**ARQUITECTURA PROCESADOR MONO-CICLO**

**Next Program Counter (NPC):** Es un registro de 32 bits que lo que hace es permitir colocar en la salida el valor que esta en la entrada cada vez que hay un ciclo de reloj, típicamente es un valor de 32 bits, normalmente el NPC se ira incrementando de 4 en 4 cuando nuestra instrucción así lo diga o simplemente se hará un salto de 0 en 0.

**Program Counter (PC):** Es un dispositivo igual al NPC se puede crear un instancia de este, el NPC mantiene el siguiente valor al cual va a saltar el PC y el PC counter mantiene el valor de la instrucción que se esta siendo ejecutada en ese momento en el procesador. En grandes rasgos ambos son unos registros de 32 bits que nos van a permitir indicarle al procesador en que instrucción va.

**Sumador 32 bits:** Una de sus entradas es el NPC y otra entrada es un valor quemado este ultimo depende de cuanto necesitamos incrementar nuestro PC porque el direccionamiento que se realiza dentro de la IM (Este tipo de procesador es un direccionamiento que se hace por “var” de tal manera que debemos hacer un incremento de 4 en 4. Un sumador toma dos valores un valor es el del NPC le suma 4 y ese valor vuelve y entra a nuestro NPC.

**Instruccions Memory(IM):** Es un elemento vital dentro de nuestra arquitectura porque nos va a permitir contener o mantener allí guardado el programa que nosotros deseemos ejecutar, allí están todo el conjunto de instrucciones que nosotros hemos construido una vez hemos realizado todo el proceso de traducción de un programa que esta escrito en un lenguaje de alto nivel y lo hemos llevado hasta lenguaje de maquina, la IM entonces nos va a permitir guardar todas esa instrucciones en lenguaje de maquina y básicamente el ancho de esa IM va a hacer de 32 bits o 4 bytes y el largo es muy dependiente de la implementación, típicamente en el curso se construye las IM de 32 bits por 32bits de alto para que podamos hacer un procedimiento simple sobre nuestro procesador y generalmente las instrucciones o programas que se realizan no superan los 32 instrucciones de tal manera que es un buen numero. Básicamente la IM no es mas que una matriz que contiene un conjunto de casillas, cada una de estas casillas serán entonces de 31 hasta 0 y por el otro lado desde 0 hasta 31 de tal manera que

podamos construir una memoria ROM que va a permitir guardar todas las instrucciones allí una sola vez y durante todo el tiempo de ejecución de nuestro programa no se podrá entonces modificar ese valor que esta ahí guardado en la IM. Es importante entonces ver que la entrada de nuestra IM va a hacer una dirección y la salida va a hacer un dato que esta almacenado en esa dirección especifica de tal manera que nuestra IM se convierte en una memoria ROM. De tal manera entonces que nuestra IM es una memoria que tiene 32 bits, esos 32bits entonces hacen referencia a la instrucción como tal, estos 32 bits deben ser repartidos en un conjunto de elementos adicionales que se desarrollaran mas adelante.

**Register File(RF):** Es un elemento vital donde tenemos para la arquitectura Sparc v8, 40 registros de alto y cada registro puede guardar un valor de 32bits. En el RF es entonces donde se empieza a jugar con el ventaneo de dicha arquitectura, para este caso especifico se va a utilizar una sola ventana y lo que va a pasar en este caso es que cada uno de los elementos que son llamados RS1, RS2, RD son los registros que se van a entrar a buscar dentro el RF, cada uno de ellos siendo de un tamaño de 5bits de tal manera se debe saber que si de la IM salen 32 bits, se sabe que por ejemplo el RD van a hacer los bits desde el 29 hasta 25, RS1 van a hacer los bits desde el 18 hasta el 14 y RS2 van a hacer los bits del 4 al 0. de igual manera el RF se convierte en una especie de memoria RAM.

**UC(Unidad de Control):** Es el elemento de mayor importancia dentro de nuestro procesador y es el que le da sentido como tal a él, porque gracias a este elemento es que las nuestra arquitectura se puede ver como una arquitectura de propósito general, esta unidad es la que permite tomar todo un conjunto de decisiones partiendo de los diferentes valores de entrada que tenga, para este primer caso de acercamiento a la arquitectura Sparckv8 va a ser un elemento que tiene solamente dos entradas que son entonces el OP que viene siendo de los bits 31 al 30 y el OP3 que son los bits del 24 al 19 de tal manera con esos dos elementos la UC puede determinar que instrucción aritmética o que instrucción lógica quiero yo ejecutar dentro de mi arquitectura.

**ALUOP:** Es una señal que nos va a permitir a nosotros seleccionar en la ALU que tipo de instrucción queremos realizar como por ejemplo si queremos realizar una suma una resta, una and, una or, etc, ese valor entonces va a hacer un valor que nosotros mismos podemos utilizar le podemos decir entonces que para realizar un suma vamos a utilizar un ALUOP= “0000”, eso dependerá entonces de las necesidades y la cantidad de instrucciones que que nosotros queramos implementar en nuestra arquitectura.

**SEU(Unidad de extensión de Signo):** Esta lo que hace es tomar el campo de 13 bits dentro de nuestra instrucción una vez sale de la IM y lo que va a hacer es tomar esos 13 bits y realizar un proceso de extensión de signo (Es decir él verifica si el primer valor suponiendo que se tiene el número 11110 y se quiere realizar un proceso de extensión, lo que se hace entonces es tomar el numero como esta y suponiendo que se necesita completar o extender este valor hasta un valor de 32 bits lo que vamos a hacer es dado que el primer elemento de este numero es un 1 por ende se sabe que es un numero negativo en complemento A2 y se va completar el valor que necesitemos en este caso son 32bits con unos a la izquierda, lo cual va a mantener el signo y ademas la magnitud del valor que necesitamos reemplazar. De igual manera sucede si se necesita hacer una extensión de un numero positivo se verifica si el primer valor de el numero es 0 se llena con ceros a la izquierda.

**MUX(MULTIPLEXOR):** Es un elemento vital dentro de todas las arquitecturas de computadores, el mux es básicamente un elemento que me coloca en la salida el valor de alguna de las entradas de acuerdo a una entrada adicional que es conocida como el selector. En este caso él elige entre los dos valores que están en el RF (CRS1, CRS2) , el valor CRS1 Entra directamente a la ALU sin ningún problema pero el segundo operando (CRS2) puede cambiar porque este puede ser el contenido como tal del RF o puede ser un valor inmediato, en el caso de que sea un valor inmediato se debe dejar pasar lo que es la salida de la SEU. Siendo así entonces para nuestro caso en especifico vamos a suponer que si el selector vale 0 dejo pasar entonces el CRS2 a la ALU y si por el contrario es un 1 dejo pasar entonces la salida de la SEU (Inmediato con extensión de signo). Si observamos en detalle nosotros tenemos nuestro campo i, este campo es el bit 13 de nuestra instrucción que sale de la IM el cual define si existe o no valor inmediato siendo así entonces, éste bit termina convirtiéndose en el selector del MUX. La salida del mux se convierte en ultimas en el valor que entra a la ALU, para que ésta pueda operar entre el CR1 y el IMM si i=1 o el CRS2 si i=0.

**Elementos que salen del RF:**

**CRS1(Contenido registro fuente 1)**

**CRS2(Contenido registro fuente 2)**

Estos dos contenidos entonces van a hacer los que ingresan a la ALU y nos van a permitir ya sea sumarlos, restarlos o hacerles algún tipo de instrucción que sea aritmética o lógica en este caso especifico.

**ALU(Unidad Aritmético Lógica):** Es en esta entonces donde se puede realizar todas las instrucciones u operaciones sobre diferentes contenidos de elementos o datos que estén guardados en el RF.

Finalmente a la salida de la ALU vamos a tener un elemento llamado DWR(dato que debe escribir en el RF) porque el RF tiene un RD que es donde voy a guardar el resultado de la operación que estoy realizando entre dos contenidos de dos registros.

**CONCEPTOS VHDL**

# **std\_logic\_unsigned:**

This library defines all of the same arithmetic (+, -, \*), comparison (<, <=, >, >=, =, /=) and shift (shl, shr) operations as the std\_logic\_arith library. This difference is that the extensions will take std\_logic\_vector values as arguments and treat them as unsigned integers (ie. just like type unsigned values).

**Conv\_integer:**

The function conv\_integer is also defined on std\_logic\_vector and treats the value like an unsigned integer:

function conv\_integer(arg: std\_logic\_vector) return integer;

**Impure Fuction:**

An impure function is a function that mutates variables/state/data outside of it’s lexical scope.

**Process:** Describe el comportamiento de un circuito cuyo estado puede variar cuando cambian ciertas señales, utiliza señales muy expresivas como: if, then, else, for while, etc. aplicable a lógica combinacional y secuencial.

**process (lista de señales)**

**parte declarativa(variables, procedimiento, tipos, etc.)**

**begin**

**Instrucciones de que describen el comportamiento.**

**End process;**

**A <= B (Asignación de señal)** asignación de señales: después de ejecutarse esta línea, "a" seguirá valiendo lo mismo, sólo se actualiza al acabar el PROCESS

Z := A or C; asignación de variables: en el momento de ejecutarse esta línea z valdrá "a or c" (el valor que tenía a cuando empezó el PROCESS)