoria de lectura contiene un Sprite de 32 x 32 pixeles, dado que es una memoria, recibe una dirección de lectura y obtiene el dato almacenado en esta.

5.3.12. Sprite-Position Comparator

Objetivo: Determinar si en la posición entrante se debe dibujar un sprite.
Entradas:

- row: posición vertical en la que se encuentra el pixel a dibujar
- column: posición horizontal en la que se encuentra el pixel a dibujar
- game_state: estado del tablero para así determinar si se debe dibujar un sprite o no.
- data: datos del sprite almacenados en la memoria en la posición estipulada.

Salidas:

- Address: posición del pixel en la memoria, que se desea dibujar.
- RGB: información de la composición del pixel que se dibujará en la pantalla.
- visible: valor lógico de 1 bit que define si en la posición definida se debe dibujar un sprite o no.

Explicación General: Esta componente realiza una comparación de la posición en la que se dibujará un píxel y determina si este posición se encuentra en un rango donde se deba dibujar un sprite

---

5.3.13. Selection-Position Comparator

Objetivo: Determinar si en la posición entrante se debe dibujar un color distinto en el fondo para determinar que se encuentra seleccionado por el usuario.
Entradas:

- row: posición vertical en la que se encuentra el pixel a dibujar
- column: posición horizontal en la que se encuentra el pixel a dibujar
- game_state: estado del tablero para así determinar si se debe dibujar un sprite o no.
- data: datos del sprite almacenados en la memoria en la posición estipulada.

Salidas:

- Address: posición del pixel en la memoria, que se desea dibujar.
- RGB: información de la composición del pixel que se dibujará en la pantalla.
- visible: valor lógico de 1 bit que define si en la posición definida se debe cambiar el color de fondo o no.

Explicación General: Esta componente realiza una comparación de la posición en la que se dibujará un píxel y determina si este posición se encuentra en un rango donde se deba cambiar el fondo del tablero pues el jugador está seleccionando esta posición.

5.3.14. Line-Position Comparator

Objetivo: Determinar si en la posición entrante se debe dibujar una linea del tablero.
Entradas:

- row: posición vertical en la que se encuentra el pixel a dibujar
- column: posición horizontal en la que se encuentra el pixel a dibujar

Salidas:

- RGB: información de la composición del pixel que se dibujará en la pantalla, representa el color de la línea.
- visible: valor lógico de 1 bit que define si en la posición definida se debe dibujar una línea o no.

Explicación General: Esta componente realiza una comparación de la posición en la que se dibujará un píxel y determina si este posición se encuentra en un rango donde equivale a una linea del tablero por lo que se debe cambiar el color por mostrar.

5.3.15. Mux

Objetivo: Decidir cual de las señales será mostrada en pantalla entre sus posibles entradas.
Entradas:

- RGB1: información de la composición del pixel que se dibujará en la pantalla, representa el color del sprite.
- RGB2:información de la composición del pixel que se dibujará en la pantalla, representa el color de la línea.
- RGB3:información de la composición del pixel que se dibujará en la pantalla, representa el color del fondo si este se encuentra seleccionado.

Salidas:

- VGA_R: bus de datos de 8 bits con la intensidad del color rojo del pixel seleccionado.

- VGA_G: bus de datos de 8 bits con la intensidad del color verde del pixel seleccionado.
- VGA_B: bus de datos de 8 bits con la intensidad del color azul del pixel seleccionado.

Explicación General: Este componente es de gran importancia y permite controlar y coordinar la información que está siendo enviada sobre cada pixel, a partir de este componente se conoce la posición que debe tomar cada pixel y además la señal de la fila y columna actual permiten a los demás componentes decidir que deberá mostrarse en dicha posición.

A continuación se muestran todos los componentes que forman parte del módulo presentado en el diagrama de segundo nivel como VGA controller.

5.3.16. Horizontal Counter

Objetivo: Contar el número de columna en que se encuentra el píxel actual.
Entradas:

- pixel_clk: reloj de píxeles, su frecuencia define la velocidad en que se envían los datos de cada uno de estos píxeles.

Salidas:

- VGA_HS: Esta señal es la encargada de la sincronización horizontal.
- column: Este valor numérico indica en qué posición horizontal se encuentra el píxel que se está evaluando.

Explicación General: Este componente es de gran importancia y permite controlar y coordinar de forma horizontal la información que está siendo enviada sobre cada píxel. La señal de sincronización horizontal se activa cada vez que se inicia una nueva fila, además indica cuando se debe reiniciar el conteo de la columna. La salida del valor de la columna, se relaciona con el componente anterior que determina que debe ser dibujado según la posición (fila y columna) dada.

5.3.17. Vertical Counter

Objetivo: Contar el número de fila en que se encuentra el píxel actual.
Entradas:

- column_clk: reloj de columnas, esta señal se activa cada vez que el contador de columnas alcanza un valor específico, lo cual indica que debe haber un cambio de fila.

Salidas:

- VGA_VS: Esta señal es la encargada de la sincronización Vertical.
- row: Este valor numérico indica en qué posición vertical se encuentra el píxel que se está evaluando.

Explicación General: Este componente es de gran importancia y permite controlar y coordinar de forma vertical la información que está siendo enviada sobre cada píxel. La señal de sincronización vertical se activa cada vez que se inicia un nuevo frame, además indica cuando se debe reiniciar el conteo de la fila. La salida del valor de la fila, se relaciona con los componentes comparadores del renderer que determinan que debe ser dibujado según la posición (fila y columna) dada.

5.3.18. Comparator

Objetivo: Generar la señal de conteo de filas a partir del conteo de columnas.
Entradas:

- column_clk: Frequencia de cambio de columna, representa el movimiento horizontal.

Salidas:

- VGA_SYNC_N: esta señal no maneja ninguna señal de control o datos, solo debe ser utilizada durante el proceso que está activa la señal VGA_BLANK_N.
- VGA_BLANK_N: señal lógica que se activa cuando se encuentra fuera del rango visible para que el controlador no intente colocar los colores de esas posiciones.

Explicación General: Este componente realiza una comparación del valor de salida del contador horizontal, y una vez que este alcanza un valor específico, genera una señal que será utilizada para aumentar el contador vertical.

5.3.19. Clock Divider

Objetivo: Disminuir la frecuencia de operación del reloj base del sistema a la frecuencia apropiada para los pixeles.
Entradas:

- clk: Reloj base de aproximadamente 50MHz

Salidas:

- pixel_clk: Reloj de frecuencia apropiada para ser utilizada por el controlador VGA.

Explicación General: Este componente se comporta como un contador, que aumenta su conteo con cada flanco del reloj base, además de esto involucra un comparador encargado de generar una señal luego de una cantidad definida de ciclos, lo cual se representa en la salida como una disminución de la frecuencia del reloj. Basta con definir la cantidad de ciclos que deben transcurrir para modificar la división de la frecuencia.

5.4. Cuarto Nivel

A continuación se muestran todos los componentes que forman parte del módulo presentado en el diagrama de segundo nivel como CPU.

5.4.1. Memoria de instrucciones

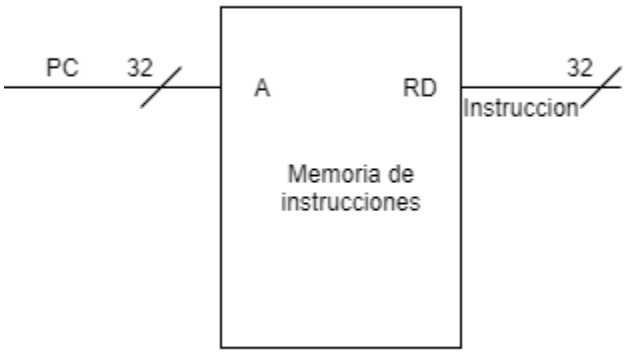

Figura 17: Elemento utilizado para almacenaje de instrucciones

Objetivo: El objetivo de este modulo es el contener todas las instrucciones que ejecutara el procesador en el programa realizado.

Entradas:

- PC: Esta es una posición de memoria de aproximadamente 32 bits, utilizada para cargar una instrucción en especifico.

Salidas:

- Instrucción: Es un numero de 32 bits que representa la instrucción a ejecutar por el procesador.

Explicación General: Este modulo como su nombre lo dice es una memoria la cual contiene todas las instrucciones de un programa en especifico, en este caso del programa de flappy bird,este recibe una posición de memoria PC de 32 bits y genera de salida la instrucción de 32 bits contenida en dicha posición.

5.4.2. Memoria de Registros

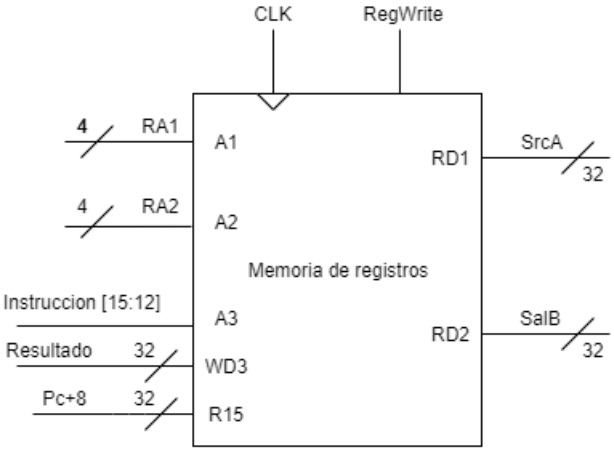

Figura 18: Representación de un contador de 4 bits

Objetivo: Almacena toda la informacion que se encuentra en los diferentes registros del procesador(16 nen total).

Entradas:

- RA1:Es la posición de memoria de alguno de los 16 registros.
- RA2:Es la posición de memoria de alguno de los 16 registros.
- Instruccion: Esto son 4 bits de la instrucción que contienen.
- Resultado: este es un dato que proviene de un resultado de la ALU o de la memoria
- CLK: Este es el relog del procesador, esta memoria escribe en el ciclo negativo.
- RegWrite: Esta es una señal que le indica a la memoria escribir un dato en el registro que entra a WD3

Salidas:

- SalB: Es un dato de 32 bits contenido en la posición de memoria que indica RA2.
- SrcA: Es un dato de 32 bits contenido en la posición de memoria que indica RA1.

Explicación General: Este modulo ejecuta el manejo de los registros, tanto como lectura como guardado de datos, en cada ciclo se escribe PC+8 en la posición numero 16 de la memoria y en caso de que se requiera se escribe el dato Resultado en el registro indicado por Instruccion.

5.4.3. Memoria de Datos

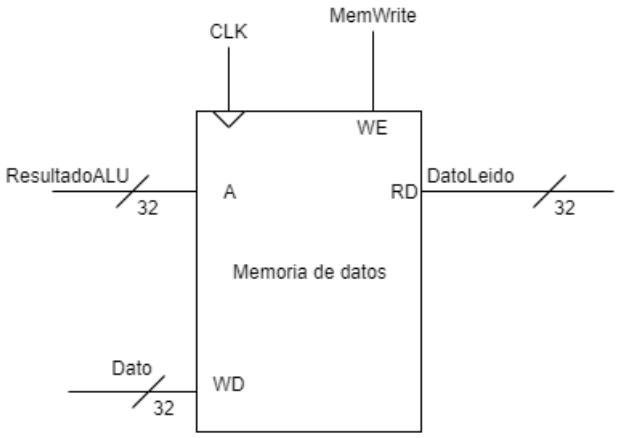

Figura 19: Memoria de datos del procesador

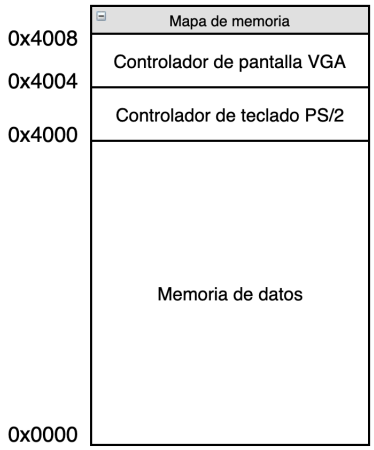

Figura 20: Mapa de memoria del procesador

Objetivo: Permite almacenar diferentes datos necesarios para el programa en ejecución.
Entradas:

- ResultadoALU: Esta es la posición de memoria del dato que se quiere leer.
- Clk: esta señal representa el reloj de la memoria, la cual escribe en su ciclo negativo.
- Dato: Es un dato de 32 bits que se requiere escribir en la memoria.
- MemWrite: Esta es una señal que le indica a la memoria si escribir el dato recibido o no.

Salidas:

- DatoLeido: Es el dato extraído de la posición de memoria indicada por ResultadoALU.

Explicación General: Este modulo es una memoria en la cual se escriben diferentes datos requeridos por el programa en ejecución en el procesador.

5.4.4. Mux

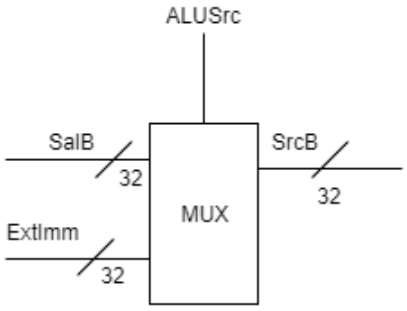

Figura 21: Mux de seleccion de operando

Objetivo: Permite seleccionar entre los datos de un registro y un inmediato para la operacion en la ALU
Entradas:

- ExtImm: Este es un inmediato proveniente de la instrucción a ejecutar.
- SalB: Este es un dato que sale de la memoria de registros.
- ALUSrc: Señal de un bit que permite elegir entre el inmediato y el dato de un registro dependiendo de la instrucción.

Salidas:

- SrcB: Un operando de 32 de la ALU.

Explicación General: Este multiplexor simple nos permite elegir entre un inmediato y un dato de un registro dependiendo del tipo de instruccion que se este ejecutando.

5.4.5. ALU

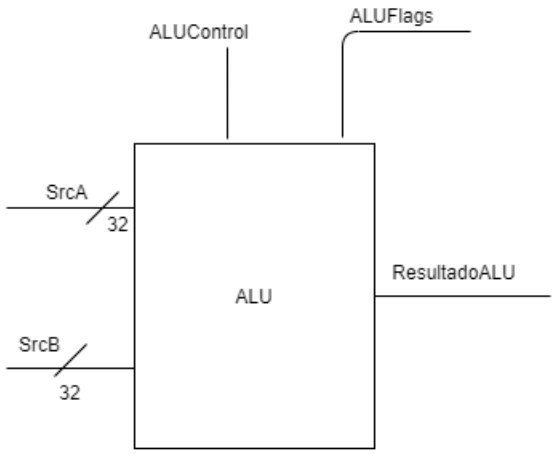

Figura 22: Representación de la unidad lógica aritmética

Table 7.4 ALU Decoder truth table enhanced for CMP

ALUOp	$Funct_{4:1}$ (*cmd*)	$Funct_0$ (*S*)	Notes	$ALUControl_{1:0}$	$FlagW_{1:0}$	*NoWrite*
0	X	X	Not DP	00	00	0
1	0100	0	ADD	00	00	0
		1			11	0
	0010	0	SUB	01	00	0
		1			11	0
	0000	0	AND	10	00	0
		1			10	0
	1100	0	ORR	11	00	0
		1			10	0
	1010	1	CMP	01	11	1

Figura 23: Tabla de verdad de la unidad lógica aritmética. Tomado de Harris y Harris [2]

Objetivo: Permite realizar las diferentes operaciones lógicas como aritméticas.
Entradas:

- SrcA y SrcB: Son los dos operandos a los cuales se les va a aplicar una operacion.
- ALUControl, esta es una señalde 2 bits indicando que operacion realizar,10 para and,01 resta,00 suma y 11 es or.

Salidas:

- ResultadoALU: Este es el resultado de haber aplicado operaciones a SrcA y SrcB
- ALUFlags: Estas son banderas generadas por las operaciones de la ALU

Explicación General: Este es el modulo que ejecuta todas las operaciones logicas y aritmeticas del procesador, teniendo dos entradas SrcA y SrcB como operandos y el resultado como salida.

5.4.6. Sumadores

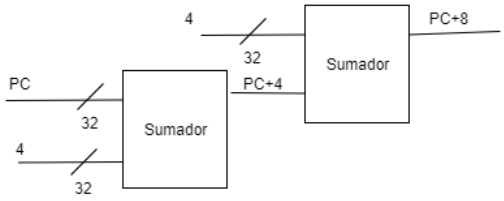

Figura 24: Sumadores de PC

Objetivo: Permite ejecutar la suma de la dirección de instrucciones en cada ciclo.
Entradas:

- PC: Dirección en la memoria de direcciones de la instrucción en ejecución.
- 4: Es el numero 4 en binario como entrada para realizar la suma.

Señales intermedias:

- PC+4: Dirección de la siguiente instrucción a ejecutar.

Salidas:

- PC+8: es una posición de la memoria de instrucciones que sera guardada en el registro R15 y sera usado para realizar operaciones de salto.

Explicación General: Estos dos modulos son sumadores simples utilizados para realizar la sumas necesarias para el manejo de las instrucciones, tanto para cargar la siguiente instrucion como para realizar saltos.

5.4.7. Mux Decodificacion

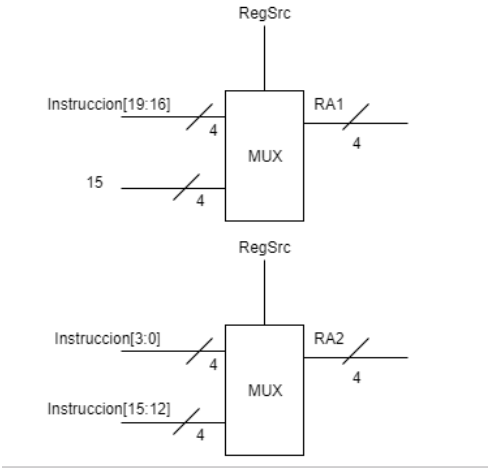

Figura 25: Sumadores de PC

Objetivo: Estos multiplexores permiten seleccionar entre los diferentes registros a decodificar.
Entradas:

- Instrucción[19:16],Instrucción[3:0],Instrucción[15:12]: Registros a seleccionar para decodificación provenientes de la introducción.
- 15: Este es una constante hecha para seleccionar el registro 15.
- RegSrc: Esta es la señal de selección para el mux.

Salidas:

- RA2 y RA1: Son un numero de 4 bits seleccionado por el mux dependiendo de la instrucción a decodificar.

Explicación General: Estos dos modulo como se menciono anteriormente se utilizan para seleccionar entre las diferentes opciones de posiciones de memoria de la memoria de registros.

5.4.8. Extendedor

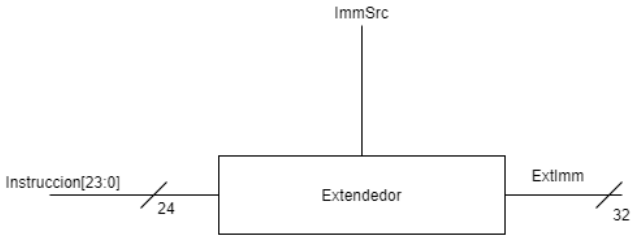

Figura 26: Sumadores de PC

Objetivo: Este modulo permite extender el inmediato proveniente de la instruccion.
Entradas:

- Instrucción[23:0]: Inmediato proveniente de la instrucción.
- ImmSrc:Decide entre extender el numero con 1 o con 0.

Salidas:

- ExtImm: Es el inmediato proveniente de la instrucción luego de ser extendido por el modulo.

Explicación General: Este modulo es el que se encarga de extender el numero proveniente de una instrucción a un numero de 32 bits con una extensión definida por ImmSrc.

5.4.9. Mux y registro

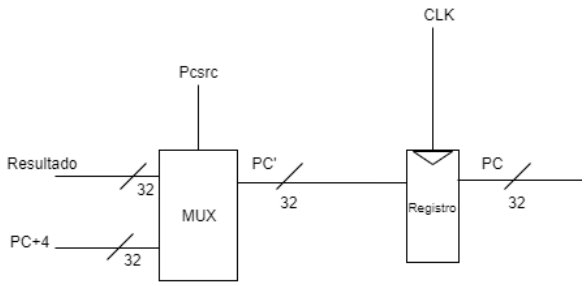

Figura 27: Pc junto su selección

Objetivo: Este par de módulos permite lo que vendría a ser los saltos tanto condicionales como no condicionales, también la carga de la siguiente instrucción o dirección.

Entradas:

- Resultado: Posición de memoria calculada por la ALU o cargada de la sección de memoria.
- PC+4:Dirección de memoria de la siguiente instrucción a ejecutar por el procesador.
- CLK:Señal de relog, el registro escribe en el ciclo negativo y lee en el positivo.
- PCsrc:Selección entre las direcciones de memoria Resultado y pc+4.

Intermedias:

- PC': Dirección de memoria de la instrucción que sera cargada en el siguiente ciclo del reloj.

Salidas:

- PC:Posición de memoria de la instrucción ejecutándose en un ciclo de reloj especifico.

Explicación General: Este par de modulo recibe la siguiente posición de memoria de la instrucción o una posicion de memoria generada por un tipo de salto, dicha posición es escogida dependiendo de si se cumple los saltos o no, el registro carga el dato en el ciclo negativo del reloj y lee en el ciclo positivo.

5.4.10. Unidad de control

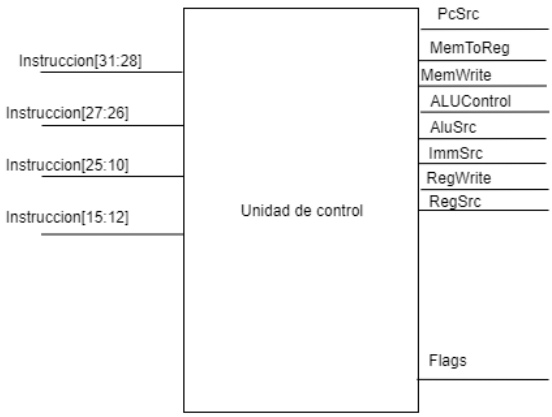

Figura 28: Unidad de control del procesador.

Table 7.2 Main Decoder truth table

Op	$Funct_5$	$Funct_0$	Type	Branch	MemtoReg	MemW	ALUSrc	ImmSrc	RegW	RegSrc	ALUOp
00	0	X	DP Reg	0	0	0	0	XX	1	00	1
00	1	X	DP Imm	0	0	0	1	00	1	X0	1
01	X	0	STR	0	X	1	1	01	0	10	0
01	X	1	LDR	0	1	0	1	01	1	X0	0
10	X	X	B	1	0	0	1	10	0	X1	0

Figura 29: Tabla de verdad de la unidad de control. Tomado de Harris y Harris [2]

Objetivo: Este modulo es el cerebro del procesador ya que a partir de las señales que entra genera todas las señales de control necesarias para ejecutar las diferentes instrucciones.

Entradas:

- Instrucción: Esto encapsula todas las entradas de la instrucción, estas son las que deciden que salidas se activan o no, todo esto depende de la instrucción que entra a la unidad de control.
- Flags: Estas son las flags generadas por la ALU, esta es utilizada en el caso de que se necesite realizar un salto en las instrucciones.

Salidas:

- PcSrc: Esta señal se encarga de la seleccion de donde proviene el siguiente PC.
- MemToReg: Se encarga de la seleccion del dato que sera escrito en un registro del banco de registros.
- MemWrite:Permite que se escriba en memoria.
- AluControl: Permite la seleccion de que operacion tanto aritmetica como logica que se aplicara en la ALU.
- AluSrc: Permite la seleccion de uno de los operandos de la ALU.

- ImmSrc: Este s el que permite seleccionar que tipo de extension se debe aplicar al inmediato de la instruccion a ejecutar.
- RegWrite: Es el que permite escribir en la memoria de registros
- RegSrc:Permite la seleccion de las direcciones de los registros a decodificar en el banco de registros.

Explicación General: Como se menciono anteriormente este es el cerebro del procesador, ya que genera todas las señales de control para la ejecución de instrucciones, incluido las decisiones de la realización de datos.

A continuación se muestran todos los componentes que forman parte del módulo presentado en el diagrama de segundo nivel como Renderer.

5.4.11. Controlador PS/2

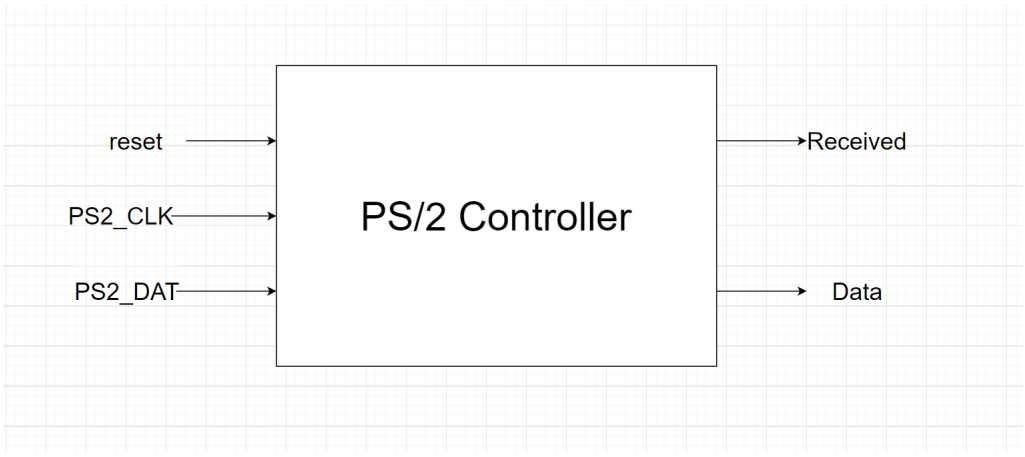

Figura 30: Implementación del controlador PS/2 para conectar el teclado

Objetivo: Interpretar las señales provenientes del teclado para que puedan ser utilizadas por la unidad de procesamiento.

Entradas:

- PS2_CLK: Señal de reloj para la sincronización de lectura.
- PS2_DAT: Flujo de datos que define el inicio y fin de una tecla presionada además de la tecla que fue presionada.

Salidas:

- Recieved: valor lógico estabilizado que indica cuando un valor es recibido del teclado;
- Data: valor lógico que especifica cual tecla fue presionada;

Explicación General: Este módulo representa una interfaz de las señales provenientes del teclado, acoplándolas y poniéndolas a disposición del procesador; esto se logra pues el procesador escucha la señal de interrupción y se detiene para leer el valor de la tecla presionada. La salida de este módulo es esencial para la interacción del procesador con el exterior pues maneja los periféricos de entrada.

5.4.12. Sprite Memory

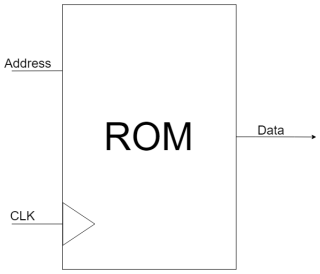

Figura 31: Implementación del almacenamiento del Sprite mediante una ROM

Objetivo: Permite almacenar la representación del Sprite de los jugadores.
Entradas:

- Addr: esta señal lógica de 10 bits corresponde a la dirección de la memoria que se desea acceder.

Salidas:

- Data: esta señal corresponde al dato almacenado en la posición seleccionada en addr.

Explicación General: Esta memoria de lectura contiene un Sprite de 32 x 32 pixeles, dado que es una memoria, recibe una dirección de lectura y obtiene el dato almacenado en esta. El sprite se encuentra distribuido en la memoria por líneas, es decir, las primeras posiciones contienen todos los píxeles de la primera linea del sprite, posteriormente la segunda y así sucesivamente.

5.4.13. Sprite-Position Comparator

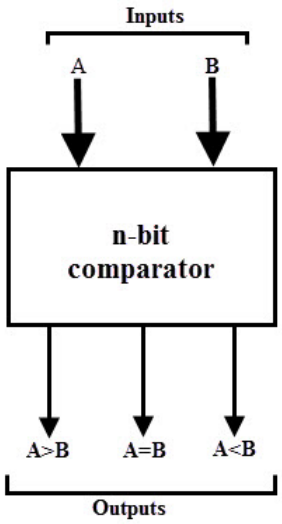

Figura 32: Comparador de Sprite

Objetivo: Determinar si en la posición entrante se debe dibujar un sprite.
Entradas:

- row: posición vertical en la que se encuentra el pixel a dibujar

- column: posición horizontal en la que se encuentra el pixel a dibujar
- game_state: estado del tablero para así determinar si se debe dibujar un sprite o no.
- data: datos del sprite almacenados en la memoria en la posición estipulada.

Salidas:

- Address: posición del pixel en la memoria, que se desea dibujar.
- RGB: información de la composición del pixel que se dibujará en la pantalla.
- visible: valor lógico de 1 bit que define si en la posición definida se debe dibujar un sprite o no.

Explicación General: Esta componente realiza una comparación de la posición en la que se dibujará un píxel y determina si este posición se encuentra en un rango donde se deba dibujar un sprite

---

5.4.14. Selection-Position Comparator

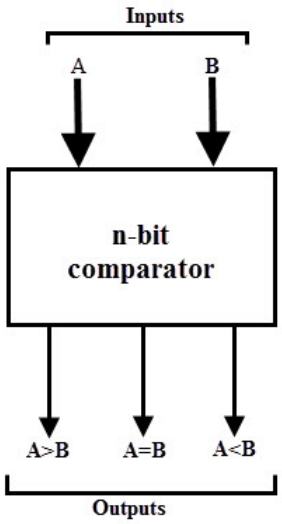

Figura 33: Comparador de selección

Objetivo: Determinar si en la posición entrante se debe dibujar un color distinto en el fondo para determinar que se encuentra seleccionado por el usuario.

Entradas:

- row: posición vertical en la que se encuentra el pixel a dibujar
- column: posición horizontal en la que se encuentra el pixel a dibujar
- game_state: estado del tablero para así determinar si se debe dibujar un sprite o no.
- data: datos del sprite almacenados en la memoria en la posición estipulada.

Salidas:

- Address: posición del pixel en la memoria, que se desea dibujar.
- RGB: información de la composición del pixel que se dibujará en la pantalla.
- visible: valor lógico de 1 bit que define si en la posición definida se debe cambiar el color de fondo o no.

Explicación General: Esta componente realiza una comparación de la posición en la que se dibujará un píxel y determina si este posición se encuentra en un rango donde se deba cambiar el fondo del tablero pues el jugador está seleccionando esta posición.

5.4.15. Line-Position Comparator

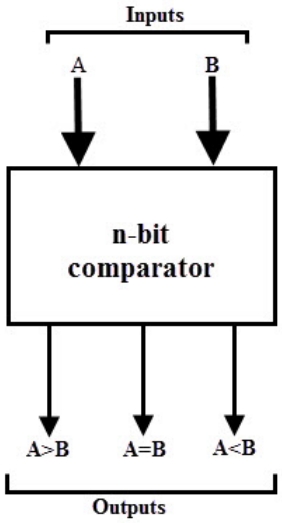

Figura 34: Comparador de linea

Objetivo: Determinar si en la posición entrante se debe dibujar una linea del tablero.
Entradas:

- row: posición vertical en la que se encuentra el pixel a dibujar
- column: posición horizontal en la que se encuentra el pixel a dibujar

Salidas:

- RGB: información de la composición del pixel que se dibujará en la pantalla, representa el color de la línea.
- visible: valor lógico de 1 bit que define si en la posición definida se debe dibujar una línea o no.

Explicación General: Esta componente realiza una comparación de la posición en la que se dibujará un píxel y determina si este posición se encuentra en un rango donde equivale a una linea del tablero por lo que se debe cambiar el color por mostrar.

5.4.16. Mux

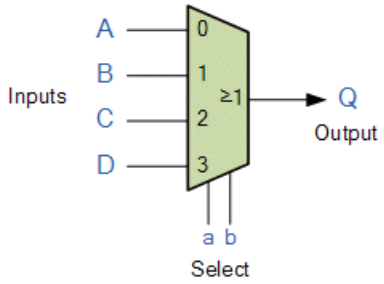

Figura 35: Multiplexor de informaicón del pixel

Objetivo: Decidir cual de las señales será mostrada en pantalla entre sus posibles entradas.
Entradas:

- RGB1: información de la composición del pixel que se dibujará en la pantalla, representa el color del sprite.
- RGB2:información de la composición del pixel que se dibujará en la pantalla, representa el color de la línea.
- RGB3:información de la composición del pixel que se dibujará en la pantalla, representa el color del fondo si este se encuentra seleccionado.

Salidas:

- VGA_R: bus de datos de 8 bits con la intensidad del color rojo del pixel seleccionado.
- VGA_G: bus de datos de 8 bits con la intensidad del color verde del pixel seleccionado.
- VGA_B: bus de datos de 8 bits con la intensidad del color azul del pixel seleccionado.

Explicación General: Este componente es de gran importancia y permite controlar y coordinar la información que está siendo enviada sobre cada pixel, a partir de este componente se conoce la posición que debe tomar cada pixel y además la señal de la fila y columna actual permiten a los demás componentes decidir que deberá mostrarse en dicha posición.

A continuación se muestran todos los componentes que forman parte del módulo presentado en el diagrama de segundo nivel como VGA controller.

5.4.17. Horizontal Counter

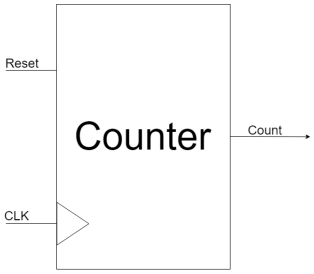

Figura 36: Contador horizontal

Objetivo: Contar el número de columna en que se encuentra el píxel actual.
Entradas:

- pixel_clk: reloj de píxeles, su frecuencia define la velocidad en que se envían los datos de cada uno de estos píxeles.

Salidas:

- VGA_HS: Esta señal es la encargada de la sincronización horizontal.
- column: Este valor numérico indica en qué posición horizontal se encuentra el píxel que se está evaluando.

Explicación General: Este componente es de gran importancia y permite controlar y coordinar de forma horizontal la información que está siendo enviada sobre cada píxel. La señal de sincronización horizontal se activa cada vez que se inicia una nueva fila, además indica cuando se debe reiniciar el conteo de la columna. La salida del valor de la columna, se relaciona con el componente anterior que determina que debe ser dibujado según la posición (fila y columna) dada.

5.4.18. Vertical Counter

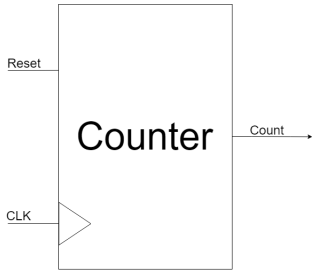

Figura 37: Contador Vertical

Objetivo: Contar el número de fila en que se encuentra el píxel actual.
Entradas:

- column_clk: reloj de columnas, esta señal se activa cada vez que el contador de columnas alcanza un valor específico, lo cual indica que debe haber un cambio de fila.

Salidas:

- VGA_VS: Esta señal es la encargada de la sincronización Vertical.
- row: Este valor numérico indica en qué posición vertical se encuentra el píxel que se está evaluando.

Explicación General: Este componente es de gran importancia y permite controlar y coordinar de forma vertical la información que está siendo enviada sobre cada píxel. La señal de sincronización vertical se activa cada vez que se inicia un nuevo frame, además indica cuando se debe reiniciar el conteo de la fila. La salida del valor de la fila, se relaciona con los componentes comparadores del renderer que determinan que debe ser dibujado según la posición (fila y columna) dada.

5.4.19. Comparator

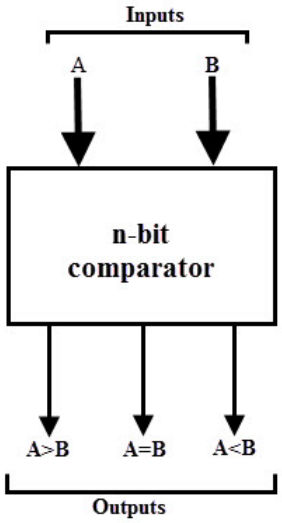

Figura 38: Comparador de columnas

Objetivo: Generar la señal de conteo de filas a partir del conteo de columnas.
Entradas:

- column_clk: Frequencia de cambio de columna, representa el movimiento horizontal.

Salidas:

- VGA_SYNC_N: esta señal no maneja ninguna señal de control o datos, solo debe ser utilizada durante el proceso que está activa la señal VGA_BLANK_N.
- VGA_BLANK_N: señal lógica que se activa cuando se encuentra fuera del rango visible para que el controlador no intente colocar los colores de esas posiciones.

Explicación General: Este componente realiza una comparación del valor de salida del contador horizontal, y una vez que este alcanza un valor específico, genera una señal que será utilizada para aumentar el contador vertical.

5.4.20. Clock Divider

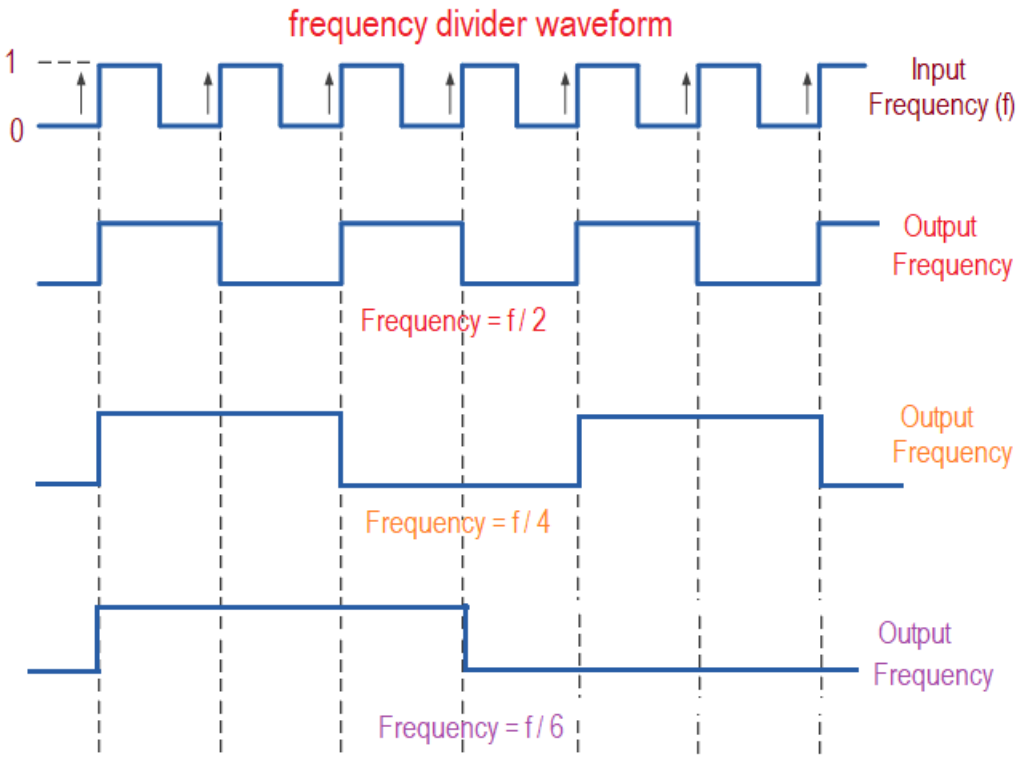

Figura 39: División de frecuencia de un reloj de ejemplo

Objetivo: Disminuir la frecuencia de operación del reloj base del sistema a la frecuencia apropiada para los pixeles.

Entradas:

- clk: Reloj base de aproximadamente 50MHz

Salidas:

- pixel_clk: Reloj de frecuencia apropiada para ser utilizada por el controlador VGA.

Explicación General: Este componente se comporta como un contador, que aumenta su conteo con cada flanco del reloj base, además de esto involucra un comparador encargado de generar una señal luego de una cantidad definida de ciclos, lo cual se representa en la salida como una disminución de la frecuencia del reloj. Basta con definir la cantidad de ciclos que deben transcurrir para modificar la división de la frecuencia.

5.5. Quinto Nivel

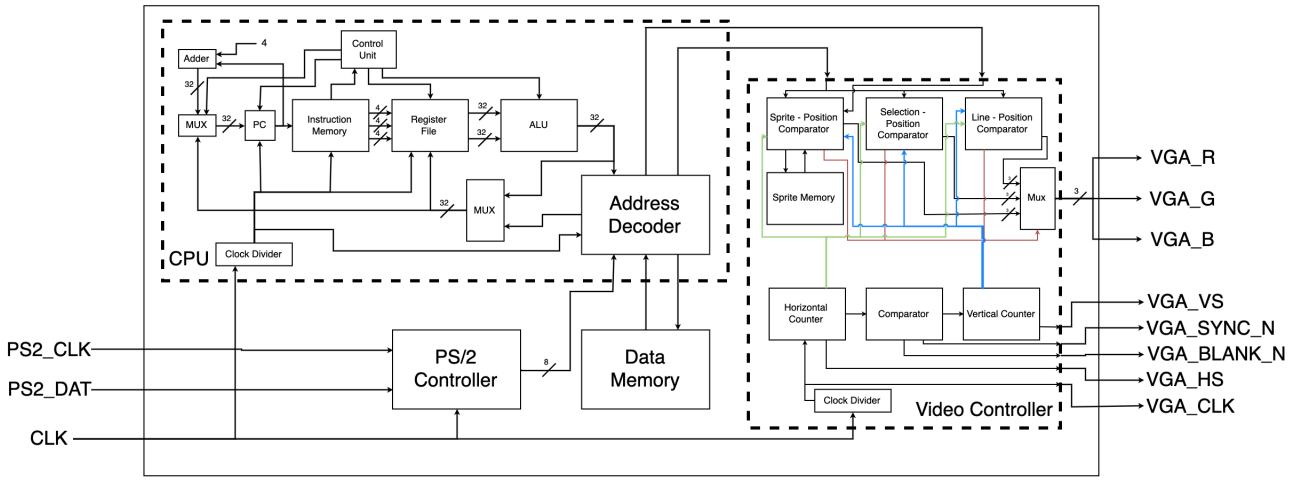

Figura 40: Diagrama de quinto nivel

Referencias

[1] Terasic De1-SoC User Manual

[2] Sarah Harris and David Harris. Digital Design and Computer Architecture: ARM Edition. Morgan Kaufmann, 2015.