1. Arquitectura del µprocesador

El micro-procesador poseerá un core de 32 bits con memorias de datos e instrucciones separadas, instanciadas además en el testbench en lugar de dentro del mismo proyecto, puesto que empeorarían mucho el estudio en frecuencia del diseño, siendo en realidad un tema muy forzado por la tecnología empleada, no por el diseño creado. Así mismo posee un banco doble de 16 registros para permitir dos lecturas simultáneas además de una escritura. Y también posee un juego de instrucciones de tamaño fijo pero estructura diferente según el modo de direccionamiento y la instrucción en concreto.

A la hora de elegir la posición de los campos en cada tipo de trama, se tuvo en cuenta cuales iban a ser utilizados y cuáles no en cada caso, principalmente para no tener que ir cambiando su posición en otras configuraciones que compartiesen ese campo (rd, funct y rt). En las siguientes tablas se pueden observar la distribución de los bits en la trama de instrucción:

Nº bits:	3	1	4	4	3	4	13
Tipo-R:	opcode	I	rd	rs	funct	rt	sin uso

Nº bits:	3	1	4	4	3	17
Tipo-I*:	opcode	I	rd	rs	funct	dirección/inmediato

Nº bits:	3	1	4	4	20
Tipo-X*:	opcode	I	rd	rs	dirección/inmediato

Nº bits:	3	1	4	24
Tipo-Y*:	opcode	I	rd	dirección/inmediato

Nº bits:	3	1	28
Tipo-J*:	opcode	I	Dirección/inmediato

*Cada una de las distribuciones de tipo inmediato corresponde a un tipo determinado con 2 bits que luego se utilizan para identificarlas en el micro-controlador: I=00, X=01, Y=10, J=11

2. Codificación

A la hora de elegir la codificación del opcode y de las funciones aritmético-lógicas (definidas por el campo funct) , se tuvo que reeditar la estructura a lo largo del diseño por varios motivos. Los saltos condicionales comparten todos un bit que los diferencia del resto de instrucciones para utilizarlo directamente en las señales de control. Del mismo modo y para ahorrar bits de programa, las funciones aritmético-lógicas comparten opcode y utilizan un campo funct. Así mismo para que las funciones fueran 8 y cupiesen en 3 bits se aprovechó que las órdenes load y store, que no necesitaban usar el bit que diferenciaba funciones inmediatas de no-inmediatas, y se unieron en el mismo opcode diferenciándolas precisamente con ese bit. También fue

preciso hacer un cambio en la codificación del opcode debido a que se eligió para la dirección 000 la función beq, que al resetear el micro-controlador provocaba que al ser los operandos iguales (eran 0 debido al reset), siempre ejecutaba la orden de salto y no llegaba nunca a funcionar.

Para decidir el código del campo funct sólo cabe comentar que al ser requeridas sólo 6 funciones, sobraban dos códigos válidos que no tenían implementación. Entonces se decidió hacer que esos códigos fueran dar una salida con los 32 bits a nivel lógico bajo para así aprovechar esta función en la ALU para generar un uno en la salida zero y así implementar mediante funct=000 ó funct=111 los saltos incondicionales. Todas las instrucciones tienen una función "funct" asociada, pero sólo las operaciones aritmético-lógicas indican dicha función en un campo de su trama, el resto las genera el decodificador de la fase ID según el opcode que reciba.

Tabla de Asignación de "opcode":	
bgt	000
blt	001
bne	010
beq	011
Op. Aritm-log.	100
lw/sw	101
nop	110
jmp	111

Tabla de Asignación de "funct":	
zero=1	000
add	001
sub	010
and	011
or	100
not	101
xor	110
zero=1	111

El bit "I" (4º bit) diferencia si es operación inmediata de no inmediata salvo para lw/sw que identifica precisamente si es lw o sw. Por defecto:

I=0 --> no inmediato/lw / I=1 --> inmediato/sw

El decodificador genera las siguientes señales de control:

RegWrite: Activa la entrada write del banco de registros.

MemtoReg: Indica si la salida de la orden proviene de la ALU (0) o de memoria (1).

MemWrite: Activa la entrada write de la memoria de datos.

Branch: Indica si la orden es un salto (1) o no (0).

funct: Indica a la ALU la operación a efectuar (sólo la genera si opcode no es 100).

tipo: Indica el tipo de inmediato, si es de 17 bits(00), de 20 bits(01), de 24 bits(10) o 28 bits(11).

ALUsrc: Indica si la operación es sobre un registro (0) o un inmediato (1) [en el caso de jump indica si se direcciona a inm+reg (1) o sólo a inm (0)]

mbs: Por defecto se encuentra siempre nivel bajo, salvo para los saltos condicionales bgt y blt, pues indica que la salida que se evalúa a la salida de la ALU para saltar es el bit de mayor peso en lugar de la salida zero.

negar: Por defecto se encuentra siempre a nivel bajo, salvo para los saltos condicionales bne y bgt, pues invierte la señal que indica si se produce el salto, puesto que para bne y bgt sería 0 en lugar de 1.

3. Segmentación

El proceso del microprocesador se ha dividido en las cinco fases:

Instruction Fetch – Instruction Decode - EXecute - MEMory – Write Back