Imagen que contiene dibujo, señal, reloj

Descripción generada automáticamente

**ING. en Computación**

**Seminario de Solución de Problemas de Arquitectura de Computadoras "D12"**

**Profesor: Jorge Ernesto López Arce Delgado**



***PROYECTO FINAL: PROGRAMACIÓN EN VERILOG Y LENGUAJE ENSAMBLADOR***

**Equipo:**

**Estrada Huerta Félix Eduardo**

**Ortega Morales Juan José**

**González Ramírez Carlos Arturo**

**01 - Junio - 2021.   
Guadalajara, Jalisco**

**CARACTERISTICAS GENERALES DEL PROCESADOR MIPS DE 32 BITS**

MIPS (Microprocessor without Interlocked Pipeline Stages), se conoce como MIPS a toda una familia de microprocesadores de arquitectura RISC desarrollados por MIPS Technologies.

El origen de la arquitectura MIPS se remonta al año 1981. En la universidad de Stanford, donde un equipo liderado por John L. Hennessy comienza a trabajar con lo que sería el primer procesador MIPS. Con la idea de mejorar a gran escala el rendimiento de la máquina a través del uso de la segmentación .

Sintetiza las principales características de la arquitectura RISC, es una arquitectura simple y eficiente.

Un procesador MIPS es toda una familia de microprocesadores de arquitectura RISC. Es utilizado en Windows CE, routers Cisco, en la Nintendo 64, PlayStation, PlayStation 2 y PlayStation Portable.

***Organización de un MIPS***

* Unidad Aritmética y Lógica (ALU).
* Unidad Aritmética entera, operaciones de multiplicación y división.
* Unidad Punto Flotante.
* Coprocesador dedicado al manejo de memoria caché y virtual.

La arquitectura MIPS requiere que el software implemente algunas limitaciones en el diseño que está normalmente considerado parte de la implementación del hardware. Este papel presenta resultados experimentales en la efectividad de este procesador como un programa anfitrión.

***Registros***

Tiene 32 registros de 32 bits de propósito general (GPR)

32 registros de 32 bits de punto flotante (FPR)

Cuenta con instrucciones diferenciadas para GPR y FPR

***Tipos de datos***

Half Word (16 bits)

Word (32 bits)

Double Word (64 bits)

Simple precisión (32 bits)

Doble precisión (64 bits)

Los datos half Word y Word se cargan en GPRs y se completan con 0 o el signo

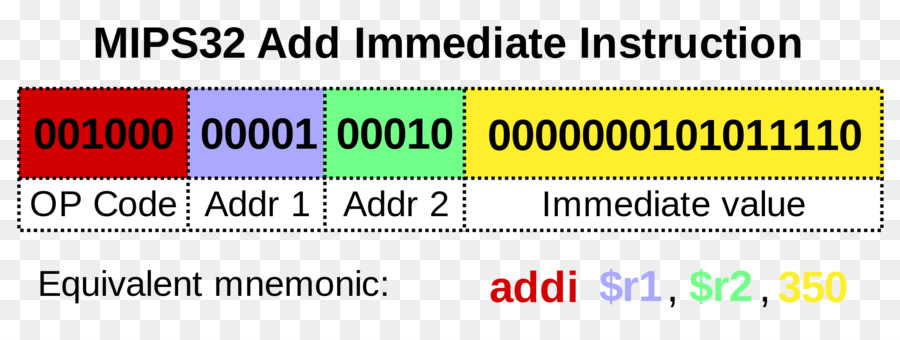
Las direcciones de memoria para cada instrucción son múltiplos de 4.

**SET DE INSTRUCCIONES**

Es una especificación que detalla las instrucciones que una CPU puede entender y ejecutar, o el conjunto de todos los comandos implementados por un diseño particular de una CPU. El término describe los aspectos del procesador generalmente visibles para un programador, incluidos los tipos de datos nativos, las instrucciones, los registros, la arquitectura de memoria y las interrupciones, entre otros aspectos.

Existen 3 principales tipos de set de instrucciones: CISC, RISC y SISC.

La arquitectura del conjunto de instrucciones (ISA) se emplea a veces para distinguir este conjunto de características de la microarquitectura, que son los elementos y técnicas que se emplean para implementar el conjunto de instrucciones. Entre estos elementos se encuentras las microinstrucciones y los sistemas de caché.



***TIPOS DE INSTRUCCIONES MIPS***

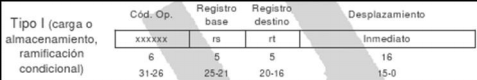
***Formato R***

Utilizado por las instrucciones aritméticas y lógicas



***Formato I***

Utilizado por las instrucciones de transferencia, las de salto condicional y las instrucciones con operando inmediatos



***Formato J***

Utilizado por las instrucciones de bifurcación



**TABLA DE INSTRUCCIONES MIPS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Instrucción* | *Tipo* | *Sintaxis* | *Descripción* | *Tabla Info* |
| Add | **R** | **Add $rd, $rs, $rt** | Suma Palabra (Add Word) | **∞** |
| Addi | **I** | **Addi $rd, $ro1, $inm** | Suma Palabra Inmediata (Add Word Inmediate) |
| Sub | **R** | **Sub $rd, $ro1, $ro2** | Resta palabra (Subtract Word) | **ℓ** |
| Mul | **R** | **Mul $rd, $ro1, $ro2** | Multiplica palabra (Multiply Word) | **Ω** |
| Div | **R** | **Div $ro1, $ro2** | Divide Palabra (Divide Word) | **∂** |
| Or | **R** | **Or $rd, $ro1, $ro2** | O (Or) | **∆** |
| Ori | **I** | **Ori $rd, $ro1, $inm** | O Inmediato (Or Inmediate) |
| And | **R** | **And $rd, $ro1, $ro2** | Y (And), Pone el registro | **∑** |
| Andi | **I** | **Addi $rd, $ro1, $inm** | Y Inmediato (And inmediate) |
| slt | **R** | **slt $rd, $ro1, $ro2** | Activa si menor (Set on Less Than) | **∫** |
| Slti | **I** | **slti $rd, $ro, $inm** | Activa si menor inmediato (Set on Less Than Inmediate) |
| not | **R** | **not $rd, $ro** | No (Not) | **∩** |
| Lw | **I** | **Lw $rd, $dir** | Carga Palabra (Load Word) | **ⱴ** |
| Sw | **I** | **Sw $ro, $dir** | Almacena Palabra (Store Word) | **ⱷ** |
| beq | **I** | **beq $ro1, $ro2, $etiq** | Bifurcación si igual (Branch on equal) | **●** |
| bne | **I** | **bne $ro1, $ro2, $etiq** | Bifurcación si no igual (Branch on not equal) |
| bgtz | **I** | **bgtz $ro, $etiq** | Bifurcación si mayor que cero (Branch on greater or equal than zero) | **○** |
| j | **J** | **j $etiq** | Salto (Jump), | **◊** |

|  |  |
| --- | --- |
| Simbología | **Significado** |
| ∞ | Suma el valor del registro ro1 y el registro ro2 (o el valor inmediato de 16 bits inm) y deja el resultado en el registro rd. Se consideran los operandos en complemento a 2, y cuando se produce un desbordamiento no se modifica el registro rd y se provoca una excepción del procesador |
| ℓ | Pone en el registro rd el resultado de calcular la resta entre los valores de los registros ro1 y ro2. Si la instrucción se consideran los operadores en complemento a 2, y cuando se produce desbordamiento no se modifica el registro rd y se provoca una excepción del procesador |
| Ω | Multiplica el valor de los registros ro1 y ro2 y deja los 32 bits menos significativos del resultado en el registro rd. Las instrucciones mul realizan multiplicación en complemento a 2. |
| ∂ | Divide el valor del registro ro1 entre el valor del registro ro2. Ambos valores se consideran cantidades enteras con signo. El cociente de la división se almacena en el registro L0 y el resto de la división en el registro HI. Esta instrucción nunca provoca una excepción |
| ∆ | Pone en el registro rd el resultado de realizar una operación or a nivel de bit entre los registros ro1 y ro2. En el caso de la instrucción ori entre el registro ro y el valor de 16 bits inm extendido con ceros |
| ∑ | Pone en el registro rd el resultado de realizar una operación *and* a nivel de bit entre los registros ro1 y ro2 (en caso de la instrucción and) o entre el registro ro y el valor de 16 bits inm extendido con ceros (Instrucción Andi) |
| ∫ | Pone el valor 1 en el registro rd, si los valores de los registros ro1 y ro2 cumplen la correspondiente condición. En otro caso pone el registro rd a cero. Si la comparación implica algún tipo de orden se considera que los números se interpretan en complemento a 2. |
| ∩ | Pone el registro rd el resultado de realizar una operación not a nivel de bit sobre el registro ro |
| ⱴ | Carga el valor de 4 bytes almacenado a partir de la posición de memoria dir en el registro rd. Si la dirección dir no está alineada a nivel de palabra (dirección múltiplo de 4) se produce una excepcion |
| ⱷ | Almacena a partir de la posición de memoria dir el valor de 4 bytes que se encuentra en el registro ro. Si la dirección dir no está alineada a nivel de palabra (dirección multiplo de 4) se produce una excepción. |
| ● | Salta, si los valores de los registro ro1 y ro2 cumplen una relación, a la instrucción marcada con etiq |
| ○ | Salta, si el valor del registro ro cumple una relación con respecto al valor cero, a la instrucción marcada con etiq |
| ◊ | Salta a la dirección marcada con etiq |

**DESARROLLO DE DATA PATH**

Una Data Path es una colección de unidades funcionales como unidades lógicas aritméticas o multiplicadores que realizan operaciones de procesamiento de datos, registros y buses.

Junto con la unidad de control, compone la unidad central de procesamiento (CPU). Se puede crear una ruta de datos más grande uniendo más de una ruta de datos utilizando multiplexores. Una ruta de datos es la ALU, el conjunto de registros y los buses internos de la CPU que permiten que los datos fluyan entre ellos.

Una ruta de datos de microarquitectura organizada alrededor de un solo bus, el diseño más simple de una CPU utiliza un bus interno común. La utilización eficiente requiere una estructura de tres buses internos un poco más complicada. Muchas CPU relativamente simples tienen un archivo de registro de 2 lecturas y 1 escritura conectado a las 2 entradas y 1 salida de la ALU.

A fines de la década de 1990, hubo una creciente investigación en el área de las rutas de datos reconfigurables (rutas de datos que se pueden reutilizar en tiempo de ejecución utilizando una estructura programable), ya que dichos diseños pueden permitir un procesamiento más eficiente y ahorros de energía sustanciales.

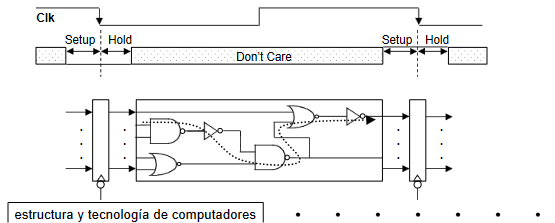
***Temporización Monociclo***

Ejecución típica (de una instrucción)

* Todos los registros se cargan simultáneamente (de modo selectivo)
* Todos los valores se propagan a través de las redes combinacionales hasta estabilizarse  
  en las entradas de los registros
* Se repite indefinidamente el proceso

Todos los elementos de almacenamiento están sincronizados al mismo flanco de reloj:

* Tiempo de ciclo = CKL-to-Q + Camino con retardo máximo + Setup + Clock Skew
* ( CLK-to-Q + Camino con retardo mínimo - Clock skew ) > Hold



**MEMORIA DE INSTRUCCIONES Y MEMORIA DE DATOS**

***Banco de Registros:***

|  |  |  |
| --- | --- | --- |
| Registro | Valor | Instrucción |
| *20* | *3* | *SUM* |
| *21* | *1* | *SUB* |
| *24* | *0* | *AND* |
| *25* | *10* | *OR* |
| *26* | *1* | *SLT* |
| *0* | *0* | *NOP* |

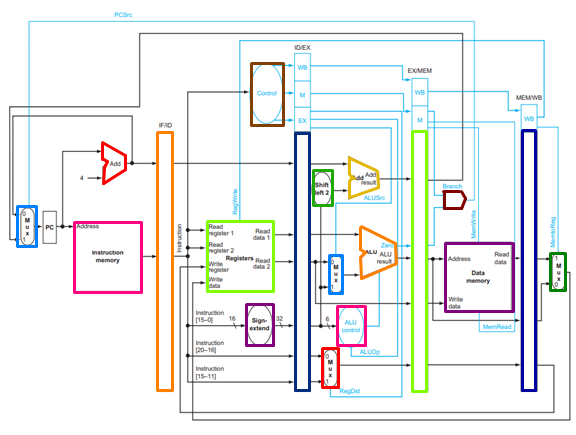
***Existe un error en una instrucción del archivo "TestF1\_MemInst.mem" debe identificarlo, anotarlo en el reporte y modificarlo para su correcto funcionamiento***

**REPORTE DE PROGRAMA EN VERILOG**

**FASE II:**

En la fase 2 del proyecto se implemento el uso de instrucciones de tipo I, instrucciones de sw, lw y beq en nuestro módulo creado en la fase 1, se agregó cuatro buffers para el uso de algunas señales (con las que así se requiere) lleguen al mismo tiempo.

La creación de estos módulos no fue de gran dificultad, la implementación de otro tipo de operaciones en el código en opinión personal, fue uno de los retos más complicados de realizar, al igual que cada una de las interconexiones de los módulos y el asegurar el correcto funcionamiento y traslado de los bits, así evitando una gran cantidad de errores y perdidas de datos.

 **Figura 1**

**CAND**

**Mux3**

**Buf4**

**Buf3**

**MEMO**

**ALUCONTROL**

**ALU**

**Adder2**

**shift**

**Buf2**

**UC**

**SignEx**

**BR**

**Buf1**

**InstMEMO**

**Mux2**

**Mux4**

**Mux1**

**Adder1**

En la figura 1 podemos ver el módulo completo y los nombres de cada módulo que usamos en nuestro proyecto, sólo para que sea más fácil entender el código al momento de verlo. El módulo que contiene todos estos módulos mostrados anteriormente tiene por nombre PF1 y el testbench se llama TB\_Prueba.

Para esta fase del proyecto, primero se necesito terminar el código de la fase 1 que no se termino por completo, cuando se terminó la implementación comprobé su funcionalidad con instrucciones de tipo R, se inicio agregando la instrucción Beq en el código, cuando se logro hacerlo funcionar esta instrucción en el módulo se añadio las instrucciones lw y sw, después de eso implementó las instrucciones de tipo I (addi, andi, slti, ori) y se volvió a probar todas estas instrucciones una por una en el código, posteriormente se añadieron los buffers y reconectaron todos los módulos de la forma que se muestra en la figura 1.

Cuando el módulo estaba listo nuevamente se realizaron las pruebas de las instrucciones individualmente, estos fueron los resultados:

Interfaz de usuario gráfica

Descripción generada automáticamente **Figura 2**

Primero, en la figura 2 podemos ver los datos precargados en el banco de registros.

**Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente Figura 3**

La primera instrucción que probé fue Nop, en la figura 3 podemos ver que no se realiza operación alguna y se almacena un 0 en la dirección 0.

**Interfaz de usuario gráfica

Descripción generada automáticamente Figura 4**

La siguiente instrucción fue addi, en la figura 4 se muestra la instrucción y el banco de registros con el nuevo dato. En la instrucción se indica primero la operación a realizar (010000), luego el registro operando (00100), luego el registro destino (10000) y por último 0000100010001100 que es el valor inmediato que se va a sumar.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente **Figura 5**

La siguiente fue Slti, en la figura 5 podemos ver que el resultado de la operación se almacena en la dirección 17, el código es el siguiente: 001010 (para la operación), 00101 (para operando 1), 10001 (dirección destino) y 0011100111100000 como valor inmediato, al ser menor el operando 1 que el valor inmediato, se almacena un 1.

**Interfaz de usuario gráfica

Descripción generada automáticamente Figura 6**

La próxima operación es andi, el resultado se va a almacenar en la dirección 18, como lo podemos ver en la figura 6, los datos que se van a operar son el de la dirección 9 (00000000000000000000000011111110) y el valor inmediato 0110110011110001.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente **Figura 7**

La próxima instrucción es ori, el resultado se almacena en la dirección 19 (ver figura 7), el operando 1 es el contenido de la dirección 15 del banco de registros (00000000000000000000000000101111) y el otro operando es el inmediato 0011100101010011.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente **Figura 8**

Como vamos a ver las instrucciones lw y sw, en la figura 8 podemos ver los datos que hay precargados en nuestra memoria de datos.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente ***Figura 9***

En la figura 9 podemos ver la operación lw, la dirección base es 0, la dirección destino es la 4 y offset es igual a 6, entonces, el dato que se almacena en el banco de registros en la dirección 4 es el dato que se encuentra en la dirección 6 de la memoria de datos.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente **Figura 10**

La siguiente operación es sw, se muestra en la figura 10, la dirección base también es la 0, la dirección del banco de registros cuyo valor se va a guardar en la memoria de datos es la 8, el valor se va a guardar en la dirección 1.

Interfaz de usuario gráfica

Descripción generada automáticamente **Figura 11**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente **Figura 12**

La última operación es Beq, en la figura 11 podemos ver que se realizaron todas las instrucciones que expliqué de forma secuencial, la primera instrucción que agregué en esta simulación fue un beq que va a comparar el registro 0 con el 0 y como son iguales se va a saltar dos instrucciones, estas dos instrucciones que se va a saltar tendrían que almacenarse en las direcciones 1 y 2 (en caso de que no se saltaran) y como se puede ver en la figura, los datos en estas dos direcciones permanecen iguales que al principio. La figura 12 muestra la memoria de datos para comprobar que se realizó también la operación sw.

**CONCLUSIÓN DE FASE**

La realización de esta tarea fue algo compleja y complicada, en especial por el hecho de tener el tiempo contado al tener el atraso de la fase anterior, ya que se necesitaba complementar y lograr funcionar el código al 100%, no fue sencillo ni complejo, ya que todos tenemos una idea base de lo que se necesita realizar en cada uno de los módulos, pero al mismo tiempo, el plasmar las ideas es un tema complicado.

Al terminar la fase 1, no se perdió el tiempo y directamente se comenzó a realizar la implementación de la fase 2, la cual en opinión personal fue de las más complicadas, ya que a diferencia de la fase 1 se necesitaba realizar la implementación de mayor cantidad de módulos, la mayoría de las de los errores que se encontraron en el desarrollo de este proyecto fueron al momento de realizar las interconexiones de los módulos, fue un gran reto, pero al final se logró terminar con buenos resultados. Dejando mucho aprendizaje de este.