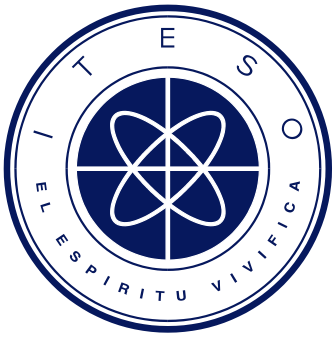
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Reconocimiento de validéz oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación el 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

MAESTRÍA EN DISEÑO ELECTRÓNICO



REPORTE DE FORMACIÓN COMPLEMENTARIA EN ÁREA DE CONCENTRACIÓN EN DISEÑO DE SISTEMAS DIGITALES

Trabajo recepcional que para obtener el grado de

MAESTRO EN DISEÑO ELECTRÓNICO

Presenta: Enrique Israel Hernández Meza

Asesor: Mtro. José Luis Chávez Hurtado

San Pedro Tlaquepaque, Jalisco. Septiembre de 2017.

Contenido

Instituto Tecnológico y de Estudios Superiores de Occidente i

1. Resumen de los proyectos realizados 6

1.1. DISEÑO DE MICROPROCESADOR *MIPS* 32 *BITS* BÁSICO MULTI-CICLO 7
1.1.1 Introducción 7
1.1.2 Antecedentes 7
1.1.3 Solución Desarrollada 8
1.1.4 Análisis de resultados 9
1.1.5 Conclusiones 9
1.2. DISEÑO DE UN MICROPROCESADOR SUPERESCALAR. 10
1.2.1 Introducción 10
1.2.2 Antecedentes 10
1.2.3 Solución Desarrollada 11
1.2.4 Análisis de resultados 11
1.2.5 Conclusiones 12
1.3. PROPUESTA PARA DETERMINAR EL CÁLCULO DEL TAMAÑO IDEAL DE BLOQUE DE CACHÉ 12
1.3.1 Introducción 12
1.3.1 Antecedentes 13
1.3.2 Solución Desarrollada 13
1.3.3 Análisis de resultados 14
1.3.4 Conclusiones 14
1.4. CONCLUSIÓN GENERAL 15

Apéndices 17

A. DISEÑO DE MICROPROCESAR MIPS 32 BITS BASICO MULTICICLO 19
B. DISEÑO DE UN MICROPROCESADOR SUPERESCALAR 68
1.1. ARQUITECTURA PROPUESTA 72
1.2. ARQUITECTURA DEL BRANCH PREDICTOR & TABLE BUFFER 73
1.3. SET DE INSTRUCCIONES DEL MIPSR4000 A IMPLEMENTAR 74
1.4. LISTADO DE UNIDADES IMPLEMENTADAS DEL PROYECTO 76
1.5. LISTADO IMPLEMENTACIÓN DE MÓDULOS EN RTL 76
1.6. PROGRAMAS DE PRUEBA. 77
C. PROPUESTA PARA DETERMINAR EL CÁLCULO DEL TAMAÑO IDEAL DE BLOQUE DE CACHÉ 90

Introducción

El área de concentración elegida para la obtención del grado de Maestro en diseño electrónico fue la de Diseño de Sistemas Digitales, la razón por la cual se decidió esta rama obedece a mi interés por trascender en la creación de soluciones en el diseño de algoritmos en hardware y software.

Es importante remarcar que las materias elegidas, además de sustentar el marco teórico para el análisis, diseño, arquitectura e implementación, también otorgaron la experiencia práctica a través de los proyectos propuestos que permitieron el desarrollo de nuevas soluciones en trabajos posteriores dentro de las empresas para las cuales he trabajado, algunos de ellos son:

En *Intel Corporation*, impactó trascendentalmente en la profundización del conocimiento requerido para aportar soluciones complejas de alta tecnología e influyó significativamente en la creación de arquitectura y desarrollo de procesadores X86 *multi-core, y many integrated core* de alto rendimiento conocidos como *Xeon y Xeon Phi*.

También influyó en mis habilidades en el desarrollo e implementación de algoritmos de cómputo visual en hardware para la realización de un procesador prototipo de reconocimiento de imágenes y análisis de video que enmarcó las bases para sustentar la posición de líder de grupo para el desarrollo de algoritmos de *Advanced Driver Assistance Systems (ADAS)* en la compañía *Continental Automotive Group*.

Y, actualmente en la compañía *Future Mobility Corporation* como gerente de desarrollo de funciones *ADAS* para vehículos autónomos eléctricos; las habilidades adquiridas en esta área influyen fuertemente en la arquitectura de los sistemas.

Las materias que se cursaron son las siguientes:

Diseño de Sistemas Digitales impartida por el Dr. Mariano Aguirre.

Diseño de Microprocesadores impartida por el Mtro. German Fabila García.

Arquitectura de Microprocesadores impartida por el Dr. Marcos de Alba.

1. Resumen de los proyectos realizados

Diseño de Sistemas Digitales.

Se trabajó en la realización de un microprocesador MIPS Básico Multi-ciclo, el resultado final, mostró la implementación en físico de dicho procesador MIPS en un *FPGA Spartan II*, con etapa para el manejo de datos externo (teclado) y externo (visualización de resultados en pantalla LCD), y al cual se le proporcionó un programa en ensamblador que realizaba las funciones de una calculadora aritmética hexadecimal. Dicho programa era fácilmente intercambiable por medio de una memoria NVRAM y enmarcó las bases para la utilización de técnicas de optimización para mejorar el desempeño de las arquitecturas y algoritmos que favorecen la velocidad y la potencia de cómputo.

Diseño de Microprocesadores.

Conceptualizó la realización de un microprocesador superescalar de arquitectura avanzada que implementó el conjunto de instrucciones *MIPS R4000*. El resultado final fue la entrega del microprocesador superescalar implementado en un FPGA Virtex6; siendo lo más enriquecedor del proyecto final, la etapa de optimización y desempeño de las unidades de predicción y saltos (*branch prediction*), de control especulativo y buffers de reordenamiento (*ROB*), así como la preparación para el manejo de protocolos de coherencia en memorias *cache* y diseño de anillos.

Arquitectura de Microprocesadores.

El trabajo final se centró en el desarrollo de una propuesta para determinar dinámicamente el cálculo del tamaño ideal de un bloque de memoria *cache* utilizado en un microprocesador experimental y cuya inclusión del algoritmo se realizó en la etapa de *issue, distpatch, y writeback;* dentro del microprocesador experimental, nos permitió caracterizar los comportamientos de las diferentes micro-operaciones y sus llamadas a memoria *cache* y con ello desarrollar un algoritmo dinámico el cual en un trabajo posterior se incluyó en el microprocesador superescalar realizado en la materia de Diseño de Microprocesadores.

1.1. Diseño de microprocesador *MIPS* 32 *Bits* Básico Multi-ciclo

El objetivo final fue la implementación en un FPGA de un microprocesador MIPS 32 Bits Básico Multi-ciclo basado en el *Instruction Set Architecture R2000*, y el desarrollo de un programa en ensamblador que realizara las funciones de una calculadora aritmética hexadecimal.

1.1.1 Introducción

Un microprocesador es un sistema de cómputo digital compuesto por la interconexión de módulos de propósito específico, y cuya organización interna está compuesta por un conjunto de registros que contienen los datos y las micro-operaciones a realizar. Las funciones de control inicializan cada una de las secuencias ya sean transferencias de registros, aritméticas, lógicas, de corrimiento, de predicción, entre otras.

El ciclo básico de una instrucción se compone de las siguientes fases:

1. Obtener una instrucción de la memoria (*fetch*)
2. Identificar la operación a realizar (*Decode*)
3. Obtener el dato de memoria, si la instrucción tiene un acceso indirecto (*Read*)
4. Ejecutar la instrucción (*execute*)
5. Almacenar el resultado en memoria (*write*)

1.1.2 Antecedentes

Las etapas en un procesador uní-ciclo se ejecutan en un solo ciclo máquina, que puede contener uno o más ciclos de reloj e implica que dichos módulos utilicen lógica combinacional comportamental y consideren el tiempo de latencia para mover los datos de una etapa a la siguiente, aumentando la dificultad en el diseño, ya que el dato tiene que estar listo antes de que se requiera utilizar durante su ejecución en la etapa. El rendimiento de estos microprocesadores es pobre para la ejecución secuencial de programas complejos pero fueron base fundamental de nuevas mejoras y arquitecturas.

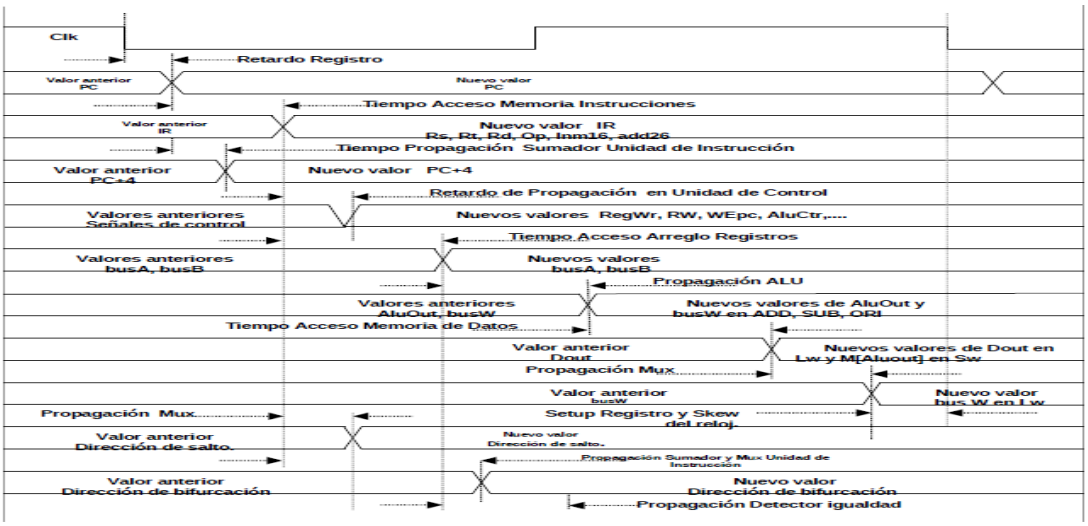


Figura 1-1 Diagrama de tiempo de las operación en un MIPS básico uní-ciclo

La idea fundamental de la implementación de un microprocesador MIPS (por sus siglas en inglés de *Microprocessor without Interlocked Pipeline Stages*) basados en arquitectura RISC es la de mejorar drásticamente el rendimiento mediante el uso de la técnica de segmentación formada por las mismas 5 etapas, bajo la premisa de que cada etapa se ejecuta durante un ciclo de reloj con lo cual se deja de tener dependencia de los tiempos de propagación de las señales y simplifica la lógica de control en comparación con la uní-ciclo y permite la ejecución de una nueva instrucción por etapa antes de que sea finalizada la ejecución de la instrucción anterior; otra ventaja adyacente es la incorporación de lógica que atienda saltos de brinco de programa de manera más eficiente.

1.1.3 Solución Desarrollada

El proyecto presentado divide la arquitectura en dos etapas principales, la de ruta de datos (*Data Path*) y unidad de control (*Control Path*).

La ruta de datos comprende: la implementación de las unidades de memoria, registros de lectura, escritura, y de instrucción, la lógico aritmética, los módulos de interconexión y multiplexado de ruta, los registros de corrimiento para la derecha e izquierda, la de operación de signo extendido y el contador de programa; todos ellos unidos por el módulo TOP de ruta de datos.

La ruta de control comprende: la implementación del módulo de la unidad de control, el cual proporciona la lógica para activación de cada módulo cuando es requerida la precarga de los datos y el funcionamiento de la transferencia de registros, los parámetros para los estados en el CPU y la parametrización de los códigos de operación en sincronía con el ciclo de reloj.

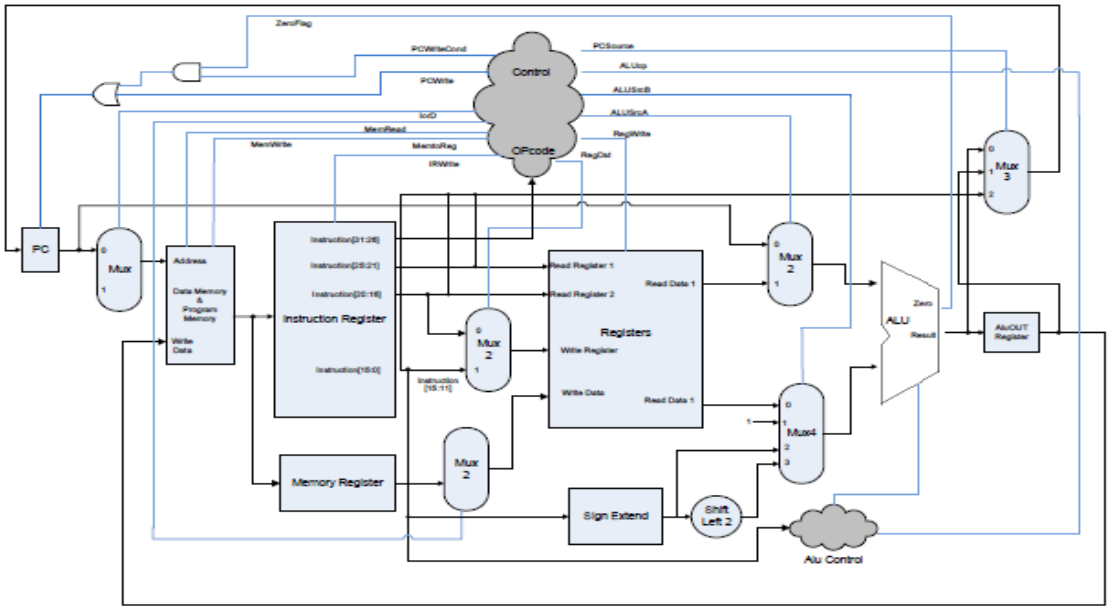


Figura 1-2 Arquitectura propuesta para la implementación del MIPS multi-ciclo de 5 etapas

En el apéndice A parte 1 microprocesador multi-ciclos se detalla el código RTL utilizado

1.1.4 Análisis de resultados

En el apéndice A parte 1 se muestra el resultado de la simulación e implementación de la cama de pruebas que se utilizó para verificar y validar el código, su funcionamiento y depurar la implementación antes de sintetizarlo en el FPGA; así mismo en el apéndice A parte 2 viene descrito el programa de una calculadora aritmética para operaciones en hexadecimal, con el cual se evaluó la implementación y se muestran los resultados que el FPGA ejecuta; siendo éstos los esperados por cada una de las operaciones que se efectuaron al momento de la evaluación en clase.

1.1.5 Conclusiones

Cabe destacar que el impacto en la realización de este microprocesador despertó el interés por buscar nuevas técnicas, desarrollar algoritmos que mejoren el rendimiento, permitiendo aumentar la capacidad de procesamiento con un menor consumo de energía, dirigiendo mi interés a buscar la implementación de algoritmos en hardware y posteriormente en software.

1.2. Diseño de un microprocesador Superescalar.

El objetivo del proyecto final fue la implementación y entrega de un microprocesador superescalar en un FPGA, basándose en el set de instrucciones del MIPS R4000, cubriendo en su totalidad el diseño y arquitectura de las diferentes etapas y unidades que lo componen.

1.2.1 Introducción

La idea fundamental fue la de aplicar técnicas avanzadas para mejorar el rendimiento del microprocesador mediante el uso de la segmentación formada por las mismas 5 etapas, incorporando un pipeline de 5 capas conocida como superescalar.

1.2.2 Antecedentes

La arquitectura Superescalar con *pipeline lineal* se distingue con respecto de la Escalar, Uní-clico, y Multi-ciclo con *pipeline* no lineal, en las siguientes características: el ciclo de maquina toma un ciclo de reloj, las instrucciones son despachadas cada ciclo de reloj (idealmente), utilizan lógica de control y de transferencia de registro más elaborada, lo que se refleja en un mayor rendimiento, aumentando las instrucciones ejecutadas por ciclo de reloj, lo que se conoce como *throughput computing*. Esta técnica ha sido utilizada en procesadores de Pentium 4 el cual tiene un pipeline de 20 y en posteriores versiones con hasta 41 etapas.

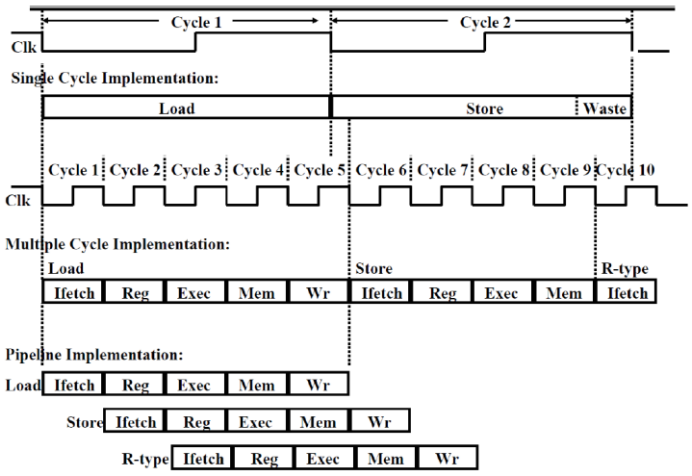


Figura 2-1 Comparación de las arquitecturas contra el Pipeline

1.2.3 Solución Desarrollada

Contempló la implementación del microprocesador superescalar MIPS con la arquitectura ligada al *ISA* (*Instruction Set Architecture*) R4000, y el diseño de las siguientes etapas del *pipeline*:
I-Fetch: I-Cache e IFQ: comprende las unidades de pre-carga para instrucciones, datos de *cache* y colas.
Dispatch: comprende la unidad de *dispatch*, el RST, un TAG FIFO y el banco de registros.
Issue Queues: la misma para las unidades de ejecución de enteros, divisiones y multiplicaciones.
Load/Store Queue. Unidades de Ejecución: enteros, D-*Cache*, divisor y multiplicador
Issue Unit: comprende la unidad de *issue* y el manejo del CDB.

En el apéndice B 1.1 ¨Arquitectura propuesta¨, se encuentra el diagrama y la descripción completa de cada una de las etapas; el listado de unidades se encuentra descrito en el Apéndice B 1.7 ¨Listado de unidades Implementadas del proyecto¨, así mismo, se proporcionan los archivos con extensión ¨.v¨ que contiene la descripción de la lógica utilizada y que se sintetizó en el FPGA para la entrega final de proyecto; en el Apéndice B 1.6 se listan los módulos implementados en RTL.

1.2.4 Análisis de resultados

Una vez implementado en el Kit del FPGA Virtex6, se procedió con la precarga del código prueba el cual era el ordenamiento de datos utilizando el algoritmo de burbuja (Apéndice B 1.5 Programas de prueba). La idea fue comprobar que el microprocesador implementado pudiera correr y ejecutar las instrucciones con dependencia de datos y medir el tiempo que tardaba en realizar el programa completo y con ello validar el funcionamiento y el rendimiento apropiado del mismo, además de comparar los resultados obtenidos contra los esperados. La implementación que se hizo se probó de dos maneras:

Primero, se utilizó el *pipeline* lineal de 5 etapas sin activar el reordenamiento de registros con lo cual se obtuvieron 7.6 segundos en la ejecución completa del programa entregando el resultado esperado, con una frecuencia de reloj en el FPGA de 500MHZ.

Segundo, se activó el reordenamiento de registros vía la habilitación por hardware de un bit externo, obteniendo mejoras significativas en el rendimiento. El tiempo en que tardó en ejecutar el mismo programa fue de 5.2 segundos y con ello se logró una mejora del 42%, cabe mencionar que estos resultados solo aplican para este procesador experimental, ya que otras implementaciones obtuvieron resultados similares o mejores.

1.2.5 Conclusiones

Para la entrega final de este trabajo se dedicaron muchas horas de diseño e implementación, proponiendo mejoras en la etapa de optimización y desempeño; se desarrolló un compilador y traductor de pseudocódigo ensamblador a código máquina, para la precarga del programa a ejecutar en el microprocesador, el cual utilizó un algoritmo de ordenamiento de burbuja para datos, mostrando resultados óptimos durante la ejecución del mismo. El trabajo se dividió como se especifica en el Apéndice B.

1.3. Propuesta para determinar el Cálculo del tamaño ideal de bloque de caché

El objetivo principal de este proyecto final fue el de demostrar el desarrollo de un método dinámico para determinar el tamaño ideal en el bloque de memoria *cache* utilizado en un microprocesador experimental, además se utilizó la medición del tamaño de los datos leídos de la memoria *cache* de datos, y la frecuencia que una dirección de memoria es utilizada a lo largo de la ejecución de cualquier aplicación para con ello determinar de manera dinámica el tamaño ideal durante la transferencia de información de forma general.

1.3.1 Introducción

El tamaño de bloque de caché típicamente usado es de 64 bytes, por estándar, sin embargo, en algunas aplicaciones ese tamaño no es el óptimo y se necesita estudiar el comportamiento de las aplicaciones para determinar cuál es la tendencia del rendimiento y cuáles serían las ventajas

de encontrar el tamaño exacto de la línea de *cache* dinámicamente para optimizar la cantidad de transferencias de datos de memoria principal a las memorias *caches*.

1.3.1 Antecedentes

El método caracteriza el comportamiento que tienen las operaciones de lectura de datos en una aplicación a lo largo de su ejecución en el microprocesador y consideró:

1. Los accesos a memoria de Datos *cache* a través de lecturas de 64bits, 32 bits, 16bits, 8 bits 0bits.

2. El tamaño del bloque de memoria principal es igual al tamaño del bloque de memoria *cache*, esto es: una localidad de memoria *cache* incluye una y solo una localidad de memoria principal.

3. Los accesos a memoria *cache* de datos se modelaron bajo el análisis estadístico de distribución gaussiana, con lo cual se determina el patrón comportamental de las muestras, y con ello encontrar el valor óptimo para el tamaño de la *cache* para así optimizar de manera dinámica su tamaño.

4. El algoritmo hará el muestreo del número de veces que ocurre la ejecución de la micro instrucción del tipo *load* para acceder a los datos a memoria *cache;* y diferenciar el número de eventos en los que ocurrieron los accesos a la misma dirección de memoria *cache* contabilizando cada uno de los diferentes tamaños de longitud posible por dirección de memoria leída.

1.3.2 Solución Desarrollada

La solución comprendió el desarrollo de un algoritmo que permite su inclusión en la etapa de *issue*, de decodificación y de *writeback*; para la captura de las micro-operaciones que se utilizan para acceder a los datos que se encuentran alojados en memoria *cache* y que son requeridos por las macro operaciones para la realización de sus operaciones.

Etapa *Issue*, desarrollada por Enrique Israel Hernández.

Para la etapa de *Issue* se hicieron incorporaciones de código para instrumentar el comportamiento de las micro operaciones de *load* y *effective address*, para determinar la dirección física que se necesita para traer el dato de la memoria *cache*, y determinar la frecuencia con la que una dirección de memoria de *cache* de datos es utilizada, en la etapa de *writeback*.
Etapa de Decodificación, ésta etapa fue implementada por Pablo Márquez.

En esta etapa se realizó la modificación del archivo *uop.c* para incorporar el código necesario que permita la identificación de la micro operación de lectura a *cache* (load) que será utilizada en etapas posteriores. También se incluyó en la estructura "*uop_t*" (estructura de datos de las micro operaciones) un nuevo elemento que nos proporciona el tamaño de la memoria leída.
Etapa de *Writeback*, Esta etapa se desarrolló en conjunto

En esta etapa se instrumentó el algoritmo que captura y procesa los datos. Aquí se realizó la selección de las micro-operaciones que tienen acceso a memoria *cache* de datos, el procesamiento de la dirección de donde se obtuvo el dato, el almacenamiento de la longitud del dato leído, el agrupamiento en eventos de la ocurrencia del tamaño de esos datos, la contabilización y la frecuencia con la que fueron utilizados.

1.3.3 Análisis de resultados

Con base en los resultados observados en el Apéndice C la mayor ocurrencia de tamaño de lecturas de *cache* se dan en tamaños de 4 bytes, siendo obtenidos mayormente en una sola transferencia, este comportamiento es relevante, ya que muestra la diferencia entre los *IPC* usando líneas de *cache* de 64 Bytes y de 32 Bytes. La propuesta para el mejor tamaño de línea de *cache* se basó en que el 95% de los accesos a memoria *cache* implican, la utilización de un tamaño de datos de 4 a 8 bytes e involucra una reducción del tamaño de las líneas de hasta un mínimo de 32 a 16 bytes, este tamaño de línea estaría sujeto a ser corroborado mediante un mayor número de corridas de simulación, la cual se dejará para trabajos posteriores en procesadores comerciales.

1.3.4 Conclusiones

Los resultados que se obtuvieron en esta investigación impulsaron el interés por aplicar el conocimiento adquirido en la implementación del algoritmo en hardware y hacer las

modificaciones pertinentes al código del microprocesador *MIPS Superscalar* implementado en la clase de diseño de microprocesadores y llevar a la práctica dicha implementación dinámica.

1.4. Conclusión General

Durante la realización de los proyectos anteriormente mencionados, se generó un interés genuino por desarrollar técnicas con base en el análisis científico, el comportamiento, y la búsqueda de patrones que aporten soluciones eficaces que optimicen la funcionalidad, mejoren el rendimiento, y aumenten la capacidad de procesamiento con un menor consumo de energía en cualquier circuito. Dirigiendo así mi interés por diseñar nuevos algoritmos que permitan su implementación en software y posteriormente en hardware.

El resultado obtenido de esta experiencia significó un notable avance en mi carrera ya que me permitió utilizar este conocimiento en proyectos de diseño de procesadores durante mi periodo en Intel, involucrándome en el desarrollo de varias versiones de micro procesadores multi-núcleo (*Xeon* y *Xeon-Phi*), ya que me permitió fundamentar la arquitectura y el desarrollo de protocolos de coherencia de *cache MESI, y MESIF*, con base en el análisis del comportamiento en simulaciones de interconexión en protocolos de transferencia de información en el *MESH* de datos e instrucciones, en donde cada cruce de nodos significó la incorporación de un núcleo con un procesador *Atom* de arquitectura *Silvermonth;* teniendo acceso a segmentos de datos de otros núcleos y trayendo de manera dinámica ráfagas de datos e instrucciones usados en *IDI* y *QPI, a*sí como en el desarrollo e implementación de algoritmos de cómputo visual en *hardware* como la transformada de HOUGH, VIOLA JONES, *Canny Edge Detection,* para el procesamiento de imágenes en 2D en *video analytics processors.*

En Continental Automotive Group me permitió alcanzar la posición como Líder de Grupo de desarrollo de Algoritmos para *Advanced Driving Assistance Systems* dirigiendo al equipo en la implementación de algoritmos de detección de transeúntes, objetos estáticos y dinámicos en radares de corto y de largo alcance, además de implementar funciones como Automatic Cruise Control, Blind Spot Detection, Rear Cross Traffic Alert, Bycicle Detection, Traffic Sign Recognition entre otros.

Hoy en día este trabajo sigue influyendo en mi carrera como Gerente de desarrollo de funciones *ADAS* para vehículos autónomos eléctricos, en la compañía *Future Mobility Corporation* en San José California cuya marca comercial de vehículos es BYTON.

La implementación de algoritmos en software y hardware es fundamental para el desarrollo de estas tecnologías y el conocimiento y las técnicas adquiridas en la Maestría en Diseño Electrónico han sido fundamentales en mi carrera.

Apéndices

A. DISEÑO DE MICROPROCESAR MIPS 32 BITS BASICO MULTICICLO

ITESO

Universidad Jesuita
de Guadalajara

Proyecto Final Parte 1
Diseño de microprocesador MIPS 32 BITS BASICO MULTICICLO

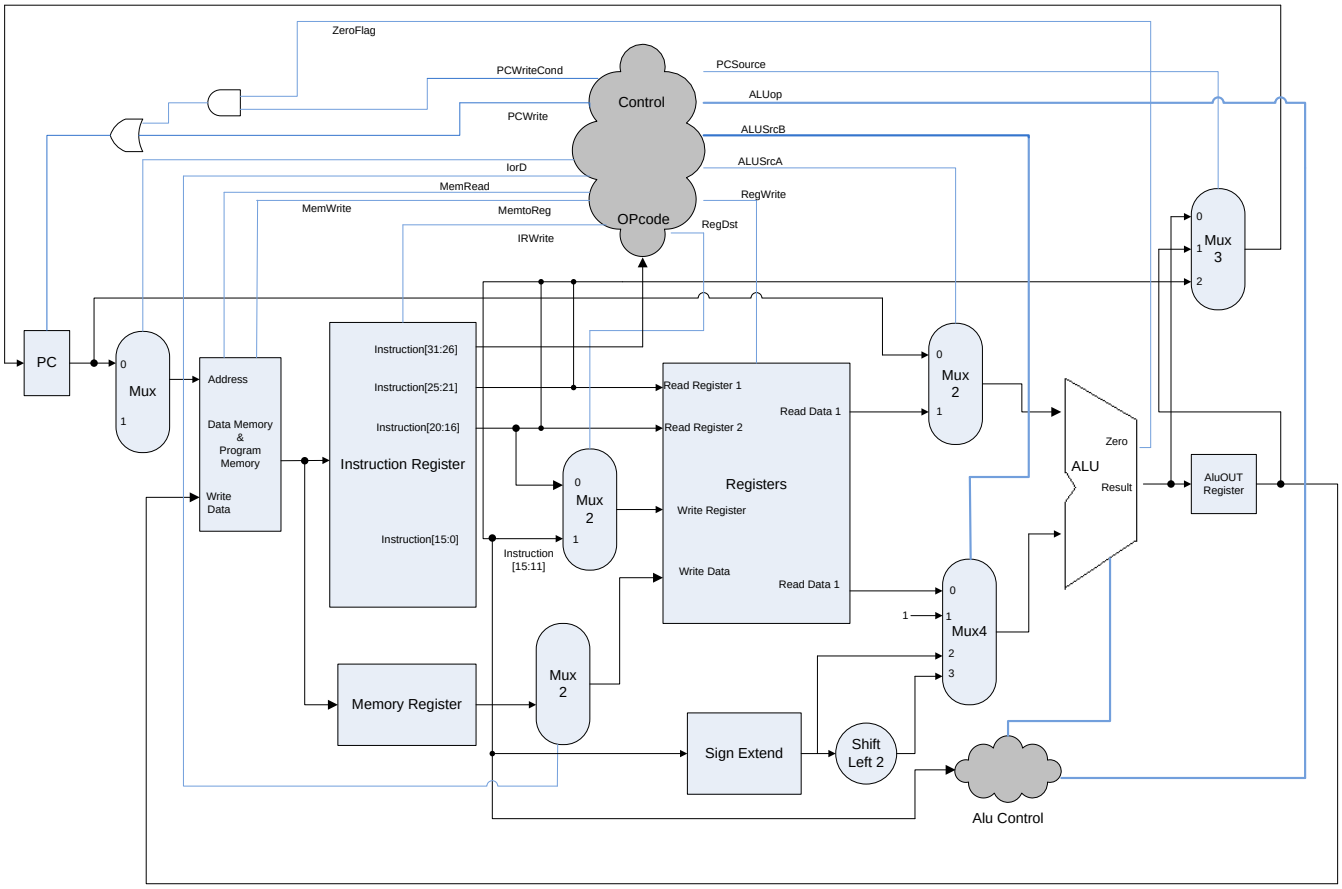
Alumno:
ENRIQUE ISRAEL HERNANDEZ MEZA

Diseño de microprocesador MIPS32 básico multi-ciclo

1) Diseñe un procesador MIPS32 básico, basado en la arquitectura multi-ciclos presentada en clase. El procesador debe ejecutar las siguientes instrucciones:
ADD RD, RS, RT
RD = RS + RT
ADDI RD, RS, CONST16
RD = RS + CONST16±
SUB RD, RS, RT
RD = RS – RT
SRL RD, RS, SHIFT5
RD = RS >> SHIFT5
SLL RD, RS, SHIFT5
RD = RS << SHIFT5
AND RD, RS, RT
RD = RS & RT
XOR RD, RS, RT
RD = RS Å RT
LW RD, OFF16(RS)
RD = MEM32(RS + OFF16±)
SW RS, OFF16(RT)
MEM32(RT + OFF16±) = RS
BEQ RS, RT, OFF18
IF RS = RT, PC += OFF18±
J ADDR28
PC = PC[31:28] :: ADDR28

Modifique la ruta de datos para que los registros 30 y 31 sean los puertos de entrada (sólo lectura) y salida (sólo escritura), respectivamente. El registro 0, es de sólo lectura y siempre lee un 0.
Inicialice la memoria de código del procesador con el programa necesario para ejecutar cada una de las instrucciones requeridas.
El reporte debe incluir:
a) Diagrama a bloques de la arquitectura implementada, realizado por el alumno.
b) Código RTL del procesador.
c) Formas de onda de la simulación mostrando las etapas de ejecución de las instrucciones del procesador.
d) Resumen del proceso de síntesis en el FPGA, mostrando la frecuencia máxima de operación y los recursos utilizados. Cerciorarse que no haya reporte de latches o bucles combinacionales.
- El reporte deberá ser convertido a un archivo PDF y enviado usando la página del curso en Moodle, no se aceptarán envíos por correo electrónico.
- El retraso en la entrega tendrá una penalización de un punto por cada día transcurrido.
- Cualquier indicio de plagio entre diseños, será penalizado con la baja académica para los ambos alumnos.

Reporte de resultados
Diagrama a Bloques de la Arquitectura



Código RTL del Procesador

Memory Unit

```
`timescale 1ns/100ps
//////////////////////////
// Titulo: Procesador MIPS Multiciclos
// Alumno: Enrique I Hernández Meza
// Maestro: Mariano Aguirre
// Interface: Memory Unit
// module file: memory_unit.v
// version:1.0
//
//////////////////////////
module memory_unit(Address, WriteData, MemRead, MemWrite, clk, rst, MEMData);
parameter word_size = 32;
parameter mem_size  = 64;
input [word_size-1:0]Address;
input [word_size-1:0]WriteData;
input MemWrite;
input MemRead;
input clk;
input rst;
output [word_size-1:0]MEMData;

reg [word_size-1:0]memory_reg[0:mem_size-1];
reg [word_size-1:0]MEMData;

always@(posedge clk or negedge rst)begin

    if(!rst)begin
            /// MEMORIA DE PROGRAMA
       memory_reg[32'h00] <= 32'h8c01_0010; // LW r1 <= mem[10'h] = 5'h;
       memory_reg[32'h01] <= 32'h8c02_0011; // LW r2 <= mem[11'h] = 7'h;
       memory_reg[32'h02] <= 32'h0022_1820; // ADD r1+r2->r3, r3= C'h
       memory_reg[32'h03] <= 32'h1041_0002; // BEQ= (r2-r1) pc+=02; r1 = 7'h
       memory_reg[32'h04] <= 32'h2021_0001; // ADDi r1 = r1 + 1'h;
       memory_reg[32'h05] <= 32'h0800_0003; // JUMP addr4'h;
       memory_reg[32'h06] <= 32'hAC03_0012; // SW r03 off12(r0), MEM[r0+off]= C'h
       memory_reg[32'h07] <= 32'h0061_2024; // AND r4 = r3 & r1 , r4 = 4'h;
       memory_reg[32'h08] <= 32'h0082_2826; // XOR r5 = r4 ^ r2 , r5 = 3'h
       memory_reg[32'h09] <= 32'h0065_3022; // SUB r6 = r3 - r5 , r6 = 9'h
       memory_reg[32'h0a] <= 32'h20C7_4007; // ADDi r7 = r6 + 10'h; r7 = 4010'h
       memory_reg[32'h0b] <= 32'h80e0_4142; // SRL r8 = r7 >> shift5 = 200'h
       memory_reg[32'h0c] <= 32'h80A0_4940; // SLL r9 = r5 >> shift5
       memory_reg[32'h0d] <= 32'h0000_0000; //
       memory_reg[32'h0e] <= 32'h0000_0000; //
       memory_reg[32'h0f] <= 32'h0000_0000; //
            // MEMORIA DE DATOS
       memory_reg[32'h10] <= 32'h0000_0005;  // DATO FUENTE DEL REGISTRO R1
       memory_reg[32'h11] <= 32'h0000_0007;  // DATO FUENTE DEL REGISTRO R2
       memory_reg[32'h12] <= 32'h0000_0000;  // DATO DESTINO DEL REGISTRO R3
       memory_reg[32'h13] <= 32'h0000_0000;  //
       memory_reg[32'h14] <= 32'h0000_0000;
       memory_reg[32'h15] <= 32'h0000_0000;
       memory_reg[32'h16] <= 32'h0000_0000;
    end
   else
   if (MemWrite)
      memory_reg[Address] <= WriteData;
   else
   if (MemRead)
      MEMData <= memory_reg[Address];
  end
//assign MEMData = memory_reg[Address];

Endmodule
```

Register Unit

```
`timescale 1ns/100ps
//////////////////////////
// Titulo: Procesador MIPS Multiciclos
// Alumno: Enrique I Hernandez Meza
// Maestro: Mariano Aguirre
// Interface:Register Unit
// module file: Register_unit.v
// version:1.0
//
//////////////////////////
module Register_unit(Read_register_1, Read_register_2, Write_register, RegWrite, Write_data, clk,
                     rst, Read_data_1, Read_data_2);
    parameter word_size = 32;
    input [4:0]Read_register_1;
    input [4:0]Read_register_2;
    input [4:0]Write_register;
    input RegWrite;
    input [31:0]Write_data;
    input clk;
    input rst;
    output [word_size-1:0]Read_data_1;
    output [word_size-1:0]Read_data_2;
    reg [word_size-1:0]registers[0:word_size-1];
    wire [word_size-1:0]reg_A;
    wire [word_size-1:0]reg_B;
    reg [word_size-1:0]Read_data_1;
    reg [word_size-1:0]Read_data_2;
     always@(posedge clk or negedge rst)begin
      if(!rst)
           registers [0]<= 32'h0;


      else     // avoiding to wirte reg 0 and reg 30 those register are only read type
      if(RegWrite && ((Write_register != 0)||( Write_register != 30)))
          registers[Write_register] <= Write_data;
    end
  always@(posedge clk)begin
           Read_data_1 <=  reg_A;
           Read_data_2 <=  reg_B;
    end

    assign reg_A = (Read_register_1 != 31)?registers[Read_register_1]:'hx;
    assign reg_B = (Read_register_1 != 31)?registers[Read_register_2]:'hx;
endmodule
```

Instruction register Unit

```
`timescale 1ns/100ps
//////////////////////////
// Titulo: Procesador MIPS Multiciclos
// Alumno: Enrique I Hernandez Meza
// Maestro: Mariano Aguirre
// Interface:Instruction register Unit
// module file: Instruction_register_unit.v
// version:1.0
//
//////////////////////////
module Instruction_register_unit(Instruction_in, IRwrite, clk, rst, Instruction_op, Instruction_rs,
          Instruction_rt, Instruction_rd, Instruction_addr);
   parameter word_size = 32;
   input [word_size-1:0]Instruction_in;
   input IRwrite;
   input clk;
   input rst;

   output [5:0]Instruction_op;
   output [4:0]Instruction_rs;
   output [4:0]Instruction_rt;
   output [4:0]Instruction_rd;
   output [15:0]Instruction_addr;
   reg [5:0]Instruction_op;
   reg [4:0]Instruction_rt;
   reg [4:0]Instruction_rs;
   reg [4:0]Instruction_rd;
   reg [15:0]Instruction_addr;

   always@(posedge clk, negedge rst)begin
      if (!rst)begin
         Instruction_op   <= 0;
         Instruction_rs   <= 0;
         Instruction_rt   <= 0;
         Instruction_addr <= 0;end
      else
      if(IRwrite)begin
         Instruction_op   <= Instruction_in[31:26];
         Instruction_rs   <= Instruction_in[25:21];
         Instruction_rt   <= Instruction_in[20:16];
         Instruction_rd   <= Instruction_in[15:11];
         Instruction_addr <= Instruction_in[15:0];end
    end
endmodule
```

ALU Unit

```
`timescale 1ns/100ps
//////////////////////////
// Titulo: Procesador MIPS Multiciclos
// Alumno: Enrique I Hernandez Meza
// Maestro: Mariano Aguirre
// Interface:ALU Unit
// module file: ALU_unit.v
// version:1.0
//
//////////////////////////
module ALU_unit(Data_in_A, Data_in_B, ALUop, Inst_func_field, clk, rst, ALU_result, Zero_flag, ALUOUT);
   parameter word_size = 32;
   input [word_size-1:0]Data_in_A; // Datos de Entrada del segundo operando
   input [word_size-1:0]Data_in_B; // Datos de Entrada del primer operando
   input [5:0]Inst_func_field; // Instrucion a ejecutarse en la ALU
   input [1:0]ALUop; // Habilitador de la ALU Para que realice una Operacion
   input clk;
   input rst;
   output [word_size-1:0]ALU_result; // Salida de la ALU al Jump address
   output Zero_flag; // Este senal es la bandera de cero?
```

```
    output [word_size-1:0]ALUOUT; // Salida de la ALU Registrada

    reg [3:0]Alu_ctrl;
    reg [word_size-1:0]ALUOUT;
    reg [word_size-1:0]ALU_result;
   parameter Addi = 2'b00, Subi = 2'b01, R_type = 2'b10;
    parameter ADD = 6'b100000, SUB = 6'b100010, AND = 6'b100100, OR = 6'b100101,
              SLT = 6'b101010, XOR = 6'b100110, NOR = 6'b100111, NOP = 6'b101100,
              SRL = 6'b000010, SLL = 6'b000000;

    always@(*)begin
      case(ALUop)
       Addi: ALU_result  = (Data_in_A + Data_in_B);
       Subi: ALU_result  = (Data_in_A - Data_in_B);
       R_type:
             case(Inst_func_field)
               ADD: ALU_result = (Data_in_A + Data_in_B);

               SUB: ALU_result = (Data_in_A - Data_in_B);
               AND: ALU_result = (Data_in_A & Data_in_B);
               OR:  ALU_result = (Data_in_A | Data_in_B);
               XOR: ALU_result = (Data_in_A ^ Data_in_B);
               NOR: ALU_result = ~(Data_in_A | Data_in_B);
               SLT: ALU_result = Data_in_A + Data_in_B; // investigar que hace esta operacion y como se comporta
               SRL: ALU_result = Data_in_A >> Data_in_B[10:6];
               SLL: ALU_result = Data_in_A << Data_in_B[10:6];
               NOP: ALU_result = 0;
               default: ALU_result = 'bx;
             endcase
      endcase
    end
   always@(posedge clk)
   ALUOUT <=ALU_result;
   assign  Zero_flag = ~|ALU_result;
 endmodule
```

Multiplexer 2 to 1 Unit

```
`timescale 1ns/100ps
//////////////////////////
// Titulo: Procesador MIPS Multiciclos
// Alumno: Enrique I Hernandez Meza
// Maestro: Mariano Aguirre
// Interface:Multiplexer 2 to 1 Unit
// module file: Mux2_unit.v
// version:1.0
//
//////////////////////////
module Mux2_unit(In_mux_0, In_mux_1, En_mux, Out_mux);

    input [31:0]In_mux_0;
    input [31:0]In_mux_1;
    input En_mux;
    output [31:0]Out_mux;
    assign Out_mux = (En_mux)?In_mux_1:In_mux_0;
endmodule
```

Multiplexer 3 to 1 Unit

```
`timescale 1ns/100ps
//////////////////////////
// Titulo: Procesador MIPS Multiciclos
// Alumno: Enrique I Hernandez Meza
// Maestro: Mariano Aguirre
// Interface:Multiplexer 3 to 1 Unit
// module file: Mux3_unit.v
// version:1.0
//
//////////////////////////
module Mux3_unit(In_mux_0, In_mux_1, In_mux_2, En_mux, Out_mux);
    input [31:0]In_mux_0;
    input [31:0]In_mux_1;
```

```
    input [31:0]In_mux_2;
    input [1:0]En_mux;
    output [31:0]Out_mux;
    reg [31:0]Out_mux;
    always@(*)begin
      case(En_mux)
        2'b00: Out_mux = In_mux_0;
        2'b01: Out_mux = In_mux_1;
        2'b10: Out_mux = In_mux_2;
        default: Out_mux = In_mux_0;
      endcase
    end
endmodule
```

Multiplexer 4 to 1 Unit

```
`timescale 1ns/100ps
//////////////////////////
// Titulo: Procesador MIPS Multiciclos
// Alumno: Enrique I Hernandez Meza
// Maestro: Mariano Aguirre
// Interface:Multiplexer 4 to 1 Unit
// module file: Mux4_unit.v
// version:1.0
//
//////////////////////////
module Mux4_unit(In_mux_0, In_mux_1, In_mux_2, In_mux_3, En_mux, Out_mux);
    input [31:0]In_mux_0;
    input [31:0]In_mux_1;
    input [31:0]In_mux_2;
    input [31:0]In_mux_3;
    input [1:0]En_mux;
    output [31:0]Out_mux;
    reg [31:0]Out_mux;
    always@(*)begin
      case(En_mux)
        2'b00: Out_mux = In_mux_0;
        2'b01: Out_mux = In_mux_1;
        2'b10: Out_mux = In_mux_2;
        2'b11: Out_mux = In_mux_3;
        default: Out_mux = In_mux_0;
      endcase
    end
endmodule
```

Shift2rl 28 Unit

```
`timescale 1ns/100ps
//////////////////////////
// Titulo: Procesador MIPS Multiciclos
// Alumno: Enrique I Hernandez Meza
// Maestro: Mariano Aguirre
// Interface: Shift2rl 28 Unit
// module file: Shift2rl_unit.v
// version:1.0
//
//////////////////////////

module Shift2rl_unit(In_shift26, Out_shift28);
   input  [31:0]In_shift26;
   output [31:0]Out_shift28;
   reg [31:0]Out_shift28;
  always@(In_shift26)begin
      Out_shift28 <={4'b0, In_shift26,2'b0}; //
  end

 endmodule
```

Shift left 32 Unit

```
`timescale 1ns/100ps
//////////////////////////
// Titulo: Procesador MIPS Multiciclos
```

```
// Alumno: Enrique I Hernandez Meza
// Maestro: Mariano Aguirre
// Interface: Shift left 32 Unit
// module file: Shift_left_32.v
// version:1.0
//
//////////////////////////

module Shift_left_32(In_shift, Out_shift);
   input  [31:0]In_shift;
   output [31:0]Out_shift;
   reg [31:0]Out_shift;
  always@(In_shift)begin
      //Out_shift <= {In_shift,2'b0};
      Out_shift <= In_shift;
  end
 endmodule
```

Sign Extend Unit

```
`timescale 1ns/100ps
//////////////////////////
// Titulo: Procesador MIPS Multiciclos
// Alumno: Enrique I Hernandez Meza
// Maestro: Mariano Aguirre
// Interface: Sign Extend Unit
// module file: sign_ext_unit.v
// version:1.0
//////////////////////////
module sign_ext_unit(data_in_16b, data_out_32b);
    input [15:0]data_in_16b;
    output [31:0]data_out_32b;
    reg [31:0]data_out_32b;
    always@(data_in_16b)begin
         if(data_in_16b[15])
          data_out_32b[31:0] =  {16'b1,data_in_16b};
        else
          data_out_32b[31:0] =  {16'b0,data_in_16b};
    end
endmodule
```

Program Counter

```
`timescale 1ns/100ps
//////////////////////////
// Titulo: Procesador MIPS Multiciclos
// Alumno: Enrique I Hernandez Meza
// Maestro: Mariano Aguirre
// Interface: Program Counter
// module file: P_counter.v
// version:1.0
//////////////////////////
module P_counter(In_pc, PCWrite, Zero_flag, PCWriteCond, clk, rst, pc_address);
    input [31:0]In_pc;
    input PCWrite;
    input Zero_flag;
    input PCWriteCond;
    input clk;
    input rst;
    output [31:0]pc_address;
    reg [31:0]count_pc;
    wire load_pc;
    assign load_pc = ( PCWrite || ( PCWriteCond&& Zero_flag ) );
    always@(posedge clk, negedge rst)begin
     if (!rst)
           count_pc <= 0;
     else
        if(load_pc)
           count_pc <= In_pc ;
     else
```

```
        count_pc <= count_pc;
    end

   assign pc_address = count_pc;
endmodule
```

Memory Data Register

```
`timescale 1ns/100ps
//////////////////////////
// Titulo: Procesador MIPS Multiciclos
// Alumno: Enrique I Hernandez Meza
// Maestro: Mariano Aguirre
// Interface: Memory Data Register
// module file: Memory_datareg.v
// version:1.0
//////////////////////////
module Memory_datareg(MemDataIn, clk, rst, MemDataOut);
   input [31:0]MemDataIn;
   input clk;
   input rst;
   output [31:0]MemDataOut;
   reg [31:0]MemDataOut;
   always@(MemDataIn)begin
        MemDataOut = MemDataIn;
   end
endmodule
```

MIPS 2010 DATAPATH

```
`timescale 1ns/100ps
//////////////////////////
// Titulo: Procesador MIPS Multiciclos
// Alumno: Enrique I Hernandez Meza
// Maestro: Mariano Aguirre
// Interface: MIPS 2010 DATAPATH
// module file: Mips2010.v
// version:1.0
//
//////////////////////////

module Datapath(clk, rst, PCSource, ALUop, ALUSrcB, ALUSrcA, RegWrite, RegDst, IRWrite, MemtoReg,
          MemWrite, MemRead, IorD, PCWrite, PCWriteCond, op_code);
input clk;
input rst;
input [1:0]PCSource;
input [1:0]ALUop;
input [1:0]ALUSrcB;
input ALUSrcA;
input RegWrite;
input RegDst;
input IRWrite;
input MemtoReg;
input MemWrite;
input MemRead;
input IorD;
input PCWrite;
input PCWriteCond;
 output [5:0]op_code;


 wire [31:0]P_counter_out;
wire [31:0]mux2_dp_1_out;
wire [31:0]mux2_dp_2_out;
wire [31:0]mux2_dp_3_out;   // salida de 5 bits
wire [31:0]mux2_dp_4_out;
wire [31:0]mux3_dp_out;
wire [31:0]mux4_dp_out;
```

```
wire [31:0]memory_dp_out;
wire [31:0]Memory_datareg_dp_out;
wire [5:0]iregister_dp_Instruction_op;
wire [4:0]iregister_dp_Instruction_rs;
wire [4:0]iregister_dp_Instruction_rt;
wire [4:0]iregister_dp_Instruction_rd;
wire [15:0]iregister_dp_Instruction_addr;
wire [31:0]register_dp_Read_data_1;
wire [31:0]register_dp_Read_data_2;
wire [31:0]alu_dp_ALU_result;
wire alu_dp_Zero_flag;
wire [31:0]alu_dp_ALUOUT;
wire [31:0]Shift28_dp_Out;
wire [31:0]Shift32_dp_Out;
wire [31:0]signextend_dp_Out;

assign op_code = iregister_dp_Instruction_op;

Mux3_unit mux3_dp(.In_mux_0(alu_dp_ALU_result), .In_mux_1(alu_dp_ALUOUT),
.In_mux_2({P_counter_out[31:28],Shift28_dp_Out[27:0]}), .En_mux(PCSource), .Out_mux(mux3_dp_out));

P_counter Pcounter_dp(.In_pc(mux3_dp_out), .PCWrite(PCWrite), .Zero_flag(alu_dp_Zero_flag), .PCWriteCond(PCWriteCond),
.clk(clk), .rst(rst), .pc_address(P_counter_out));

Mux2_unit mux2_dp_1(.In_mux_0(P_counter_out), .In_mux_1(alu_dp_ALUOUT), .En_mux(IorD), .Out_mux(mux2_dp_1_out));
Mux2_unit mux2_dp_2(.In_mux_0(alu_dp_ALUOUT), .In_mux_1(Memory_datareg_dp_out), .En_mux(MemtoReg),
.Out_mux(mux2_dp_2_out));

Mux2_unit mux2_dp_3(.In_mux_0({27'b0,iregister_dp_Instruction_rt}), .In_mux_1({27'b0,iregister_dp_Instruction_rd}),
.En_mux(RegDst), .Out_mux(mux2_dp_3_out));

Mux2_unit mux2_dp_4(.In_mux_0(P_counter_out), .In_mux_1(register_dp_Read_data_1), .En_mux(ALUSrcA),
.Out_mux(mux2_dp_4_out));

memory_unit memory_dp(.Address(mux2_dp_1_out), .WriteData(register_dp_Read_data_2), .MemRead(MemRead),
.MemWrite(MemWrite), .clk(clk), .rst(rst), .MEMData(memory_dp_out));

Memory_datareg Memory_datareg_dp(.MemDataIn(memory_dp_out), .clk(clk), .rst(rst),
.MemDataOut(Memory_datareg_dp_out));
Instruction_register_unit iregister_dp(.Instruction_in(memory_dp_out), .IRwrite(IRWrite), .clk(clk), .rst(rst),
.Instruction_op(iregister_dp_Instruction_op), .Instruction_rs(iregister_dp_Instruction_rs),
.Instruction_rt(iregister_dp_Instruction_rt), .Instruction_rd(iregister_dp_Instruction_rd),
.Instruction_addr(iregister_dp_Instruction_addr));

Register_unit register_dp(.Read_register_1(iregister_dp_Instruction_rs), .Read_register_2(iregister_dp_Instruction_rt),
.Write_register(mux2_dp_3_out[4:0]), .RegWrite(RegWrite),.Write_data(mux2_dp_2_out), .clk(clk), .rst(rst),
.Read_data_1(register_dp_Read_data_1), .Read_data_2(register_dp_Read_data_2));

Mux4_unit mux4_dp(.In_mux_0(register_dp_Read_data_2), .In_mux_1(32'h1), .In_mux_2(signextend_dp_Out),
.In_mux_3(Shift32_dp_Out), .En_mux(ALUSrcB), .Out_mux(mux4_dp_out));

ALU_unit alu_dp(.Data_in_A(mux2_dp_4_out), .Data_in_B(mux4_dp_out), .ALUop(ALUop),
.Inst_func_field(iregister_dp_Instruction_addr[5:0]), .clk(clk), . ALU_result(alu_dp_ALU_result), .Zero_flag(alu_dp_Zero_flag),
.ALUOUT(alu_dp_ALUOUT));

Shift2rl_unit Shift26_28_dp(.In_shift26({P_counter_out[31:26],iregister_dp_Instruction_rs, iregister_dp_Instruction_rt,
iregister_dp_Instruction_addr}), .Out_shift28(Shift28_dp_Out));

 Shift_left_32 Shift32_32(.In_shift(signextend_dp_Out), .Out_shift(Shift32_dp_Out));

sign_ext_unit signextend_dp(.data_in_16b(iregister_dp_Instruction_addr), .data_out_32b(signextend_dp_Out));
endmodule
```

CONTROL PATH UNIT

```
`timescale 1ns/100ps
//////////////////////////
// Titulo: Procesador MIPS Multiciclos
// Alumno: Enrique I Hernandez Meza
// Maestro: Mariano Aguirre
// Interface: Control Unit
// module file: Control_unit.v
// version:1.0
//
//////////////////////////

module Control_unit(op_code, clk, rst, PCSource, ALUop, ALUSrcB, ALUSrcA, RegWrite, RegDst, IRWrite, MemtoReg,
          MemWrite, MemRead, IorD, PCWrite, PCWriteCond);

input [5:0]op_code;
input clk;
input rst;

output [1:0]PCSource;
output [1:0]ALUop;
output [1:0]ALUSrcB;
output ALUSrcA;
output RegWrite;
output RegDst;
output IRWrite;
output MemtoReg;
output MemWrite;
output MemRead;
output IorD;
output PCWrite;
output PCWriteCond;

reg [1:0]PCSource;
reg [1:0]ALUop;
reg [1:0]ALUSrcB;
reg ALUSrcA;
reg RegWrite;
reg RegDst;
reg IRWrite;
reg MemtoReg;
reg MemWrite;
reg MemRead;
reg IorD;
reg PCWrite;
reg PCWriteCond;
reg [3:0]current_state;
reg [3:0]next_state;

        /// CPU STATES PARAMETERS
parameter Reset = 4'b1111, Fetch = 4'b0000, Decode =  4'b0001, Memory_comp = 4'b0010,
        Memory_access = 4'b0011, Memory_read = 4'b0100, Memory_write = 4'b0101,
        Execution = 4'b0110, Rtype = 4'b0111, Branch = 4'b1000, IType = 4'b1010,
        Jump = 4'b1001, IType_complete = 4'b1011,Complete = 4'b1100;

        ///OPCODE PARAMETERS
parameter Lw = 6'b10_0011 , Sw = 6'b10_1011, R_type = 6'b000000, B_EQ = 6'b000100,
        J_ump = 6'b000010, I_type=6'b001000, Shift = 6'b100000;

always@(posedge clk, negedge rst)begin
     if(!rst)
       current_state <= Reset;
     else
       current_state <= next_state;
end
always@(current_state, op_code)begin
   PCSource = 2'b00; ALUop = 2'b00; ALUSrcB = 2'b00; ALUSrcA = 1'b0; RegWrite = 1'b0;
   RegDst = 1'b0; IRWrite = 1'b0; MemtoReg = 1'b0; MemWrite = 1'b0; MemRead = 1'b0;
```

```
IorD = 1'b0; PCWrite = 1'b0; PCWriteCond  = 1'b0;

case(current_state)

Reset: begin PCSource = 2'b00; ALUop = 2'b00; ALUSrcB = 2'b00; ALUSrcA = 1'b0; RegWrite = 1'b0;
      RegDst = 1'b0; IRWrite = 1'b0; MemtoReg = 1'b0; MemWrite = 1'b0;
      IorD = 1'b0; PCWrite = 1'b0; PCWriteCond  = 1'b0; MemRead = 1'b1;
      next_state = Fetch; end

Fetch:  begin ALUSrcA = 1'b0; IorD = 1'b0;  PCWrite = 1'b1; IRWrite = 1'b1;
       ALUSrcB = 2'b01; ALUop = 2'b00; PCSource = 2'b00; MemRead = 1'b1;
       next_state = Decode;end

Decode: begin ALUSrcA = 1'b0; ALUSrcB = 2'b11; ALUop = 2'b00;
      case(op_code)
        Lw:  next_state = Memory_comp;
        Sw:  next_state = Memory_comp;
        Shift: next_state = Execution;
        R_type: next_state = Execution;
        I_type: next_state = IType;
        B_EQ:   next_state = Branch;
        J_ump:  next_state = Jump;
        default:next_state = current_state;
      endcase
     end
Memory_comp: begin ALUSrcA = 1'b1; ALUSrcB = 2'b10; ALUop = 2'b00;
      if(op_code == Lw) next_state = Memory_access;
      else next_state = Memory_write;
      end

Memory_access: begin MemRead = 1'b1; IorD = 1'b1;
      next_state = Memory_read; end

Memory_read: begin RegDst = 1'b0; RegWrite = 1'b1; MemtoReg = 1'b1;
      MemRead = 1'b1; next_state = Fetch;end

Memory_write: begin MemWrite = 1'b1; IorD = 1'b1;
      next_state = Complete;end

Execution: begin ALUSrcA = 1'b1;  ALUop = 2'b10;
       if(op_code == Shift)ALUSrcB = 2'b10;
       else ALUSrcB = 2'b00;
       next_state = Rtype;end

Rtype:  begin RegDst = 1'b1; RegWrite = 1'b1; MemtoReg = 1'b0;
       next_state = Complete;end // MemRead = 1'b1;

Branch: begin ALUSrcA = 1'b1; ALUSrcB = 2'b00; ALUop = 2'b01;
        PCWriteCond = 1'b1;  PCSource = 2'b01;  next_state = Complete;end //MemRead = 1'b1;

IType: begin ALUSrcA = 1'b1; ALUSrcB = 2'b10; RegDst = 1'b0; MemtoReg = 1'b0;
       ALUop = 2'b00; next_state = IType_complete;end

IType_complete:begin RegWrite = 1'b1; MemRead = 1'b1; next_state = Fetch;end
Jump:  begin PCWrite = 1'b1; PCSource = 2'b10;
      next_state = Complete;end
Complete: begin  MemRead = 1'b1; next_state = Fetch;end
default : next_state = current_state;
endcase
endendmodule
```

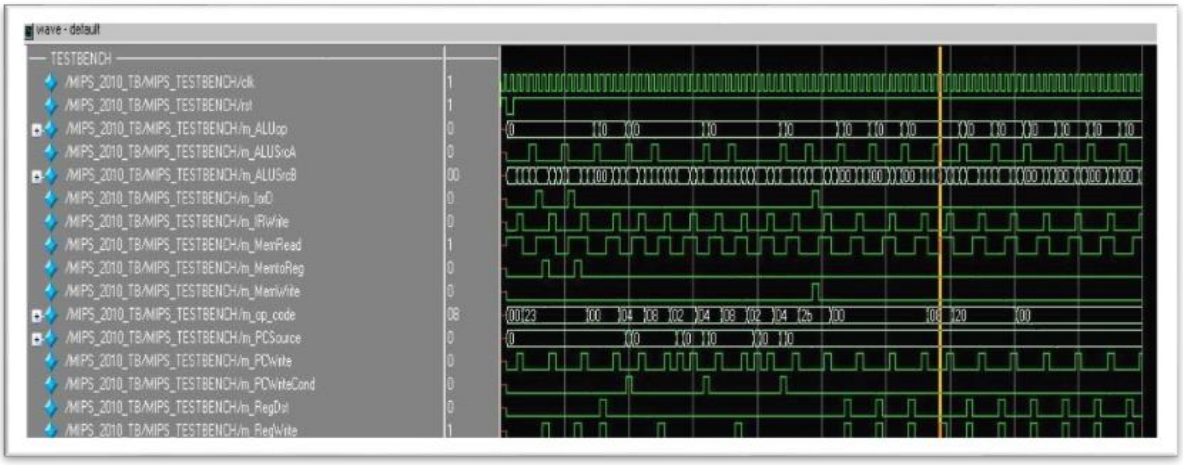
MIPS TOP

```
`timescale 1ns/100ps
//////////////////////////
// Titulo: Procesador MIPS Multiciclos
// Alumno: Enrique I Hernandez Meza
// Maestro: Mariano Aguirre
// Interface: Mips 2010 Top
// module file: Mips.v
// version:1.0
//////////////////////////

module Mips(clk, rst);
   input clk;
   input rst;

   wire [5:0]m_op_code;
   wire [1:0]m_PCSource;
   wire [1:0]m_ALUop;
   wire [1:0]m_ALUSrcB;
   wire m_ALUSrcA;
   wire m_RegWrite;
   wire m_RegDst;
   wire m_IRWrite;
   wire m_MemtoReg;
   wire m_MemWrite;
   wire m_MemRead;
   wire m_IorD;
   wire m_PCWrite;
   wire m_PCWriteCond;


   Datapath mips_Datapath(.clk(clk), .rst(rst), .PCSource(m_PCSource), .ALUop(m_ALUop), .ALUSrcB(m_ALUSrcB),
                  .ALUSrcA(m_ALUSrcA), .RegWrite(m_RegWrite), .RegDst(m_RegDst), .IRWrite(m_IRWrite),
                  .MemtoReg(m_MemtoReg), .MemWrite(m_MemWrite), .MemRead(m_MemRead), .IorD(m_IorD),
                  .PCWrite(m_PCWrite), .PCWriteCond(m_PCWriteCond), .op_code(m_op_code));


   Control_unit mips_Control(.op_code(m_op_code), .clk(clk), .rst(rst), .PCSource(m_PCSource), .ALUop(m_ALUop),
                    .ALUSrcB(m_ALUSrcB), .ALUSrcA(m_ALUSrcA), .RegWrite(m_RegWrite), .RegDst(m_RegDst),
                    .IRWrite(m_IRWrite), .MemtoReg(m_MemtoReg), .MemWrite(m_MemWrite), .MemRead(m_MemRead),
                    .IorD(m_IorD), .PCWrite(m_PCWrite), .PCWriteCond(m_PCWriteCond));


endmodule
```

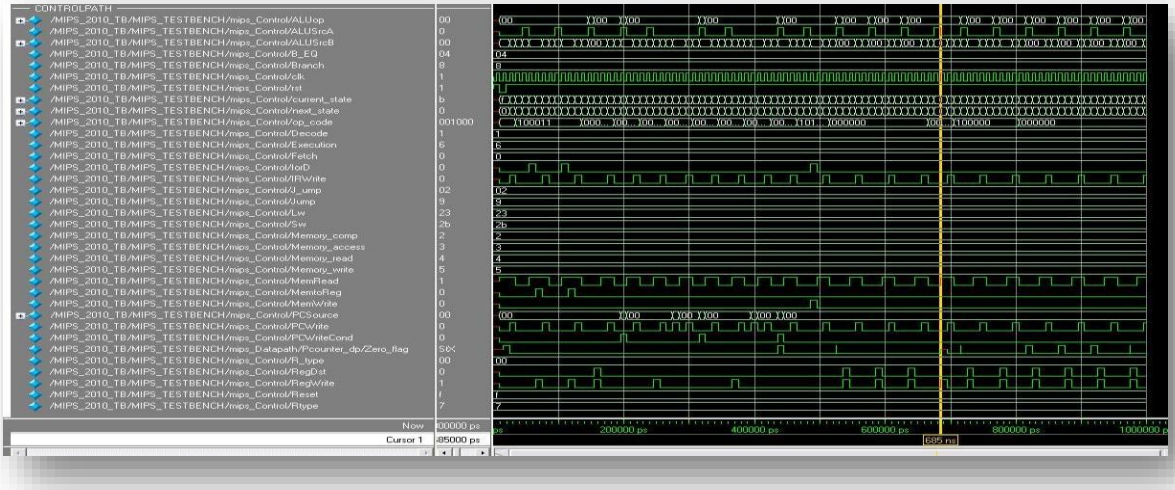
TEST BENCH CLOCK GENERATOR MIPS 2010

```
`timescale 1ns/100ps
//////////////////////////
// Titulo: Procesador MIPS Multiciclos
// Alumno: Enrique I Hernandez Meza
// Maestro: Mariano Aguirre
// Interface: TEST BENCH CLOCK GENERATOR MIPS 2010
// module file: MIPS.v
// version:1.0
//
//////////////////////////
module MIPS_2010_TB();
reg clock_tb;
reg reset_tb;
//  clock initialization
initial begin
    clock_tb = 0;
    forever clock_tb = #5 ~clock_tb;
end
// reset initialization
initial begin
   reset_tb = 1;
   # 10 reset_tb = 0;
   # 10 reset_tb = 1;
end// Instancia del micro
Mips   MIPS_TESTBENCH(.clk(clock_tb), .rst(reset_tb));
 Endmodule
```

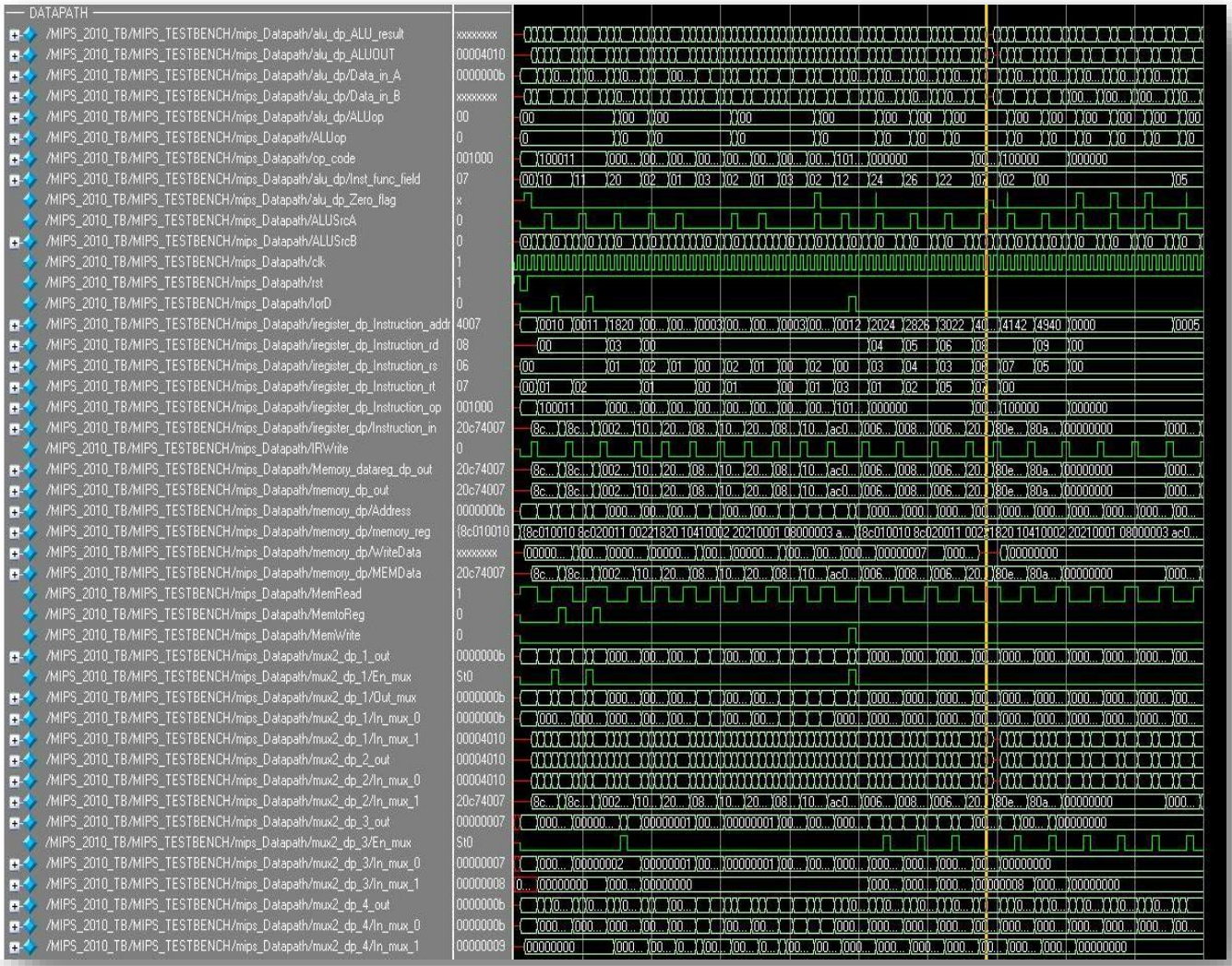
FORMAS DE ONDA OBTENIDAS
TESTBENCH VIEW



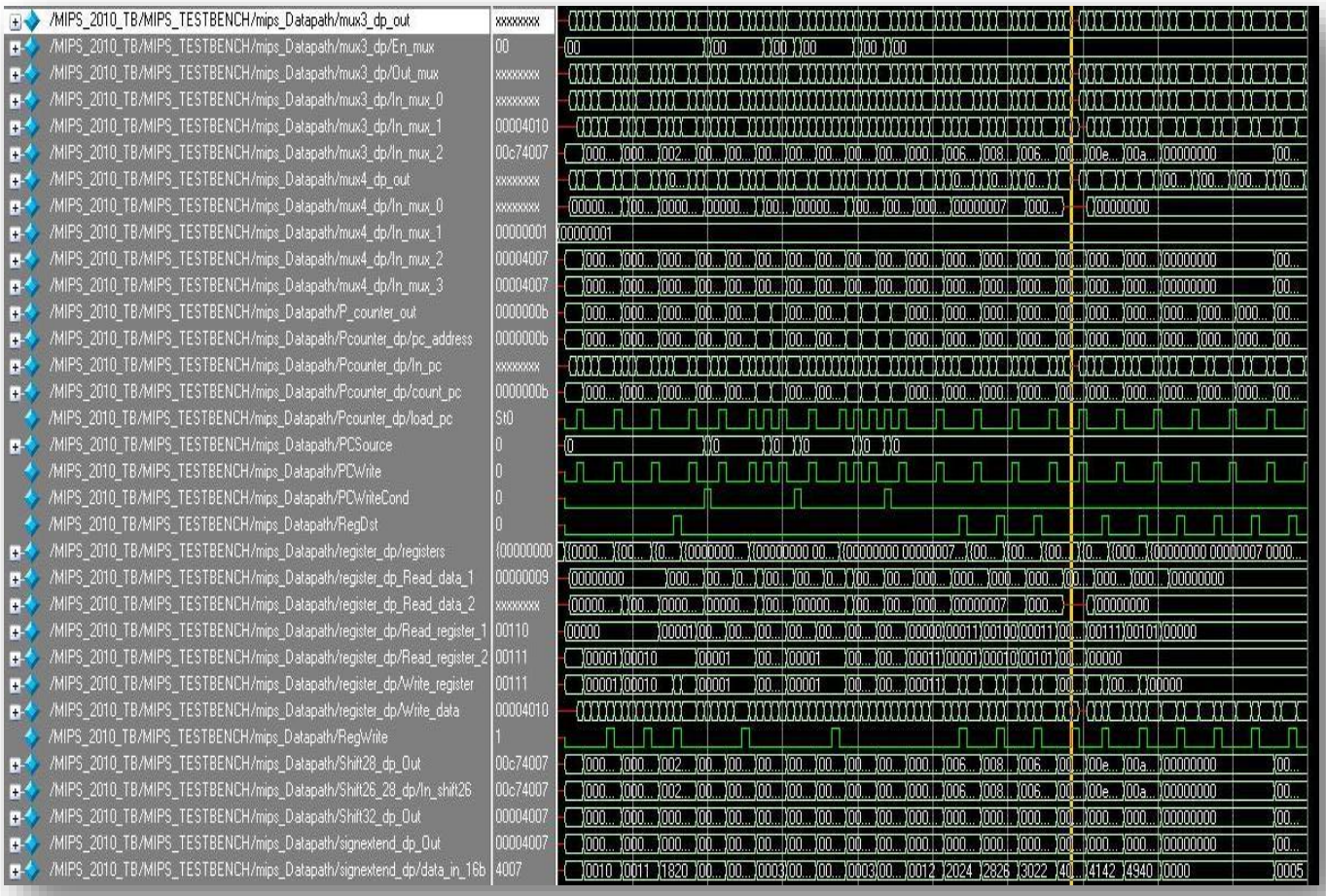
CONTROLPATH VIEW



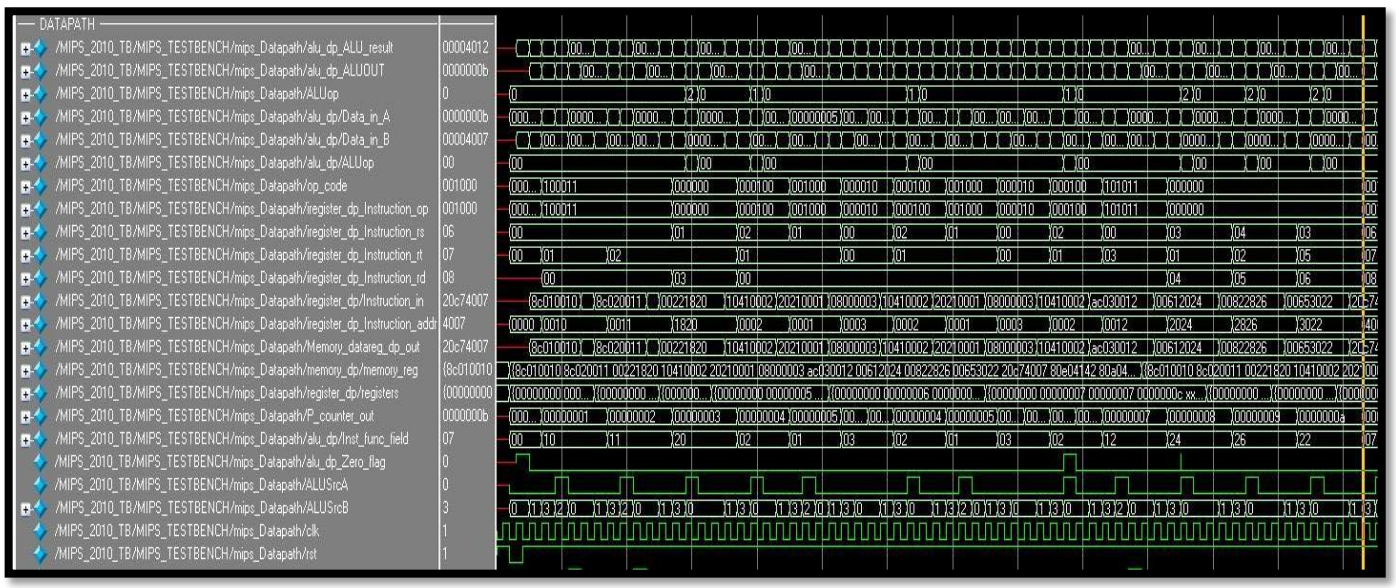
DATA PATH VIEW



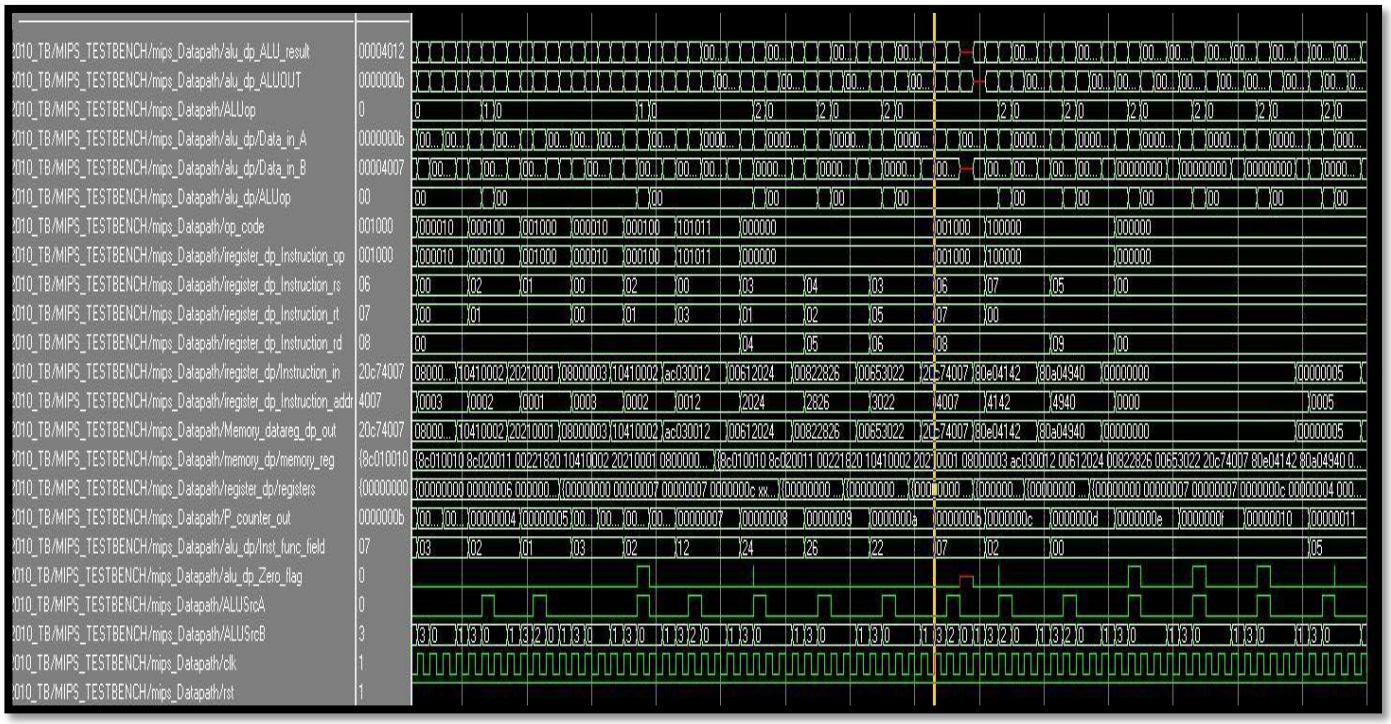
DATA PATH VIEW CONTINUE



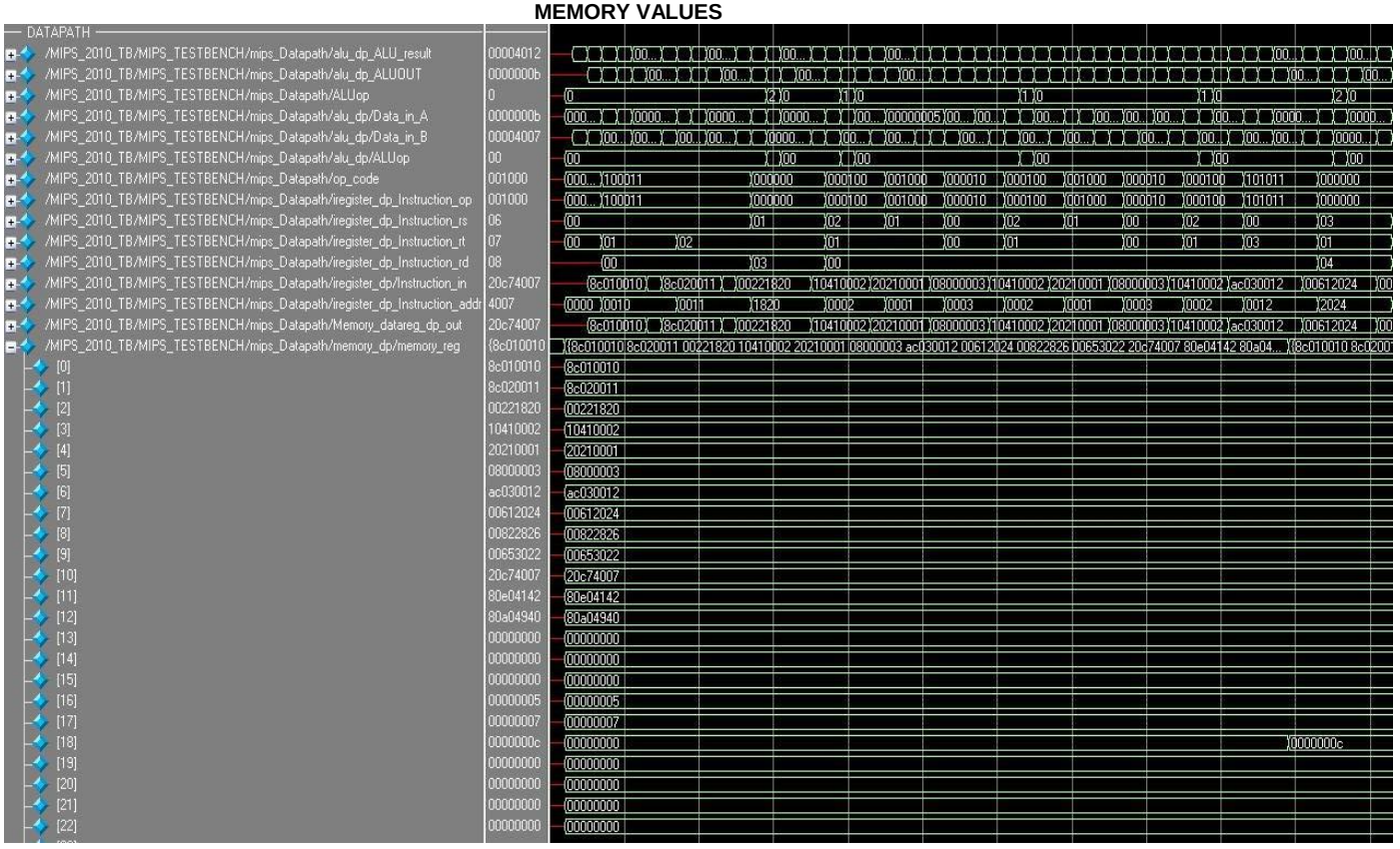
INSTRUCTIONS PERFORM VIEW



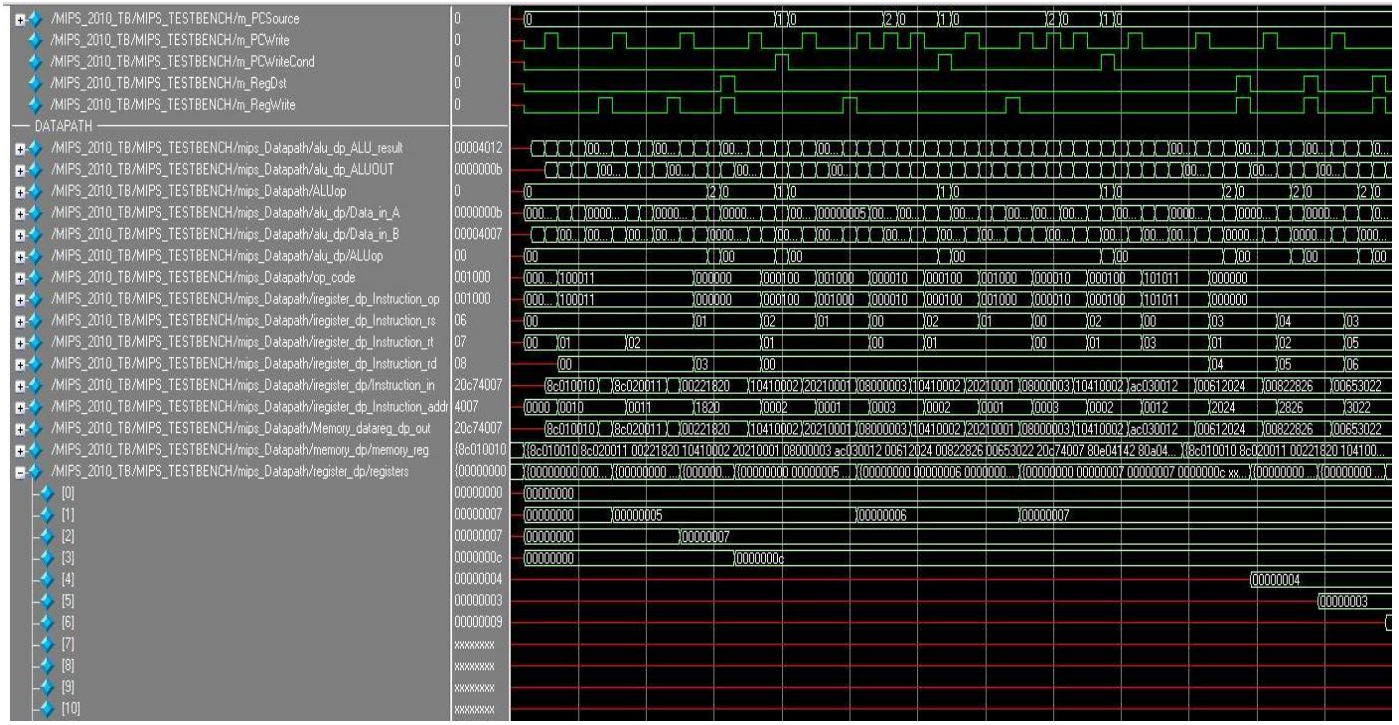
INSTRUCTION PERFORM CONTINUE



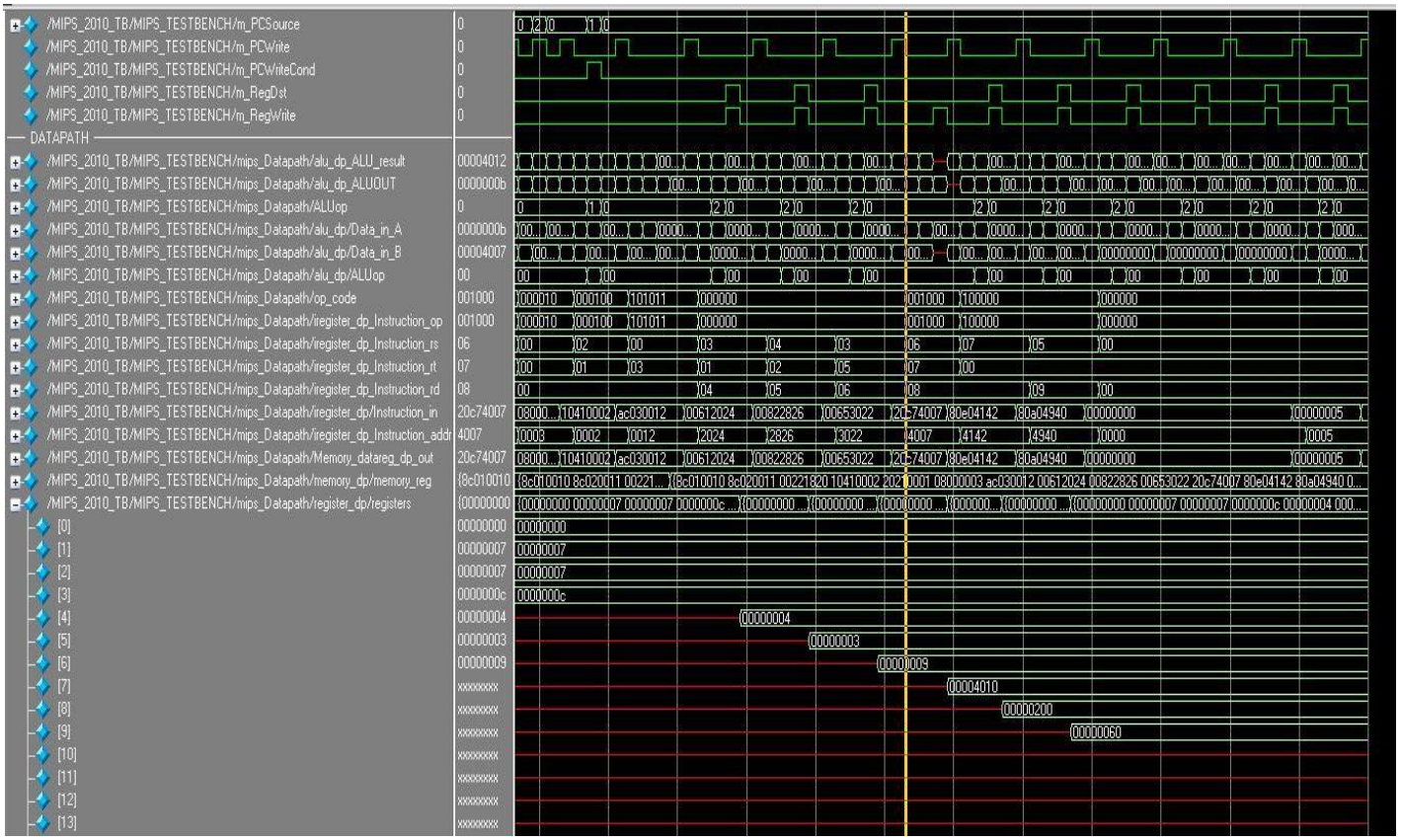
MEMORY VALUES



DATA REGISTERS VALUES



DATA REGISTERS VALUES CONTINUE



Synthesized Report for Control Unit

```
Release 11.4 - xst L.68 (nt)
Copyright (c) 1995-2009 Xilinx, Inc.  All rights reserved.
--> Parameter TMPDIR set to xst/projnav.tmp
Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.16 secs

--> Parameter xsthdpdir set to xst
Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.16 secs

--> Reading design: Control_unit.prj
TABLE OF CONTENTS
  1) Synthesis Options Summary
  2) HDL Compilation
  3) Design Hierarchy Analysis
  4) HDL Analysis
  5) HDL Synthesis
     5.1) HDL Synthesis Report
  6) Advanced HDL Synthesis
     6.1) Advanced HDL Synthesis Report
  7) Low Level Synthesis
  8) Partition Report
  9) Final Report
        9.1) Device utilization summary
        9.2) Partition Resource Summary
        9.3) TIMING REPORT
=========================================================================
*                      Synthesis Options Summary                        *
=========================================================================
---- Source Parameters
Input File Name                    : "Control_unit.prj"
Input Format                       : mixed
Ignore Synthesis Constraint File   : NO
---- Target Parameters
Output File Name                   : "Control_unit"
Output Format                      : NGC
Target Device                      : xc3s500e-5-fg320
---- Source Options
Top Module Name                    : Control_unit
Automatic FSM Extraction           : YES
FSM Encoding Algorithm             : Auto
Safe Implementation                : No
FSM Style                          : lut
RAM Extraction                     : Yes
RAM Style                          : Auto
ROM Extraction                     : Yes
Mux Style                          : Auto
Decoder Extraction                 : YES
Priority Encoder Extraction        : YES
Shift Register Extraction          : YES
Logical Shifter Extraction         : YES
XOR Collapsing                     : YES
ROM Style                          : Auto
Mux Extraction                     : YES
Resource Sharing                   : YES
Asynchronous To Synchronous        : NO
Multiplier Style                   : auto
Automatic Register Balancing       : No
---- Target Options
```

```
Add IO Buffers                      : YES
Global Maximum Fanout               : 500
Add Generic Clock Buffer(BUFG)      : 24
Register Duplication                : YES
Slice Packing                       : YES
Optimize Instantiated Primitives    : NO
Use Clock Enable                    : Yes
Use Synchronous Set                 : Yes
Use Synchronous Reset               : Yes
Pack IO Registers into IOBs         : auto
Equivalent register Removal         : YES
---- General Options
Optimization Goal                   : Speed
Optimization Effort                 : 1
Library Search Order                : Control_unit.lso
Keep Hierarchy                      : NO
Netlist Hierarchy                   : as_optimized
RTL Output                          : Yes
Global Optimization                 : AllClockNets
Read Cores                          : YES
Write Timing Constraints            : NO
Cross Clock Analysis                : NO
Hierarchy Separator                 : /
Bus Delimiter                       : <>
Case Specifier                      : maintain
Slice Utilization Ratio             : 100
BRAM Utilization Ratio              : 100
Verilog 2001                        : YES
Auto BRAM Packing                   : NO
Slice Utilization Ratio Delta       : 5
=========================================================================
=========================================================================
*                           HDL Compilation                             *
=========================================================================
Compiling verilog file "Control_unit.v" in library work
Module <Control_unit> compiled
No errors in compilation
Analysis of file <"Control_unit.prj"> succeeded.

=========================================================================
*                     Design Hierarchy Analysis                         *
=========================================================================
Analyzing hierarchy for module <Control_unit> in library <work> with parameters.
        B_EQ = "000100"
        Branch = "1000"
        Complete = "1100"
        Decode = "0001"
        Execution = "0110"
        Fetch = "0000"
        IType = "1010"
        IType_complete = "1011"
        I_type = "001000"
        J_ump = "000010"
        Jump = "1001"
        Lw = "100011"
        Memory_access = "0011"
        Memory_comp = "0010"
        Memory_read = "0100"
        Memory_write = "0101"
        R_type = "000000"
        Reset = "1111"
```

```
        Rtype = "0111"
        Shift = "100000"
        Sw = "101011"
=========================================================================
*                            HDL Analysis                               *
=========================================================================
Analyzing top module <Control_unit>.
        B_EQ = 6'b000100
        Branch = 4'b1000
        Complete = 4'b1100
        Decode = 4'b0001
        Execution = 4'b0110
        Fetch = 4'b0000
        IType = 4'b1010
        IType_complete = 4'b1011
        I_type = 6'b001000
        J_ump = 6'b000010
        Jump = 4'b1001
        Lw = 6'b100011
        Memory_access = 4'b0011
        Memory_comp = 4'b0010
        Memory_read = 4'b0100
        Memory_write = 4'b0101
        R_type = 6'b000000
        Reset = 4'b1111
        Rtype = 4'b0111
        Shift = 6'b100000
        Sw = 6'b101011
Module <Control_unit> is correct for synthesis.

=========================================================================
*                            HDL Synthesis                              *
=========================================================================
Performing bidirectional port resolution...
Synthesizing Unit <Control_unit>.
    Related source file is "Control_unit.v".
    Found finite state machine <FSM_0> for signal <current_state>.
    -----------------------------------------------------------------------
    | States             | 14                                           |
    | Transitions        | 22                                           |
    | Inputs             | 7                                            |
    | Outputs            | 16                                           |
    | Clock              | clk                          (rising_edge)   |
    | Reset              | rst                          (negative)      |
    | Reset type         | asynchronous                                 |
    | Reset State        | 1111                                         |
    | Encoding           | automatic                                    |
    | Implementation     | LUT                                          |
    -----------------------------------------------------------------------
    Summary:
        inferred   1 Finite State Machine(s).
Unit <Control_unit> synthesized.
=========================================================================
HDL Synthesis Report
Found no macro
=========================================================================
=========================================================================
*                       Advanced HDL Synthesis                          *
=========================================================================
Analyzing FSM <FSM_0> for best encoding.
```

```
Optimizing FSM <current_state/FSM> on signal <current_state[1:14]> with one-hot
encoding.
-------------------------
 State | Encoding
-------------------------
 0000  | 00000000000010
 0001  | 00000000000100
 0010  | 00000000001000
 0011  | 00000100000000
 0100  | 00010000000000
 0101  | 00001000000000
 0110  | 00000000010000
 0111  | 01000000000000
 1000  | 00000001000000
 1001  | 00000010000000
 1010  | 00000000100000
 1011  | 10000000000000
 1100  | 00100000000000
 1111  | 00000000000001
-------------------------
================================================================================
Advanced HDL Synthesis Report
Macro Statistics
# FSMs                                                 : 1
================================================================================
================================================================================
*                         Low Level Synthesis                                  *
================================================================================
Optimizing unit <Control_unit> ...
Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block Control_unit, actual ratio is 0.
Final Macro Processing ...
================================================================================
Final Register Report
Macro Statistics
# Registers                                            : 14
 Flip-Flops                                            : 14
================================================================================
================================================================================
*                           Partition Report                                   *
================================================================================
Partition Implementation Status
-------------------------------
  No Partitions were found in this design.
-------------------------------
================================================================================
*                            Final Report                                      *
================================================================================
Final Results
RTL Top Level Output File Name     : Control_unit.ngr
Top Level Output File Name         : Control_unit
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO
Design Statistics
# IOs                              : 24
Cell Usage :
# BELS                             : 34
#      GND                         : 1
#      INV                         : 1
```

```
#      LUT2                        : 3
#      LUT3                        : 8
#      LUT4                        : 20
#      MUXF5                       : 1
# FlipFlops/Latches                : 14
#      FDC                         : 13
#      FDP                         : 1
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 23
#      IBUF                        : 7
#      OBUF                        : 16
=========================================================================
Device utilization summary:
---------------------------
Selected Device : 3s500efg320-5
 Number of Slices:                      18  out of   4656     0%
 Number of Slice Flip Flops:            14  out of   9312     0%
 Number of 4 input LUTs:                32  out of   9312     0%
 Number of IOs:                         24
 Number of bonded IOBs:                 24  out of    232    10%
 Number of GCLKs:                        1  out of     24     4%
---------------------------
Partition Resource Summary:
---------------------------
  No Partitions were found in this design.
---------------------------
=========================================================================
TIMING REPORT
NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.
Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
clk                                | BUFGP                  | 14    |
-----------------------------------+------------------------+-------+
Asynchronous Control Signals Information:
----------------------------------------
-------------------------------------------------------------------------+--------------
---------------+-------+
Control Signal                                                           | Buffer(FF
name)             | Load  |
-------------------------------------------------------------------------+--------------
---------------+-------+
current_state_FSM_Acst_FSM_inv(current_state_FSM_Acst_FSM_inv1_INV_0:O)|
NONE(current_state_FSM_FFd1)| 14    |
-------------------------------------------------------------------------+--------------
---------------+-------+
Timing Summary:
---------------
Speed Grade: -5
   Minimum period: 2.040ns (Maximum Frequency: 490.088MHz)
   Minimum input arrival time before clock: 4.925ns
   Maximum output required time after clock: 6.275ns
   Maximum combinational path delay: 8.239ns
Timing Detail:
--------------
All values displayed in nanoseconds (ns)
```

```
=========================================================================
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 2.040ns (frequency: 490.088MHz)
  Total number of paths / destination ports: 20 / 13
-------------------------------------------------------------------------
Delay:               2.040ns (Levels of Logic = 1)
  Source:            current_state_FSM_FFd12 (FF)
  Destination:       current_state_FSM_FFd7 (FF)
  Source Clock:      clk rising
  Destination Clock: clk rising
  Data Path: current_state_FSM_FFd12 to current_state_FSM_FFd7
                              Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDC:C->Q              8   0.514   0.646  current_state_FSM_FFd12
(current_state_FSM_FFd12)
     LUT4:I3->O            1   0.612   0.000  current_state_FSM_FFd11-In
(current_state_FSM_FFd11-In)
     FDC:D                     0.268          current_state_FSM_FFd11
    ----------------------------------------
    Total                      2.040ns (1.394ns logic, 0.646ns route)
                                       (68.3% logic, 31.7% route)
=========================================================================
Timing constraint: Default OFFSET IN BEFORE for Clock 'clk'
  Total number of paths / destination ports: 55 / 8
-------------------------------------------------------------------------
Offset:              4.925ns (Levels of Logic = 4)
  Source:            op_code<2> (PAD)
  Destination:       current_state_FSM_FFd12 (FF)
  Destination Clock: clk rising
  Data Path: op_code<2> to current_state_FSM_FFd12
                              Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O            10   1.106   0.902  op_code_2_IBUF (op_code_2_IBUF)
     LUT4:I0->O            1   0.612   0.387  current_state_FSM_FFd12-In24
(current_state_FSM_FFd12-In24)
     LUT3:I2->O            1   0.612   0.426  current_state_FSM_FFd12-In26
(current_state_FSM_FFd12-In26)
     LUT4:I1->O            1   0.612   0.000  current_state_FSM_FFd12-In85
(current_state_FSM_FFd12-In)
     FDC:D                     0.268          current_state_FSM_FFd12
    ----------------------------------------
    Total                      4.925ns (3.210ns logic, 1.715ns route)
                                       (65.2% logic, 34.8% route)
=========================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'
  Total number of paths / destination ports: 32 / 16
-------------------------------------------------------------------------
Offset:              6.275ns (Levels of Logic = 3)
  Source:            current_state_FSM_FFd4 (FF)
  Destination:       MemRead (PAD)
  Source Clock:      clk rising
  Data Path: current_state_FSM_FFd4 to MemRead
                              Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDC:C->Q              4   0.514   0.651  current_state_FSM_FFd4
(current_state_FSM_FFd4)
     LUT3:I0->O            1   0.612   0.360  current_state_FSM_Out13_SW0 (N21)
     LUT4:I3->O            1   0.612   0.357  current_state_FSM_Out13 (MemRead_OBUF)
```

```
     OBUF:I->O                 3.169          MemRead_OBUF (MemRead)
    ----------------------------------------
    Total                      6.275ns (4.907ns logic, 1.368ns route)
                                       (78.2% logic, 21.8% route)
=========================================================================
Timing constraint: Default path analysis
  Total number of paths / destination ports: 6 / 1
-------------------------------------------------------------------------
Delay:               8.239ns (Levels of Logic = 5)
  Source:            op_code<5> (PAD)
  Destination:       ALUSrcB<1> (PAD)
  Data Path: op_code<5> to ALUSrcB<1>
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O             10   1.106   0.902  op_code_5_IBUF (op_code_5_IBUF)
     LUT4:I0->O             1   0.612   0.509  ALUSrcB<1>21 (ALUSrcB<1>21)
     LUT4:I0->O             1   0.612   0.360  ALUSrcB<1>33_SW0 (N27)
     LUT4:I3->O             1   0.612   0.357  ALUSrcB<1>33 (ALUSrcB_1_OBUF)
     OBUF:I->O                  3.169          ALUSrcB_1_OBUF (ALUSrcB<1>)
    ----------------------------------------
    Total                      8.239ns (6.111ns logic, 2.128ns route)
                                       (74.2% logic, 25.8% route)
=========================================================================
Total REAL time to Xst completion: 4.00 secs
Total CPU time to Xst completion: 4.17 secs

-->
Total memory usage is 136924 kilobytes
Number of errors   :    0 (   0 filtered)
Number of warnings :    0 (   0 filtered)
Number of infos    :    0 (   0 filtered)
```

Synthesized Report for DataPath Unit

```
Release 11.4 - xst L.68 (nt)
Copyright (c) 1995-2009 Xilinx, Inc.  All rights reserved.
--> Parameter TMPDIR set to xst/projnav.tmp
Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.14 secs

--> Parameter xsthdpdir set to xst
Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.16 secs

--> Reading design: Datapath.prj
TABLE OF CONTENTS
  1) Synthesis Options Summary
  2) HDL Compilation
  3) Design Hierarchy Analysis
  4) HDL Analysis
  5) HDL Synthesis
     5.1) HDL Synthesis Report
  6) Advanced HDL Synthesis
     6.1) Advanced HDL Synthesis Report
  7) Low Level Synthesis
  8) Partition Report
  9) Final Report
        9.1) Device utilization summary
        9.2) Partition Resource Summary
        9.3) TIMING REPORT
```

```
=========================================================================
*                     Synthesis Options Summary                         *
=========================================================================
---- Source Parameters
Input File Name                    : "Datapath.prj"
Input Format                       : mixed
Ignore Synthesis Constraint File   : NO
---- Target Parameters
Output File Name                   : "Datapath"
Output Format                      : NGC
Target Device                      : xc3s500e-5-fg320
---- Source Options
Top Module Name                    : Datapath
Automatic FSM Extraction           : YES
FSM Encoding Algorithm             : Auto
Safe Implementation                : No
FSM Style                          : lut
RAM Extraction                     : Yes
RAM Style                          : Auto
ROM Extraction                     : Yes
Mux Style                          : Auto
Decoder Extraction                 : YES
Priority Encoder Extraction        : YES
Shift Register Extraction          : YES
Logical Shifter Extraction         : YES
XOR Collapsing                     : YES
ROM Style                          : Auto
Mux Extraction                     : YES
Resource Sharing                   : YES
Asynchronous To Synchronous        : NO
Multiplier Style                   : auto
Automatic Register Balancing       : No
---- Target Options
Add IO Buffers                     : YES
Global Maximum Fanout              : 500
Add Generic Clock Buffer(BUFG)     : 24
Register Duplication               : YES
Slice Packing                      : YES
Optimize Instantiated Primitives   : NO
Use Clock Enable                   : Yes
Use Synchronous Set                : Yes
Use Synchronous Reset              : Yes
Pack IO Registers into IOBs        : auto
Equivalent register Removal        : YES
---- General Options
Optimization Goal                  : Speed
Optimization Effort                : 1
Library Search Order               : Datapath.lso
Keep Hierarchy                     : NO
Netlist Hierarchy                  : as_optimized
RTL Output                         : Yes
Global Optimization                : AllClockNets
Read Cores                         : YES
Write Timing Constraints           : NO
Cross Clock Analysis               : NO
Hierarchy Separator                : /
Bus Delimiter                      : <>
Case Specifier                     : maintain
Slice Utilization Ratio            : 100
BRAM Utilization Ratio             : 100
Verilog 2001                       : YES
```

```
Auto BRAM Packing                   : NO
Slice Utilization Ratio Delta       : 5
=========================================================================
=========================================================================
*                          HDL Compilation                              *
=========================================================================
Compiling verilog file "../Mux2_Special/Mux2SP_unit.v" in library work
Compiling verilog file "Sign_ext_unit.v" in library work
Module <Mux2SP_unit> compiled
Compiling verilog file "Shift_left_32.v" in library work
Module <Sign_ext_unit> compiled
Compiling verilog file "Shift2rl_unit.v" in library work
Module <Shift_left_32> compiled
Compiling verilog file "Register_unit.v" in library work
Module <Shift2rl_unit> compiled
Compiling verilog file "P_counter.v" in library work
Module <Register_unit> compiled
Compiling verilog file "Mux4_unit.v" in library work
Module <P_counter> compiled
Compiling verilog file "Mux3_unit.v" in library work
Module <Mux4_unit> compiled
Compiling verilog file "Mux2_unit.v" in library work
Module <Mux3_unit> compiled
Compiling verilog file "memory_unit.v" in library work
Module <Mux2_unit> compiled
Compiling verilog file "Memory_datareg.v" in library work
Module <memory_unit> compiled
Compiling verilog file "Instruction_register_unit.v" in library work
Module <Memory_datareg> compiled
Compiling verilog file "ALU_unit.v" in library work
Module <Instruction_register_unit> compiled
Compiling verilog file "Data_path.v" in library work
Module <ALU_unit> compiled
Module <Datapath> compiled
No errors in compilation
Analysis of file <"Datapath.prj"> succeeded.

=========================================================================
*                     Design Hierarchy Analysis                         *
=========================================================================
Analyzing hierarchy for module <Datapath> in library <work>.
Analyzing hierarchy for module <Mux3_unit> in library <work>.
Analyzing hierarchy for module <P_counter> in library <work>.
Analyzing hierarchy for module <Mux2SP_unit> in library <work>.
Analyzing hierarchy for module <Mux2_unit> in library <work>.
Analyzing hierarchy for module <memory_unit> in library <work>.
Analyzing hierarchy for module <Memory_datareg> in library <work>.
Analyzing hierarchy for module <Instruction_register_unit> in library <work> with
parameters.
        word_size = "00000000000000000000000000100000"
Analyzing hierarchy for module <Register_unit> in library <work> with parameters.
        word_size = "00000000000000000000000000100000"
Analyzing hierarchy for module <Mux4_unit> in library <work>.
Analyzing hierarchy for module <ALU_unit> in library <work> with parameters.
        ADD = "100000"
        AND = "100100"
        Addi = "00"
        NOP = "101100"
        NOR = "100111"
        OR = "100101"
        R_type = "10"
```

```
        SLL = "000000"
        SLT = "101010"
        SRL = "000010"
        SUB = "100010"
        Subi = "01"
        XOR = "100110"
        word_size = "00000000000000000000000000100000"
Analyzing hierarchy for module <Shift2rl_unit> in library <work>.
Analyzing hierarchy for module <Shift_left_32> in library <work>.
Analyzing hierarchy for module <Sign_ext_unit> in library <work>.
=========================================================================
*                            HDL Analysis                               *
=========================================================================
Analyzing top module <Datapath>.
Module <Datapath> is correct for synthesis.

Analyzing module <Mux3_unit> in library <work>.
Module <Mux3_unit> is correct for synthesis.

Analyzing module <P_counter> in library <work>.
Module <P_counter> is correct for synthesis.

Analyzing module <Mux2SP_unit> in library <work>.
Module <Mux2SP_unit> is correct for synthesis.

Analyzing module <Mux2_unit> in library <work>.
Module <Mux2_unit> is correct for synthesis.

Analyzing module <memory_unit> in library <work>.
Module <memory_unit> is correct for synthesis.

Analyzing module <Memory_datareg> in library <work>.
Module <Memory_datareg> is correct for synthesis.

Analyzing module <Instruction_register_unit> in library <work>.
        word_size = 32'sb00000000000000000000000000100000
Module <Instruction_register_unit> is correct for synthesis.

Analyzing module <Register_unit> in library <work>.
        word_size = 32'sb00000000000000000000000000100000
Module <Register_unit> is correct for synthesis.

Analyzing module <Mux4_unit> in library <work>.
Module <Mux4_unit> is correct for synthesis.

Analyzing module <ALU_unit> in library <work>.
        ADD = 6'b100000
        AND = 6'b100100
        Addi = 2'b00
        NOP = 6'b101100
        NOR = 6'b100111
        OR = 6'b100101
        R_type = 2'b10
        SLL = 6'b000000
        SLT = 6'b101010
        SRL = 6'b000010
        SUB = 6'b100010
        Subi = 2'b01
        XOR = 6'b100110
        word_size = 32'sb00000000000000000000000000100000
Module <ALU_unit> is correct for synthesis.
```

```
Analyzing module <Shift2rl_unit> in library <work>.
Module <Shift2rl_unit> is correct for synthesis.

Analyzing module <Shift_left_32> in library <work>.
Module <Shift_left_32> is correct for synthesis.

Analyzing module <Sign_ext_unit> in library <work>.
Module <Sign_ext_unit> is correct for synthesis.

=========================================================================
*                            HDL Synthesis                              *
=========================================================================
Performing bidirectional port resolution...
Synthesizing Unit <Mux3_unit>.
    Related source file is "Mux3_unit.v".
    Found 32-bit 4-to-1 multiplexer for signal <Out_mux>.
    Summary:
        inferred  32 Multiplexer(s).
Unit <Mux3_unit> synthesized.
Synthesizing Unit <P_counter>.
    Related source file is "P_counter.v".
    Found 32-bit register for signal <pc_address>.
    Summary:
        inferred  32 D-type flip-flop(s).
Unit <P_counter> synthesized.
Synthesizing Unit <Mux2SP_unit>.
    Related source file is "../Mux2_Special/Mux2SP_unit.v".
Unit <Mux2SP_unit> synthesized.
Synthesizing Unit <Mux2_unit>.
    Related source file is "Mux2_unit.v".
Unit <Mux2_unit> synthesized.
Synthesizing Unit <memory_unit>.
    Related source file is "memory_unit.v".
    Found 32-bit register for signal <MEMData>.
    Found 1024-bit register for signal <memory_reg>.
INFO:Xst:738 - HDL ADVISOR - 1024 flip-flops were inferred for signal <memory_reg>. You
may be trying to describe a RAM in a way that is incompatible with block and distributed
RAM resources available on Xilinx devices, or with a specific template that is not
supported. Please review the Xilinx resources documentation and the XST user manual for
coding guidelines. Taking advantage of RAM resources will lead to improved device usage
and reduced synthesis time.
    Summary:
        inferred 576 D-type flip-flop(s).
        inferred  32 Multiplexer(s).
Unit <memory_unit> synthesized.
Synthesizing Unit <Memory_datareg>.
    Related source file is "Memory_datareg.v".
Unit <Memory_datareg> synthesized.
Synthesizing Unit <Instruction_register_unit>.
    Related source file is "Instruction_register_unit.v".
    Found 16-bit register for signal <Instruction_addr>.
    Found 6-bit register for signal <Instruction_op>.
    Found 5-bit register for signal <Instruction_rd>.
    Found 5-bit register for signal <Instruction_rs>.
    Found 5-bit register for signal <Instruction_rt>.
    Summary:
        inferred  37 D-type flip-flop(s).
Unit <Instruction_register_unit> synthesized.
Synthesizing Unit <Register_unit>.
    Related source file is "Register_unit.v".
```

```
    Found 32-bit register for signal <Reg_A>.
    Found 32-bit register for signal <Reg_B>.
    Found 1024-bit register for signal <registers>.
INFO:Xst:738 - HDL ADVISOR - 1024 flip-flops were inferred for signal <registers>. You
may be trying to describe a RAM in a way that is incompatible with block and distributed
RAM resources available on Xilinx devices, or with a specific template that is not
supported. Please review the Xilinx resources documentation and the XST user manual for
coding guidelines. Taking advantage of RAM resources will lead to improved device usage
and reduced synthesis time.
    Summary:
        inferred 1088 D-type flip-flop(s).
        inferred  64 Multiplexer(s).
Unit <Register_unit> synthesized.
Synthesizing Unit <Mux4_unit>.
    Related source file is "Mux4_unit.v".
    Found 32-bit 4-to-1 multiplexer for signal <Out_mux>.
    Summary:
        inferred  32 Multiplexer(s).
Unit <Mux4_unit> synthesized.
Synthesizing Unit <ALU_unit>.
    Related source file is "ALU_unit.v".
    Found 32-bit 3-to-1 multiplexer for signal <ALU_result>.
    Found 32-bit register for signal <ALUOUT>.
    Found 32-bit addsub for signal <ALU_result$share0000> created at line 36.
    Found 32-bit shifter logical right for signal <ALU_result$shift0000> created at line
56.
    Found 32-bit shifter logical left for signal <ALU_result$shift0001> created at line
58.
    Found 32-bit xor2 for signal <ALU_result$xor0000> created at line 50.
    Summary:
        inferred  32 D-type flip-flop(s).
        inferred   1 Adder/Subtractor(s).
        inferred  32 Multiplexer(s).
        inferred   2 Combinational logic shifter(s).
Unit <ALU_unit> synthesized.
Synthesizing Unit <Shift2rl_unit>.
    Related source file is "Shift2rl_unit.v".
Unit <Shift2rl_unit> synthesized.
Synthesizing Unit <Shift_left_32>.
    Related source file is "Shift_left_32.v".
Unit <Shift_left_32> synthesized.
Synthesizing Unit <Sign_ext_unit>.
    Related source file is "Sign_ext_unit.v".
Unit <Sign_ext_unit> synthesized.
Synthesizing Unit <Datapath>.
    Related source file is "Data_path.v".
Unit <Datapath> synthesized.
INFO:Xst:1767 - HDL ADVISOR - Resource sharing has identified that some arithmetic
operations in this design can share the same physical resources for reduced device
utilization. For improved clock frequency you may try to disable resource sharing.
=========================================================================
HDL Synthesis Report
Macro Statistics
# Adders/Subtractors                                   : 1
 32-bit addsub                                        : 1
# Registers                                            : 74
 16-bit register                                      : 1
 32-bit register                                      : 69
 5-bit register                                       : 3
 6-bit register                                       : 1
# Multiplexers                                         : 6
```

```
 32-bit 3-to-1 multiplexer                              : 1
 32-bit 32-to-1 multiplexer                             : 3
 32-bit 4-to-1 multiplexer                              : 2
# Logic shifters                                        : 2
 32-bit shifter logical left                            : 1
 32-bit shifter logical right                           : 1
# Xors                                                  : 1
 32-bit xor2                                            : 1
=========================================================================
=========================================================================
*                       Advanced HDL Synthesis                          *
=========================================================================
=========================================================================
Advanced HDL Synthesis Report
Macro Statistics
# Adders/Subtractors                                    : 1
 32-bit addsub                                          : 1
# Registers                                             : 2245
 Flip-Flops                                             : 2245
# Multiplexers                                          : 68
 1-bit 32-to-1 multiplexer                              : 64
 32-bit 3-to-1 multiplexer                              : 1
 32-bit 32-to-1 multiplexer                             : 1
 32-bit 4-to-1 multiplexer                              : 2
# Logic shifters                                        : 2
 32-bit shifter logical left                            : 1
 32-bit shifter logical right                           : 1
# Xors                                                  : 1
 32-bit xor2                                            : 1
=========================================================================
=========================================================================
*                         Low Level Synthesis                           *
=========================================================================
Optimizing unit <Datapath> ...
Optimizing unit <P_counter> ...
Optimizing unit <Instruction_register_unit> ...
Optimizing unit <Register_unit> ...
Optimizing unit <ALU_unit> ...
Optimizing unit <memory_unit> ...
Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block Datapath, actual ratio is 51.
FlipFlop iregister_dp/Instruction_addr_0 has been replicated 1 time(s)
FlipFlop iregister_dp/Instruction_addr_3 has been replicated 1 time(s)
FlipFlop iregister_dp/Instruction_addr_5 has been replicated 1 time(s)
FlipFlop iregister_dp/Instruction_rs_0 has been replicated 1 time(s)
FlipFlop iregister_dp/Instruction_rt_0 has been replicated 1 time(s)
FlipFlop register_dp/Reg_B_10 has been replicated 1 time(s)
FlipFlop register_dp/Reg_B_6 has been replicated 1 time(s)
FlipFlop register_dp/Reg_B_7 has been replicated 1 time(s)
FlipFlop register_dp/Reg_B_9 has been replicated 1 time(s)
Final Macro Processing ...
=========================================================================
Final Register Report
Macro Statistics
# Registers                                             : 2254
 Flip-Flops                                             : 2254
=========================================================================
=========================================================================
*                         Partition Report                              *
=========================================================================
```

```
Partition Implementation Status
-------------------------------
  No Partitions were found in this design.
-------------------------------
=========================================================================
*                           Final Report                                *
=========================================================================
Final Results
RTL Top Level Output File Name     : Datapath.ngr
Top Level Output File Name         : Datapath
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO
Design Statistics
# IOs                              : 24
Cell Usage :
# BELS                             : 4077
#      INV                         : 3
#      LUT2                        : 17
#      LUT2_D                      : 2
#      LUT2_L                      : 7
#      LUT3                        : 1866
#      LUT3_D                      : 20
#      LUT3_L                      : 27
#      LUT4                        : 420
#      LUT4_D                      : 11
#      LUT4_L                      : 62
#      MUXCY                       : 31
#      MUXF5                       : 907
#      MUXF6                       : 384
#      MUXF7                       : 192
#      MUXF8                       : 96
#      XORCY                       : 32
# FlipFlops/Latches                : 2254
#      FD                          : 32
#      FDCE                        : 1039
#      FDE                         : 1097
#      FDPE                        : 86
# Clock Buffers                    : 1
#      BUFGP                       : 1
# IO Buffers                       : 23
#      IBUF                        : 17
#      OBUF                        : 6
=========================================================================
Device utilization summary:
---------------------------
Selected Device : 3s500efg320-5
 Number of Slices:                     2387  out of   4656    51%
 Number of Slice Flip Flops:           2254  out of   9312    24%
 Number of 4 input LUTs:               2435  out of   9312    26%
 Number of IOs:                          24
 Number of bonded IOBs:                  24  out of    232    10%
 Number of GCLKs:                         1  out of     24     4%
---------------------------
Partition Resource Summary:
---------------------------
  No Partitions were found in this design.
---------------------------
=========================================================================
TIMING REPORT
NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
```

```
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.
Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
clk                                | BUFGP                  | 2254  |
-----------------------------------+------------------------+-------+
Asynchronous Control Signals Information:
----------------------------------------
-----------------------------------------------------------------+------------------------
------------+-------+
Control Signal                                                   | Buffer(FF name)
| Load  |
-----------------------------------------------------------------+------------------------
------------+-------+
Pcounter_dp/rst_inv(register_dp/rst_inv1_INV_0:O)                                          |
NONE(iregister_dp/Instruction_addr_0)| 375   |
register_dp/rst_inv1_INV_0_1(register_dp/rst_inv1_INV_0_1:O)|
NONE(Pcounter_dp/pc_address_0)       | 375   |
register_dp/rst_inv1_INV_0_2(register_dp/rst_inv1_INV_0_2:O)|
NONE(memory_dp/memory_reg_19_25)     | 375   |
-----------------------------------------------------------------+------------------------
------------+-------+
Timing Summary:
---------------
Speed Grade: -5
   Minimum period: 13.066ns (Maximum Frequency: 76.536MHz)
   Minimum input arrival time before clock: 14.798ns
   Maximum output required time after clock: 4.040ns
   Maximum combinational path delay: No path found
Timing Detail:
--------------
All values displayed in nanoseconds (ns)
=========================================================================
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 13.066ns (frequency: 76.536MHz)
  Total number of paths / destination ports: 485898 / 4402
-------------------------------------------------------------------------
Delay:               13.066ns (Levels of Logic = 11)
  Source:            iregister_dp/Instruction_addr_6 (FF)
  Destination:       Pcounter_dp/pc_address_31 (FF)
  Source Clock:      clk rising
  Destination Clock: clk rising
  Data Path: iregister_dp/Instruction_addr_6 to Pcounter_dp/pc_address_31
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDCE:C->Q                 3    0.514     0.454   iregister_dp/Instruction_addr_6
(iregister_dp/Instruction_addr_6)
     LUT4:I3->O          135   0.612   1.101  mux4_dp/Mmux_Out_mux291 (mux4_dp_out<6>)
     MUXF5:S->O            3   0.641   0.481  alu_dp/Sh115_SW1 (N238)
     LUT3_D:I2->O          3   0.612   0.481  alu_dp/Sh113 (alu_dp/Sh113)
     LUT3:I2->O            1   0.612   0.000  alu_dp/Mmux_ALU_result22227_G (N478)
     MUXF5:I1->O                1     0.278     0.360    alu_dp/Mmux_ALU_result22227
(alu_dp/Mmux_ALU_result22227)
     LUT4_L:I3->LO              1     0.612     0.103    alu_dp/Mmux_ALU_result22272
(alu_dp/Mmux_ALU_result22272)
     LUT4:I3->O                 1     0.612     0.360    alu_dp/Mmux_ALU_result22323
(alu_dp/Mmux_ALU_result22323)
```

```
     LUT4:I3->O                    3     0.612     0.454    alu_dp/Mmux_ALU_result22353
(alu_dp_ALU_result<29>)
     LUT4:I3->O            1   0.612   0.387  Pcounter_dp/load_pc48_SW0_SW0 (N314)
     LUT4:I2->O          1   0.612   0.387  Pcounter_dp/load_pc48 (Pcounter_dp/load_pc48)
     LUT4:I2->O         32   0.612   1.073  Pcounter_dp/load_pc216 (Pcounter_dp/load_pc)
     FDCE:CE                   0.483          Pcounter_dp/pc_address_0
    ----------------------------------------
    Total                     13.066ns (7.424ns logic, 5.642ns route)
                                       (56.8% logic, 43.2% route)
=========================================================================
Timing constraint: Default OFFSET IN BEFORE for Clock 'clk'
  Total number of paths / destination ports: 290709 / 3342
-------------------------------------------------------------------------
Offset:              14.798ns (Levels of Logic = 11)
  Source:            ALUSrcB<1> (PAD)
  Destination:       Pcounter_dp/pc_address_31 (FF)
  Destination Clock: clk rising
  Data Path: ALUSrcB<1> to Pcounter_dp/pc_address_31
                              Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O            34   1.106   1.225  ALUSrcB_1_IBUF (ALUSrcB_1_IBUF)
     LUT2:I0->O           39   0.612   1.227  mux4_dp/Mmux_Out_mux1211 (N2)
     LUT4:I0->O          110   0.612   1.163  mux4_dp/Mmux_Out_mux311 (mux4_dp_out<8>)
     LUT2:I1->O            4   0.612   0.502  alu_dp/Sh14011 (alu_dp/N2)
     LUT4:I3->O            1   0.612   0.360  alu_dp/Mmux_ALU_result1171_SW0 (N671)
     LUT4_L:I3->LO             1      0.612      0.103    alu_dp/Mmux_ALU_result1171
(alu_dp/Mmux_ALU_result1171)
     LUT4:I3->O                1      0.612      0.360    alu_dp/Mmux_ALU_result168
(alu_dp/Mmux_ALU_result168)
     LUT4:I3->O                3      0.612      0.520    alu_dp/Mmux_ALU_result1227
(alu_dp_ALU_result<0>)
     LUT4:I1->O          1   0.612   0.387  Pcounter_dp/load_pc84 (Pcounter_dp/load_pc84)
     LUT4_L:I2->LO             1      0.612      0.169    Pcounter_dp/load_pc109
(Pcounter_dp/load_pc109)
     LUT4:I1->O         32   0.612   1.073  Pcounter_dp/load_pc216 (Pcounter_dp/load_pc)
     FDCE:CE                   0.483          Pcounter_dp/pc_address_0
    ----------------------------------------
    Total                     14.798ns (7.709ns logic, 7.089ns route)
                                       (52.1% logic, 47.9% route)
=========================================================================
Timing constraint: Default OFFSET OUT AFTER for Clock 'clk'
  Total number of paths / destination ports: 6 / 6
-------------------------------------------------------------------------
Offset:              4.040ns (Levels of Logic = 1)
  Source:            iregister_dp/Instruction_op_5 (FF)
  Destination:       op_code<5> (PAD)
  Source Clock:      clk rising
  Data Path: iregister_dp/Instruction_op_5 to op_code<5>
                              Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     FDCE:C->Q                 1     0.514     0.357   iregister_dp/Instruction_op_5
(iregister_dp/Instruction_op_5)
     OBUF:I->O                 3.169          op_code_5_OBUF (op_code<5>)
    ----------------------------------------
    Total                     4.040ns (3.683ns logic, 0.357ns route)
                                       (91.2% logic, 8.8% route)
=========================================================================
Total REAL time to Xst completion: 42.00 secs
Total CPU time to Xst completion: 41.98 secs
```

```
-->
Total memory usage is 171356 kilobytes
Number of errors   :    0 (   0 filtered)
Number of warnings :    0 (   0 filtered)
Number of infos    :    3 (   0 filtered)

                                    MIPS REPORT


Release 11.4 - xst L.68 (nt)
Copyright (c) 1995-2009 Xilinx, Inc.  All rights reserved.
--> Parameter TMPDIR set to xst/projnav.tmp
Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.16 secs

--> Parameter xsthdpdir set to xst
Total REAL time to Xst completion: 0.00 secs
Total CPU time to Xst completion: 0.16 secs

--> Reading design: Mips.prj
TABLE OF CONTENTS
  1) Synthesis Options Summary
  2) HDL Compilation
  3) Design Hierarchy Analysis
  4) HDL Analysis
  5) HDL Synthesis
     5.1) HDL Synthesis Report
  6) Advanced HDL Synthesis
     6.1) Advanced HDL Synthesis Report
  7) Low Level Synthesis
  8) Partition Report
  9) Final Report
        9.1) Device utilization summary
        9.2) Partition Resource Summary
        9.3) TIMING REPORT
=========================================================================
*                     Synthesis Options Summary                         *
=========================================================================
---- Source Parameters
Input File Name                    : "Mips.prj"
Input Format                       : mixed
Ignore Synthesis Constraint File   : NO
---- Target Parameters
Output File Name                   : "Mips"
Output Format                      : NGC
Target Device                      : xc3s500e-5-fg320
---- Source Options
Top Module Name                    : Mips
Automatic FSM Extraction           : YES
FSM Encoding Algorithm             : Auto
Safe Implementation                : No
FSM Style                          : lut
RAM Extraction                     : Yes
RAM Style                          : Auto
ROM Extraction                     : Yes
Mux Style                          : Auto
Decoder Extraction                 : YES
Priority Encoder Extraction        : YES
Shift Register Extraction          : YES
Logical Shifter Extraction         : YES
```

```
XOR Collapsing                      : YES
ROM Style                           : Auto
Mux Extraction                      : YES
Resource Sharing                    : YES
Asynchronous To Synchronous         : NO
Multiplier Style                    : auto
Automatic Register Balancing        : No
---- Target Options
Add IO Buffers                      : YES
Global Maximum Fanout               : 500
Add Generic Clock Buffer(BUFG)      : 24
Register Duplication                : YES
Slice Packing                       : YES
Optimize Instantiated Primitives    : NO
Use Clock Enable                    : Yes
Use Synchronous Set                 : Yes
Use Synchronous Reset               : Yes
Pack IO Registers into IOBs         : auto
Equivalent register Removal         : YES
---- General Options
Optimization Goal                   : Speed
Optimization Effort                 : 1
Library Search Order                : Mips.lso
Keep Hierarchy                      : NO
Netlist Hierarchy                   : as_optimized
RTL Output                          : Yes
Global Optimization                 : AllClockNets
Read Cores                          : YES
Write Timing Constraints            : NO
Cross Clock Analysis                : NO
Hierarchy Separator                 : /
Bus Delimiter                       : <>
Case Specifier                      : maintain
Slice Utilization Ratio             : 100
BRAM Utilization Ratio              : 100
Verilog 2001                        : YES
Auto BRAM Packing                   : NO
Slice Utilization Ratio Delta       : 5
=========================================================================
=========================================================================
*                          HDL Compilation                              *
=========================================================================
Compiling verilog file "Sign_ext_unit.v" in library work
Compiling verilog file "Shift_left_32.v" in library work
Module <Sign_ext_unit> compiled
Compiling verilog file "Shift2rl_unit.v" in library work
Module <Shift_left_32> compiled
Compiling verilog file "Register_unit.v" in library work
Module <Shift2rl_unit> compiled
Compiling verilog file "P_counter.v" in library work
Module <Register_unit> compiled
Compiling verilog file "Mux4_unit.v" in library work
Module <P_counter> compiled
Compiling verilog file "Mux3_unit.v" in library work
Module <Mux4_unit> compiled
Compiling verilog file "Mux2_unit.v" in library work
Module <Mux3_unit> compiled
Compiling verilog file "Mux2SP_unit.v" in library work
Module <Mux2_unit> compiled
Compiling verilog file "memory_unit.v" in library work
Module <Mux2SP_unit> compiled
```

```
Compiling verilog file "Memory_datareg.v" in library work
Module <memory_unit> compiled
Compiling verilog file "Instruction_register_unit.v" in library work
Module <Memory_datareg> compiled
Compiling verilog file "ALU_unit.v" in library work
Module <Instruction_register_unit> compiled
Compiling verilog file "Data_path.v" in library work
Module <ALU_unit> compiled
Compiling verilog file "Control_unit.v" in library work
Module <Datapath> compiled
Compiling verilog file "Mips.v" in library work
Module <Control_unit> compiled
Module <Mips> compiled
No errors in compilation
Analysis of file <"Mips.prj"> succeeded.

=========================================================================
*                     Design Hierarchy Analysis                         *
=========================================================================
Analyzing hierarchy for module <Mips> in library <work>.
Analyzing hierarchy for module <Control_unit> in library <work> with parameters.
        B_EQ = "000100"
        Branch = "1000"
        Complete = "1100"
        Decode = "0001"
        Execution = "0110"
        Fetch = "0000"
        IType = "1010"
        IType_complete = "1011"
        I_type = "001000"
        J_ump = "000010"
        Jump = "1001"
        Lw = "100011"
        Memory_access = "0011"
        Memory_comp = "0010"
        Memory_read = "0100"
        Memory_write = "0101"
        R_type = "000000"
        Reset = "1111"
        Rtype = "0111"
        Shift = "100000"
        Sw = "101011"
Analyzing hierarchy for module <Datapath> in library <work>.
Analyzing hierarchy for module <Mux3_unit> in library <work>.
Analyzing hierarchy for module <P_counter> in library <work>.
Analyzing hierarchy for module <Mux2SP_unit> in library <work>.
Analyzing hierarchy for module <Mux2_unit> in library <work>.
Analyzing hierarchy for module <memory_unit> in library <work>.
Analyzing hierarchy for module <Memory_datareg> in library <work>.
Analyzing hierarchy for module <Instruction_register_unit> in library <work> with
parameters.
        word_size = "00000000000000000000000000100000"
Analyzing hierarchy for module <Register_unit> in library <work> with parameters.
        word_size = "00000000000000000000000000100000"
Analyzing hierarchy for module <Mux4_unit> in library <work>.
Analyzing hierarchy for module <ALU_unit> in library <work> with parameters.
        ADD = "100000"
        AND = "100100"
        Addi = "00"
        NOP = "101100"
        NOR = "100111"
```

```
        OR = "100101"
        R_type = "10"
        SLL = "000000"
        SLT = "101010"
        SRL = "000010"
        SUB = "100010"
        Subi = "01"
        XOR = "100110"
        word_size = "00000000000000000000000000100000"
Analyzing hierarchy for module <Shift2rl_unit> in library <work>.
Analyzing hierarchy for module <Shift_left_32> in library <work>.
Analyzing hierarchy for module <Sign_ext_unit> in library <work>.
=========================================================================
*                            HDL Analysis                               *
=========================================================================
Analyzing top module <Mips>.
Module <Mips> is correct for synthesis.

Analyzing module <Control_unit> in library <work>.
        B_EQ = 6'b000100
        Branch = 4'b1000
        Complete = 4'b1100
        Decode = 4'b0001
        Execution = 4'b0110
        Fetch = 4'b0000
        IType = 4'b1010
        IType_complete = 4'b1011
        I_type = 6'b001000
        J_ump = 6'b000010
        Jump = 4'b1001
        Lw = 6'b100011
        Memory_access = 4'b0011
        Memory_comp = 4'b0010
        Memory_read = 4'b0100
        Memory_write = 4'b0101
        R_type = 6'b000000
        Reset = 4'b1111
        Rtype = 4'b0111
        Shift = 6'b100000
        Sw = 6'b101011
Module <Control_unit> is correct for synthesis.

Analyzing module <Datapath> in library <work>.
Module <Datapath> is correct for synthesis.

Analyzing module <Mux3_unit> in library <work>.
Module <Mux3_unit> is correct for synthesis.

Analyzing module <P_counter> in library <work>.
Module <P_counter> is correct for synthesis.

Analyzing module <Mux2SP_unit> in library <work>.
Module <Mux2SP_unit> is correct for synthesis.

Analyzing module <Mux2_unit> in library <work>.
Module <Mux2_unit> is correct for synthesis.

Analyzing module <memory_unit> in library <work>.
Module <memory_unit> is correct for synthesis.

Analyzing module <Memory_datareg> in library <work>.
```

```
Module <Memory_datareg> is correct for synthesis.

Analyzing module <Instruction_register_unit> in library <work>.
        word_size = 32'sb00000000000000000000000000100000
Module <Instruction_register_unit> is correct for synthesis.

Analyzing module <Register_unit> in library <work>.
        word_size = 32'sb00000000000000000000000000100000
Module <Register_unit> is correct for synthesis.

Analyzing module <Mux4_unit> in library <work>.
Module <Mux4_unit> is correct for synthesis.

Analyzing module <ALU_unit> in library <work>.
        ADD = 6'b100000
        AND = 6'b100100
        Addi = 2'b00
        NOP = 6'b101100
        NOR = 6'b100111
        OR = 6'b100101
        R_type = 2'b10
        SLL = 6'b000000
        SLT = 6'b101010
        SRL = 6'b000010
        SUB = 6'b100010
        Subi = 2'b01
        XOR = 6'b100110
        word_size = 32'sb00000000000000000000000000100000
Module <ALU_unit> is correct for synthesis.

Analyzing module <Shift2rl_unit> in library <work>.
Module <Shift2rl_unit> is correct for synthesis.

Analyzing module <Shift_left_32> in library <work>.
Module <Shift_left_32> is correct for synthesis.

Analyzing module <Sign_ext_unit> in library <work>.
Module <Sign_ext_unit> is correct for synthesis.

=========================================================================
*                           HDL Synthesis                               *
=========================================================================
Performing bidirectional port resolution...
Synthesizing Unit <Control_unit>.
    Related source file is "Control_unit.v".
    Found finite state machine <FSM_0> for signal <current_state>.
    -----------------------------------------------------------------------
    | States             | 14                                           |
    | Transitions        | 22                                           |
    | Inputs             | 7                                            |
    | Outputs            | 16                                           |
    | Clock              | clk                       (rising_edge)      |
    | Reset              | rst                       (negative)         |
    | Reset type         | asynchronous                                 |
    | Reset State        | 1111                                         |
    | Encoding           | automatic                                    |
    | Implementation     | LUT                                          |
    -----------------------------------------------------------------------
    Summary:
        inferred   1 Finite State Machine(s).
Unit <Control_unit> synthesized.
```

```
Synthesizing Unit <Mux3_unit>.
    Related source file is "Mux3_unit.v".
    Found 32-bit 4-to-1 multiplexer for signal <Out_mux>.
    Summary:
        inferred  32 Multiplexer(s).
Unit <Mux3_unit> synthesized.
Synthesizing Unit <P_counter>.
    Related source file is "P_counter.v".
    Found 32-bit register for signal <pc_address>.
    Summary:
        inferred  32 D-type flip-flop(s).
Unit <P_counter> synthesized.
Synthesizing Unit <Mux2SP_unit>.
    Related source file is "Mux2SP_unit.v".
Unit <Mux2SP_unit> synthesized.
Synthesizing Unit <Mux2_unit>.
    Related source file is "Mux2_unit.v".
Unit <Mux2_unit> synthesized.
Synthesizing Unit <memory_unit>.
    Related source file is "memory_unit.v".
    Found 32-bit register for signal <MEMData>.
    Found 1024-bit register for signal <memory_reg>.
INFO:Xst:738 - HDL ADVISOR - 1024 flip-flops were inferred for signal <memory_reg>.
You may be trying to describe a RAM in a way that is incompatible with block and
distributed RAM resources available on Xilinx devices, or with a specific template
that is not supported. Please review the Xilinx resources documentation and the XST
user manual for coding guidelines. Taking advantage of RAM resources will lead to
improved device usage and reduced synthesis time.
    Summary:
        inferred 576 D-type flip-flop(s).
        inferred  32 Multiplexer(s).
Unit <memory_unit> synthesized.
Synthesizing Unit <Memory_datareg>.
    Related source file is "Memory_datareg.v".
Unit <Memory_datareg> synthesized.
Synthesizing Unit <Instruction_register_unit>.
    Related source file is "Instruction_register_unit.v".
    Found 16-bit register for signal <Instruction_addr>.
    Found 6-bit register for signal <Instruction_op>.
    Found 5-bit register for signal <Instruction_rd>.
    Found 5-bit register for signal <Instruction_rs>.
    Found 5-bit register for signal <Instruction_rt>.
    Summary:
        inferred  37 D-type flip-flop(s).
Unit <Instruction_register_unit> synthesized.
Synthesizing Unit <Register_unit>.
    Related source file is "Register_unit.v".
    Found 32-bit register for signal <Reg_A>.
    Found 32-bit register for signal <Reg_B>.
    Found 1024-bit register for signal <registers>.
INFO:Xst:738 - HDL ADVISOR - 1024 flip-flops were inferred for signal <registers>. You
may be trying to describe a RAM in a way that is incompatible with block and
distributed RAM resources available on Xilinx devices, or with a specific template
that is not supported. Please review the Xilinx resources documentation and the XST
user manual for coding guidelines. Taking advantage of RAM resources will lead to
improved device usage and reduced synthesis time.
    Summary:
        inferred 1088 D-type flip-flop(s).
        inferred  64 Multiplexer(s).
Unit <Register_unit> synthesized.
Synthesizing Unit <Mux4_unit>.
```

```
    Related source file is "Mux4_unit.v".
    Found 32-bit 4-to-1 multiplexer for signal <Out_mux>.
    Summary:
        inferred  32 Multiplexer(s).
Unit <Mux4_unit> synthesized.
Synthesizing Unit <ALU_unit>.
    Related source file is "ALU_unit.v".
    Found 32-bit 3-to-1 multiplexer for signal <ALU_result>.
    Found 32-bit register for signal <ALUOUT>.
    Found 32-bit addsub for signal <ALU_result$share0000> created at line 36.
    Found 32-bit shifter logical right for signal <ALU_result$shift0000> created at
line 56.
    Found 32-bit shifter logical left for signal <ALU_result$shift0001> created at
line 58.
    Found 32-bit xor2 for signal <ALU_result$xor0000> created at line 50.
    Summary:
        inferred  32 D-type flip-flop(s).
        inferred   1 Adder/Subtractor(s).
        inferred  32 Multiplexer(s).
        inferred   2 Combinational logic shifter(s).
Unit <ALU_unit> synthesized.
Synthesizing Unit <Shift2rl_unit>.
    Related source file is "Shift2rl_unit.v".
Unit <Shift2rl_unit> synthesized.
Synthesizing Unit <Shift_left_32>.
    Related source file is "Shift_left_32.v".
Unit <Shift_left_32> synthesized.
Synthesizing Unit <Sign_ext_unit>.
    Related source file is "Sign_ext_unit.v".
Unit <Sign_ext_unit> synthesized.
Synthesizing Unit <Datapath>.
    Related source file is "Data_path.v".
Unit <Datapath> synthesized.
Synthesizing Unit <Mips>.
    Related source file is "Mips.v".
Unit <Mips> synthesized.
INFO:Xst:1767 - HDL ADVISOR - Resource sharing has identified that some arithmetic
operations in this design can share the same physical resources for reduced device
utilization. For improved clock frequency you may try to disable resource sharing.
=========================================================================
HDL Synthesis Report
Macro Statistics
# Adders/Subtractors                                   : 1
 32-bit addsub                                        : 1
# Registers                                            : 74
 16-bit register                                      : 1
 32-bit register                                      : 69
 5-bit register                                       : 3
 6-bit register                                       : 1
# Multiplexers                                         : 6
 32-bit 3-to-1 multiplexer                            : 1
 32-bit 32-to-1 multiplexer                           : 3
 32-bit 4-to-1 multiplexer                            : 2
# Logic shifters                                       : 2
 32-bit shifter logical left                          : 1
 32-bit shifter logical right                         : 1
# Xors                                                 : 1
 32-bit xor2                                          : 1
=========================================================================
=========================================================================
*                       Advanced HDL Synthesis                          *
```

```
=========================================================================
Analyzing FSM <FSM_0> for best encoding.
Optimizing FSM <TOP_ControlPath/current_state/FSM> on signal <current_state[1:14]>
with one-hot encoding.
-------------------------
 State | Encoding
-------------------------
 0000  | 00000000000010
 0001  | 00000000000100
 0010  | 00000000001000
 0011  | 00000100000000
 0100  | 00010000000000
 0101  | 00001000000000
 0110  | 00000000010000
 0111  | 01000000000000
 1000  | 00000001000000
 1001  | 00000010000000
 1010  | 00000000100000
 1011  | 10000000000000
 1100  | 00100000000000
 1111  | 00000000000001
-------------------------
=========================================================================
Advanced HDL Synthesis Report
Macro Statistics
# FSMs                                          : 1
# Adders/Subtractors                            : 1
 32-bit addsub                                  : 1
# Registers                                     : 2245
 Flip-Flops                                     : 2245
# Multiplexers                                  : 68
 1-bit 32-to-1 multiplexer                      : 64
 32-bit 3-to-1 multiplexer                      : 1
 32-bit 32-to-1 multiplexer                     : 1
 32-bit 4-to-1 multiplexer                      : 2
# Logic shifters                                : 2
 32-bit shifter logical left                    : 1
 32-bit shifter logical right                   : 1
# Xors                                          : 1
 32-bit xor2                                    : 1
=========================================================================
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block Mips, actual ratio is 0.
Final Macro Processing ...
=========================================================================
Final Register Report
Found no macro
=========================================================================
=========================================================================
*                           Partition Report                            *
=========================================================================
Partition Implementation Status
-------------------------------
  No Partitions were found in this design.
-------------------------------
=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : Mips.ngr
Top Level Output File Name         : Mips
Output Format                      : NGC
```

```
Optimization Goal                  : Speed
Keep Hierarchy                     : NO
Design Statistics
# IOs                              : 2
Cell Usage :
=========================================================================
Device utilization summary:
---------------------------
Selected Device : 3s500efg320-5
 Number of Slices:                        0  out of   4656     0%
 Number of IOs:                           2
 Number of bonded IOBs:                   0  out of    232     0%
---------------------------
Partition Resource Summary:
---------------------------
  No Partitions were found in this design.
---------------------------
=========================================================================
TIMING REPORT
NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.
Clock Information:
------------------
No clock signals found in this design
Asynchronous Control Signals Information:
----------------------------------------
No asynchronous control signals found in this design
Timing Summary:
---------------
Speed Grade: -5
   Minimum period: No path found
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: No path found
   Maximum combinational path delay: No path found
Timing Detail:
--------------
All values displayed in nanoseconds (ns)
=========================================================================
Total REAL time to Xst completion: 30.00 secs
Total CPU time to Xst completion: 29.59 secs

-->
Total memory usage is 165212 kilobytes
Number of errors   :    0 (   0 filtered)
Number of warnings : 2261 (   0 filtered)
Number of infos    :    3 (   0 filtered)
```

ITESO

Universidad Jesuita
de Guadalajara

Proyecto Final Parte 2:
Programación de una Calculadora hexadecimal con el procesador MIPS 32Bits Multi-ciclo

Alumno:
ENRIQUE ISRAEL HERNANDEZ MEZA

El cpu ejecuta el siguiente programa para realizar la operación de suma y resta de dos dígitos hexadecimales regresando un digito hexadecimal

```
LW r1 <= mem[10'h] = ff'h;
LW r2 <= mem[11'h] = 2b;
LW r3 <= mem[12'h] = 2d;
BEQ= (r1-r30) pc+=02; r1 = 7'h
ADD r0 + r30 -> r4;
BEQ= (r2-r30) pc+=04; r1 = 7'h ;
BEQ= (r3-r30) pc+=02; r1 = 7'h
jump  addr5'h;
jump  addr3'h;
jump  addr'h;
BEQ= (r2-r30) pc+=02; r1 = 7'h
ADD r4 + r30 -> r31;
ADD r0 + r30 -> r1;
jump  addr 3'h;
jump  addr A'h;
BEQ= (r3-r30) pc+=02; r1 = 7'h
SUB r4 - r30 -> r31; = 0 + r30;
ADD r0 + r30 -> r1;
jump  addr3'h
jump  addr15'h
```

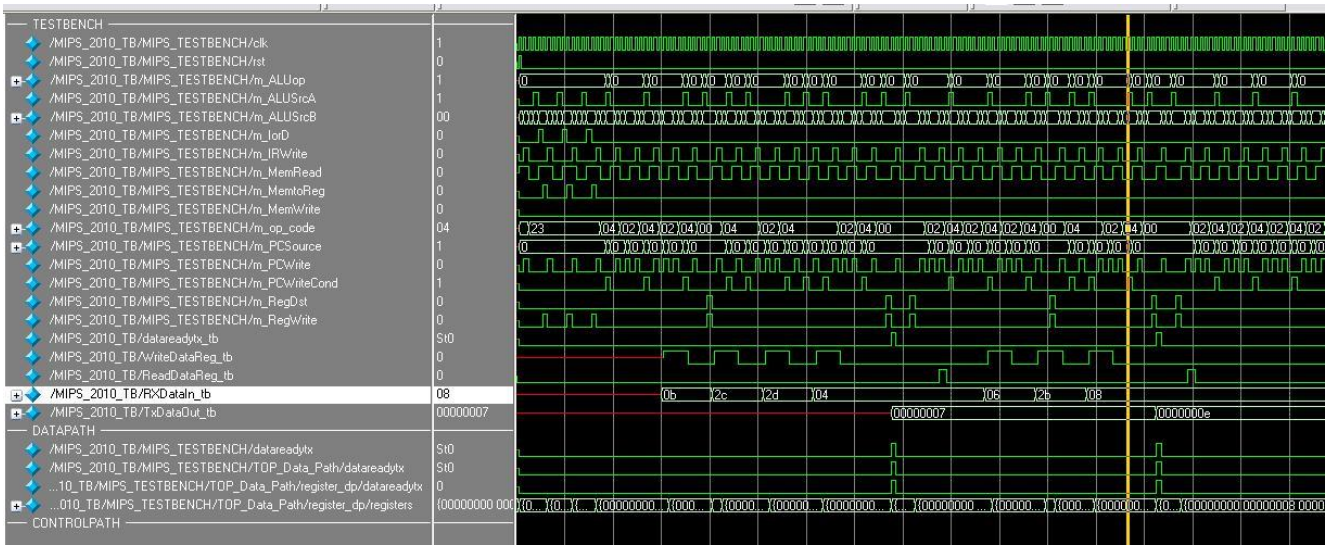
En la memoria se encuentra instalado el siguiente programa

/// MEMORIA DE PROGRAMA

```
  memory_reg[0]  <= 32'h8c01_0014; // LW r1 <= mem[10'h] = ff'h;
 memory_reg[1]  <= 32'h8c02_0015; // LW r2 <= mem[11'h] = 2b;
 memory_reg[2]  <= 32'h8c03_0016; // LW r3 <= mem[12'h] = 2d;
 memory_reg[3]  <= 32'h103e_0004; // BEQ= (r1-r30) pc+=02; r1 = 7'h
 memory_reg[4]  <= 32'h001e_2020; // ADD r0 + r30 -> r4;
 memory_reg[5]  <= 32'h105e_0004; // BEQ= (r2-r30) pc+=04; r1 = 7'h ;
 memory_reg[6]  <= 32'h107e_0008; // BEQ= (r3-r30) pc+=02; r1 = 7'h
 memory_reg[7]  <= 32'h0800_0005; // jump  addr5'h;
 memory_reg[8]  <= 32'h0800_0003; // jump  addr3'h;
 memory_reg[9]  <= 32'h0000_0000; // jump  addr'h;
 memory_reg[10] <= 32'h105e_0003; // BEQ= (r2-r30) pc+=02; r1 = 7'h
 memory_reg[11] <= 32'h009e_f820; // ADD r4 + r30 -> r31;
 memory_reg[12] <= 32'h001e_0820; // ADD r0 + r30 -> r1;
 memory_reg[13] <= 32'h0800_0003; // jump  addr 3'h;
 memory_reg[14] <= 32'h0800_000a; // jump  addr A'h;
 memory_reg[15] <= 32'h107e_0003; // BEQ= (r3-r30) pc+=02; r1 = 7'h
 memory_reg[16] <= 32'h009e_f822; // SUB r4 - r30 -> r31; = 0 + r30;
 memory_reg[17] <= 32'h001e_0820; // ADD r0 + r30 -> r1;
 memory_reg[18] <= 32'h0800_0003; // jump  addr3'h
 memory_reg[19] <= 32'h0800_000f; // jump  addr15'h
       // MEMORIA DE DATOS
 memory_reg[20] <= 32'h0000_00ff;  // DATO FUENTE DEL REGISTRO R1
 memory_reg[21] <= 32'h0000_002b;  // DATO FUENTE DEL REGISTRO R2
 memory_reg[22] <= 32'h0000_002d;  // DATO DESTINO DEL REGISTRO R3
memory_reg[23] <= 32'h0000_0000;
 memory_reg[24] <= 32'h0000_0000;
 memory_reg[25] <= 32'h0000_0000;
memory_reg[26] <= 32'h0000_0000;
 memory_reg[27] <= 32'h0000_0000;
 memory_reg[28] <= 32'h0000_0000;
 memory_reg[29] <= 32'h0000_0000;
 memory_reg[30] <= 32'h0000_0000; // Puerto de salida
 memory_reg[31] <= 32'h0000_0000; // Puerto de Entrada
```

Resultados en la Simulación

Forma de Onda



En la señal seleccionada RXDataIn_tb se le proporciona las siguientes operaciones

b – 4 = 7 , 6 + 8 = e,
en la señal DataOut_tb se observa el resultado obtenido.

B. DISEÑO DE UN MICROPROCESADOR SUPERESCALAR

ITESO

Universidad Jesuita
de Guadalajara

Proyecto Final:
Diseño de Microprocesadores

Alumno:
ENRIQUE ISRAEL HERNANDEZ MEZA
MARIO ANTONIO RUVALCABA PEREZ

Diseño de Microprocesadores Primavera 2010 Proyecto Final German Fabila García

OBJETIVO.

En este proyecto el alumno implementara el conjunto de instrucciones MIPS en una arquitectura superescalar con ejecución fuera de orden.
El diseño de cada uno de los bloques que comprenden la arquitectura será discutida en clase y se generara un documento de especificación de micro-arquitectura por cada bloque.
El funcionamiento de la implementación será verificada tanto en simulación como en la tarjeta de desarrollo para FPGA.

Bloques Principales:

- I-Fetch: I-*Cache* e IFQ.
- Dispatch: comprende la unidad de dispatch, el RST, un TAG FIFO y el banco de registros.
- Issue Queues: la misma para las unidades de ejecución de enteros, divisiones y multiplicaciones.
- Load/Store Queue.
- Unidades de Ejecución: enteros, D-*Cache*, divisor y multiplicador
- Issue Unit: comprende la unidad de issue y el manejo del CDB.

PROCEDIMIENTO.

El diseño de cada bloque será discutido en la sesión de clase, una vez que el documento de especificación de microarquitectura (MAS) sea publicado en la plataforma moodle el alumno es responsable de la implementación y verificación de dicho bloque.

Junto con la publicación del MAS se anunciara la fecha límite para subir la implementación.
Solo se revisaran las entregas parciales en caso que el proyecto no funcione en el simulador.

Por cada bloque funcional el alumno tiene que subir a la plataforma moodle un archivo comprimido con la siguiente estructura.

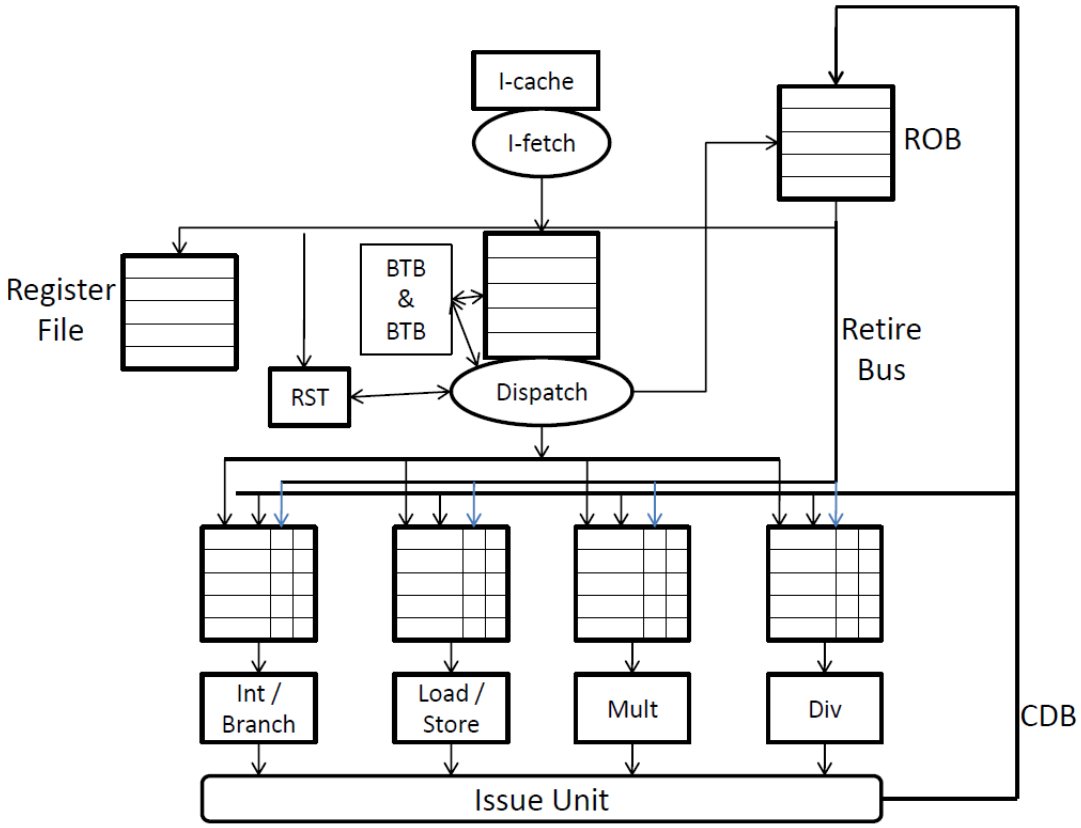
Carpeta	**Contenido**
Fuentes	Todos los archivos fuentes, sintetizables
Cama de pruebas	El archivo top de la cama de pruebas, y un archivo README con instrucciones de uso.
Casos de prueba	Los diferentes casos de pruebas

EVALUACION.

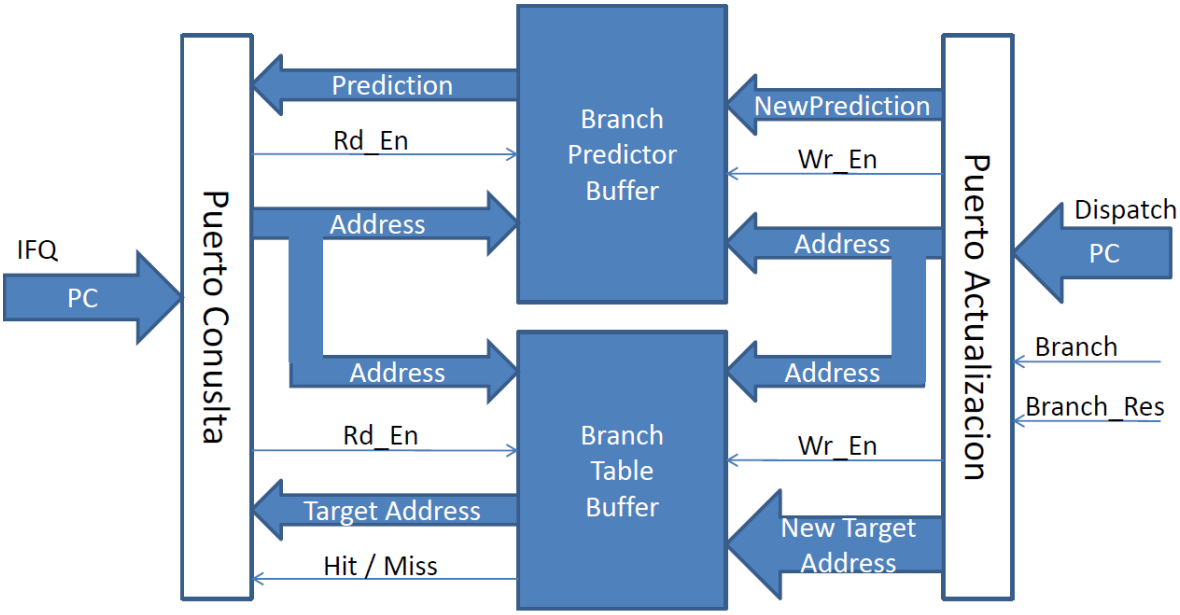
- Si funciona tanto en la FPGA como en el simulador 100%
- Si solo funciona en el simulador 80%
- En caso que no funcione en el simulador se revisaran los bloques subidos a la plataforma, cada bloque tiene un valor del 10%. Solo se revisaran los bloques que hayan sido subidos dentro de la fecha límite.

Puntos Extras: El proyecto contempla la implementación de la arquitectura superescalar más básica. Si el alumno decide añadir a su diseño los bloques de optimización y mejora que serán discutidos en clase recibirá puntos extras. ¡Un diseño SUPER completo puede llegar a recibir una calificación del 200%, o sea el 100% de la calificación del curso!!!!

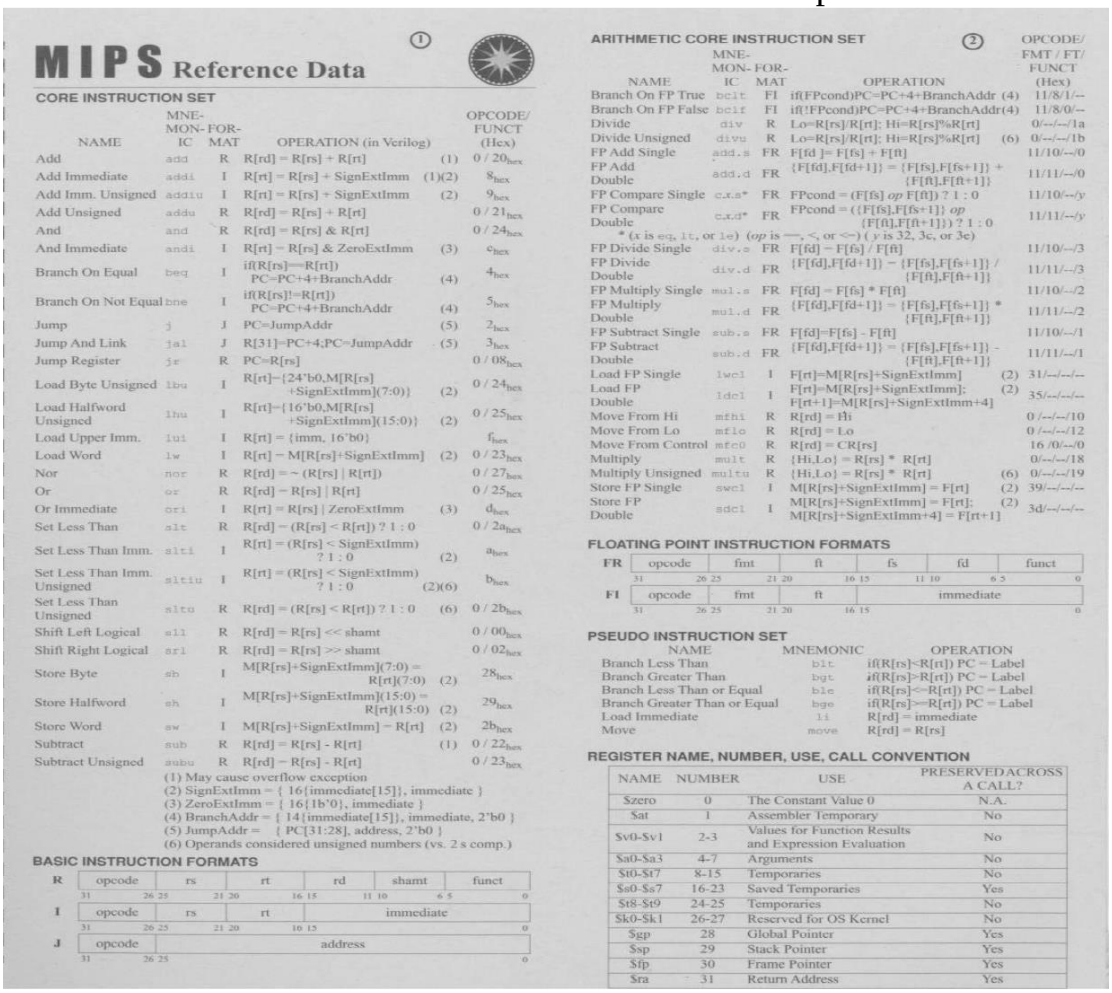
1.1. Arquitectura Propuesta



1.2. Arquitectura del Branch Predictor & Table Buffer



1.3. Set de instrucciones del MIPSR4000 a implementar



MIPS Reference Data ①

CORE INSTRUCTION SET

NAME	MNE-MON-IC	FOR-MAT	OPERATION (in Verilog)		OPCODE/ FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	0 / 20_{hex}
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1)(2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21_{hex}
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24_{hex}
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+4;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		0 / 08_{hex}
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs]+SignExtImm](7:0)}	(2)	0 / 24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)}	(2)	0 / 25_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	0 / 23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		0 / 27_{hex}
Or	or	R	R[rd] = R[rs] | R[rt]		0 / 25_{hex}
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 / $2a_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2)(6)	b_{hex}
Set Less Than Unsigned	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	0 / $2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rs] << shamt		0 / 00_{hex}
Shift Right Logical	srl	R	R[rd] = R[rs] >> shamt		0 / 02_{hex}
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	0 / 22_{hex}
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		0 / 23_{hex}

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1'b0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)

BASIC INSTRUCTION FORMATS

	31–26	25–21	20–16	15–11	10–6	5–0
R	opcode	rs	rt	rd	shamt	funct
I	opcode	rs	rt	immediate		
J	opcode	address				

ARITHMETIC CORE INSTRUCTION SET ②

NAME	MNE-MON-IC	FOR-MAT	OPERATION		OPCODE/ FMT / FT/ FUNCT (Hex)
Branch On FP True	bc1t	FI	if(FPcond)PC=PC+4+BranchAddr	(4)	11/8/1/--
Branch On FP False	bc1f	FI	if(!FPcond)PC=PC+4+BranchAddr	(4)	11/8/0/--
Divide	div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]		0/--/--/1a
Divide Unsigned	divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6)	0/--/--/1b
FP Add Single	add.s	FR	F[fd]= F[fs] + F[ft]		11/10/--/0
FP Add Double	add.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}		11/11/--/0
FP Compare Single	c.x.s*	FR	FPcond = (F[fs] *op* F[ft]) ? 1 : 0		11/10/--/*y*
FP Compare Double	c.x.d*	FR	FPcond = ({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]}) ? 1 : 0		11/11/--/*y*

* (*x* is eq, lt, or le) (*op* is ==, <, or <=) (*y* is 32, 3c, or 3e)

FP Divide Single	div.s	FR	F[fd] = F[fs] / F[ft]		11/10/--/3
FP Divide Double	div.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}		11/11/--/3
FP Multiply Single	mul.s	FR	F[fd] = F[fs] * F[ft]		11/10/--/2
FP Multiply Double	mul.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}		11/11/--/2
FP Subtract Single	sub.s	FR	F[fd]=F[fs] - F[ft]		11/10/--/1
FP Subtract Double	sub.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}		11/11/--/1
Load FP Single	lwc1	I	F[rt]=M[R[rs]+SignExtImm]	(2)	31/--/--/--
Load FP Double	ldc1	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2)	35/--/--/--
Move From Hi	mfhi	R	R[rd] = Hi		0 /--/--/10
Move From Lo	mflo	R	R[rd] = Lo		0 /--/--/12
Move From Control	mfc0	R	R[rd] = CR[rs]		16 /0/--/0
Multiply	mult	R	{Hi,Lo} = R[rs] * R[rt]		0/--/--/18
Multiply Unsigned	multu	R	{Hi,Lo} = R[rs] * R[rt]	(6)	0/--/--/19
Store FP Single	swc1	I	M[R[rs]+SignExtImm] = F[rt]	(2)	39/--/--/--
Store FP Double	sdc1	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2)	3d/--/--/--

FLOATING POINT INSTRUCTION FORMATS

	31–26	25–21	20–16	15–11	10–6	5–0
FR	opcode	fmt	ft	fs	fd	funct
FI	opcode	fmt	ft	immediate		

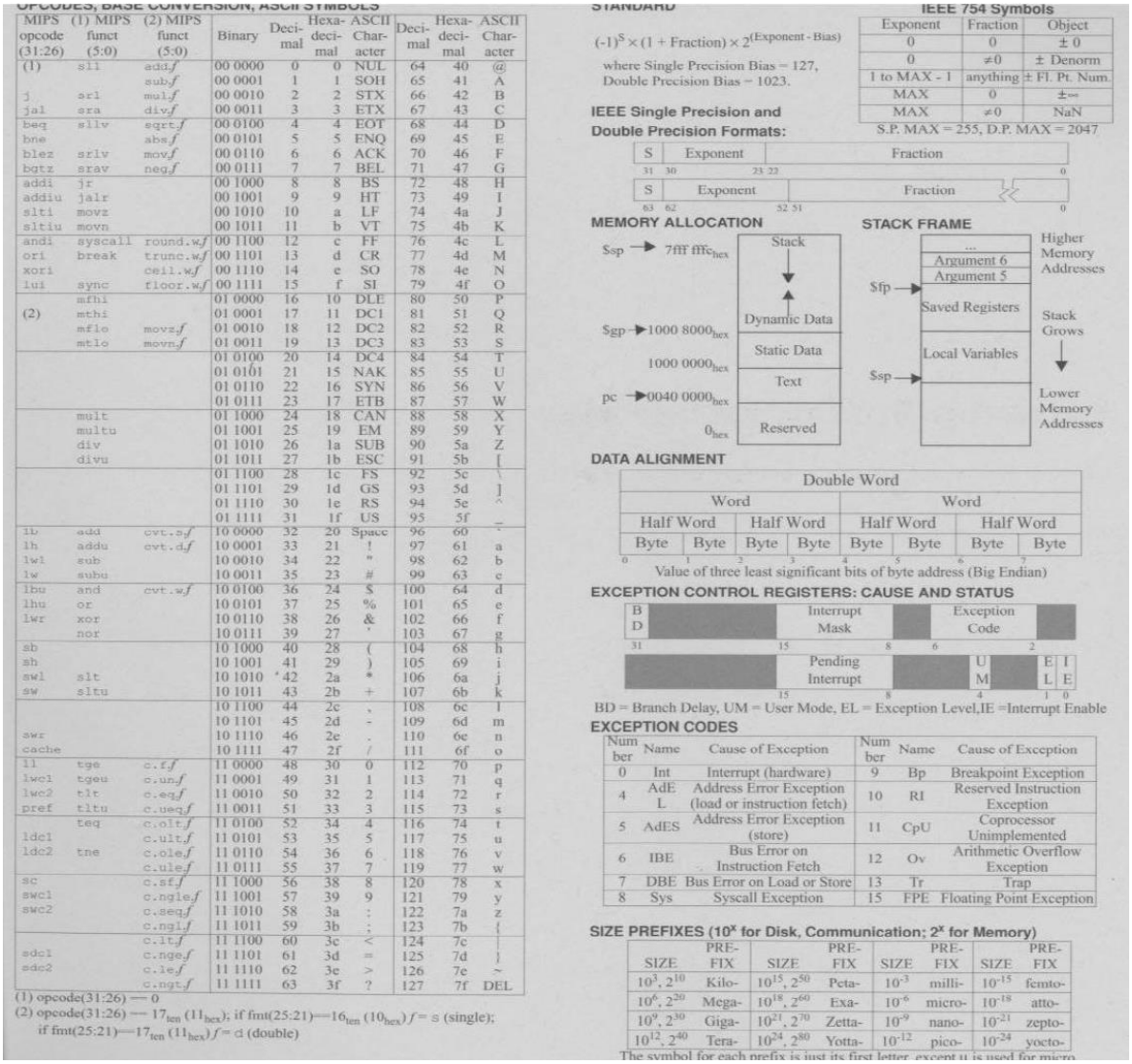
PSEUDO INSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

Figura 2 Instruction Set Architecture R4000



OPCODES, BASE CONVERSION, ASCII SYMBOLS

MIPS opcode (31:26)	(1) MIPS funct (5:0)	(2) MIPS funct (5:0)	Binary	Decimal	Hexadecimal	ASCII Character	Decimal	Hexadecimal	ASCII Character
(1)	sll	add.f	00 0000	0	0	NUL	64	40	@
		sub.f	00 0001	1	1	SOH	65	41	A
j	srl	mul.f	00 0010	2	2	STX	66	42	B
jal	sra	div.f	00 0011	3	3	ETX	67	43	C
beq	sllv	sqrt.f	00 0100	4	4	EOT	68	44	D
bne		abs.f	00 0101	5	5	ENQ	69	45	E
blez	srlv	mov.f	00 0110	6	6	ACK	70	46	F
bgtz	srav	neg.f	00 0111	7	7	BEL	71	47	G
addi	jr		00 1000	8	8	BS	72	48	H
addiu	jalr		00 1001	9	9	HT	73	49	I
slti	movz		00 1010	10	a	LF	74	4a	J
sltiu	movn		00 1011	11	b	VT	75	4b	K
andi	syscall	round.w.f	00 1100	12	c	FF	76	4c	L
ori	break	trunc.w.f	00 1101	13	d	CR	77	4d	M
xori		ceil.w.f	00 1110	14	e	SO	78	4e	N
lui	sync	floor.w.f	00 1111	15	f	SI	79	4f	O
	mfhi		01 0000	16	10	DLE	80	50	P
(2)	mthi		01 0001	17	11	DC1	81	51	Q
	mflo	movz.f	01 0010	18	12	DC2	82	52	R
	mtlo	movn.f	01 0011	19	13	DC3	83	53	S
			01 0100	20	14	DC4	84	54	T
			01 0101	21	15	NAK	85	55	U
			01 0110	22	16	SYN	86	56	V
			01 0111	23	17	ETB	87	57	W
	mult		01 1000	24	18	CAN	88	58	X
	multu		01 1001	25	19	EM	89	59	Y
	div		01 1010	26	1a	SUB	90	5a	Z
	divu		01 1011	27	1b	ESC	91	5b	[
			01 1100	28	1c	FS	92	5c	\
			01 1101	29	1d	GS	93	5d	]
			01 1110	30	1e	RS	94	5e	^
			01 1111	31	1f	US	95	5f	_
lb	add	cvt.s.f	10 0000	32	20	Space	96	60	`
lh	addu	cvt.d.f	10 0001	33	21	!	97	61	a
lwl	sub		10 0010	34	22	"	98	62	b
lw	subu		10 0011	35	23	#	99	63	c
lbu	and	cvt.w.f	10 0100	36	24	$	100	64	d
lhu	or		10 0101	37	25	%	101	65	e
lwr	xor		10 0110	38	26	&	102	66	f
	nor		10 0111	39	27	'	103	67	g
sb			10 1000	40	28	(	104	68	h
sh			10 1001	41	29	)	105	69	i
swl	slt		10 1010	42	2a	*	106	6a	j
sw	sltu		10 1011	43	2b	+	107	6b	k
			10 1100	44	2c	,	108	6c	l
			10 1101	45	2d	-	109	6d	m
swr			10 1110	46	2e	.	110	6e	n
cache			10 1111	47	2f	/	111	6f	o
ll	tge	c.f.f	11 0000	48	30	0	112	70	p
lwc1	tgeu	c.un.f	11 0001	49	31	1	113	71	q
lwc2	tlt	c.eq.f	11 0010	50	32	2	114	72	r
pref	tltu	c.ueq.f	11 0011	51	33	3	115	73	s
	teq	c.olt.f	11 0100	52	34	4	116	74	t
ldc1		c.ult.f	11 0101	53	35	5	117	75	u
ldc2	tne	c.ole.f	11 0110	54	36	6	118	76	v
		c.ule.f	11 0111	55	37	7	119	77	w
sc		c.sf.f	11 1000	56	38	8	120	78	x
swc1		c.ngle.f	11 1001	57	39	9	121	79	y
swc2		c.seq.f	11 1010	58	3a	:	122	7a	z
		c.ngl.f	11 1011	59	3b	;	123	7b	{
		c.lt.f	11 1100	60	3c	<	124	7c	|
sdc1		c.nge.f	11 1101	61	3d	=	125	7d	}
sdc2		c.le.f	11 1110	62	3e	>	126	7e	~
		c.ngt.f	11 1111	63	3f	?	127	7f	DEL

(1) opcode(31:26) — 0

(2) opcode(31:26) — 17_{ten} (11_{hex}); if fmt(25:21)—16_{ten} (10_{hex}) f = s (single); if fmt(25:21)—17_{ten} (11_{hex}) f = d (double)

STANDARD

$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$

where Single Precision Bias = 127, Double Precision Bias = 1023.

IEEE 754 Symbols

Exponent	Fraction	Object
0	0	± 0
0	≠0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	±∞
MAX	≠0	NaN

S.P. MAX = 255, D.P. MAX = 2047

IEEE Single Precision and Double Precision Formats:

MEMORY ALLOCATION

STACK FRAME

DATA ALIGNMENT

Value of three least significant bits of byte address (Big Endian)

EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS

BD = Branch Delay, UM = User Mode, EL = Exception Level, IE = Interrupt Enable

EXCEPTION CODES

Number	Name	Cause of Exception	Number	Name	Cause of Exception
0	Int	Interrupt (hardware)	9	Bp	Breakpoint Exception
4	AdEL	Address Error Exception (load or instruction fetch)	10	RI	Reserved Instruction Exception
5	AdES	Address Error Exception (store)	11	CpU	Coprocessor Unimplemented
6	IBE	Bus Error on Instruction Fetch	12	Ov	Arithmetic Overflow Exception
7	DBE	Bus Error on Load or Store	13	Tr	Trap
8	Sys	Syscall Exception	15	FPE	Floating Point Exception

SIZE PREFIXES (10^x for Disk, Communication; 2^x for Memory)

SIZE	PREFIX	SIZE	PREFIX	SIZE	PREFIX	SIZE	PREFIX
10^3, 2^{10}	Kilo-	10^{15}, 2^{50}	Peta-	10^{-3}	milli-	10^{-15}	femto-
10^6, 2^{20}	Mega-	10^{18}, 2^{60}	Exa-	10^{-6}	micro-	10^{-18}	atto-
10^9, 2^{30}	Giga-	10^{21}, 2^{70}	Zetta-	10^{-9}	nano-	10^{-21}	zepto-
10^{12}, 2^{40}	Tera-	10^{24}, 2^{80}	Yotta-	10^{-12}	pico-	10^{-24}	yocto-

The symbol for each prefix is just its first letter, except u is used for micro.

Figura 3 Opcodes and Conversions

1.4. Listado de unidades Implementadas del proyecto

Bloque	Modulo	Avance	Diseñador
Traductor MIPS	Traductor MIPS	Complete	Mario Antonio Rubalcaba Pérez
Modelo de Referencia Funcional	Modelo de Referencia Funcional	Complete	Mario Antonio Rubalcaba Pérez
ICACHE	IFQ	Complete	Mario Antonio Rubalcaba Pérez
IFQ	IFQ	Complete	Enrique Israel Hernández Meza
RegFile	Dispatch	Complete	Enrique Israel Hernández Meza
RST	Dispatch	Complete	Enrique Israel Hernández Meza
Tag FIFO	Dispatch	Complete	Enrique Israel Hernández Meza
Decoder	Dispatch	Complete	Mario Antonio Rubalcaba Pérez
Branch Stall Logic	Dispatch	Complete	Mario Antonio Rubalcaba Pérez
Branch & Jmp Address Calculation	Dispatch	Complete	Mario Antonio Rubalcaba Pérez
Jmp Execution	Dispatch	Complete	Mario Antonio Rubalcaba Pérez
Integer Issue Queue	Integer Issue Queue	Complete	Enrique Israel Hernández Meza
Multiplication Issue Queue	Multiplication Issue Queue	Complete	Enrique Israel Hernández Meza
Divide Issue Queue	Divide Issue Queue	Complete	Enrique Israel Hernández Meza
Load/Store Issue Queue	Load/Store Issue Queue	Complete	Enrique Israel Hernández Meza
Ejecution Units	Ejecution Units	Complete	Enrique Israel Hernández Meza

1.5. Listado Implementación de módulos en RTL

Modulo	IFQ	Dispatch Unit	ICACHE	Issue	Queue
	IFQ.v	Disp_Branch.v	ICACHE.v	Issue_ALU.v	queue_div.v
	IFQ_Bypass.v	Disp_Comp7.v		Issue_Divider.v	queue_int.v
	IFQ_MuxA.v	Disp_Decoder.v		Issue_Logic.v	queue_ldst.v
	IFQ_Queue.v	Disp_EncWen0.v		Issue_LS_CACHED.v	queue_mul.v
		Disp_EnWen1rst.v		Issue_Multiplier.v	
		Disp_MuxReg.v		Issue_Unit.v	
		Disp_RegFile.v		Division.v	
		Disp_RST.v		Division16.v	
		Disp_TagFIFO.v			
		Dispatcher.v			

TestBench	TopLevel
Test.v	TopLevel.v
	SuperScalar.v
	SuperScalar_tb.v

1.6. Programas de Prueba.

```
-- BUBBLE SORT & SELECTION SORT
--
-- Preconditions on Register file
--   Registers set to their register number
--   ex) $0 = 0, $1 = 1, $2 = 2 ..... $31 = 31
-- Preconditions on Data Memory
--   None. Any data in the first 5 locations will be sorted by Bubble sort.
--   The next 5 data will be sorted by Selection sort.
--
--
--000 00000020 add $0, $0, $0     -- nop *** INITIALIZATION FOR BUBBLE SORT
***
--004 0080F820 add $31, $4, $0    -- $31 = 4
--008 00BF1019 mul $2, $5, $31    -- ak = 4 * num_of_items
--00c 00000020 add $0, $0, $0     -- noop
--
--010 00001820 add $3, $0, $0     -- ai = 0 *** BUBBLE SORT STARTS ***
--014 007F2020 add $4, $3, $31    -- aj = ai + 4
--018 0082302A slt $6, $4, $2     -- (aj < ak) ?
--01c 10C0000C beq $6, $0, 12     -- if no, program finishes. goto chcker
--
--020 8C6D0000 lw  $13, 0($3)     -- mi = M(ai)       (LABEL: LOAD)
--024 8C8E0000 lw  $14, 0($4)     -- mj = M(aj)
--028 01CD302A slt $6, $14, $13   -- (mj < mi) ?
--02c 10C00002 beq $6, $0, 2      -- if no, skip swap
--
--030 AC6E0000 sw  $14, 0($3)     -- M(ai) = mj // swap
--034 AC8D0000 sw  $13, 0($4)     -- M(aj) = mi // swap
--038 007F1820 add $3, $3, $31    -- ai = ai + 4      (LABEL: SKIP SWAP)
--03c 009F2020 add $4, $4, $31    -- aj = aj + 4
--
--
--040 0082302A slt $6, $4, $2     -- (aj < ak) ?
--044 10C1FFF6 beq $6, $1, -10    -- if yes, goto LOAD
--048 005F1022 sub $2, $2, $31    -- ak = ak - 4
--04c 08000004 jmp 4              -- goto BEGIN
--
--050 00000020 add $0,  $0,  $0    -- nop *** CHECKER FOR FIRST 5 ITEMS ***
--054 0000D020 add $26, $0,  $0    -- addr1 = 0
--058 035FD820 add $27, $26, $31   -- addr2 = addr1 + 4
--05c 00BFE019 mul $28, $5, $31    -- addr3 = num_of_items * 4
--
--060 039AE020 add $28, $28, $26   -- addr3 = addr3 + addr1
--064 8F5D0000 lw  $29, 0 ($26)    -- maddr1 = M(addr1)
--068 8F7E0000 lw  $30, 0 ($27)    -- maddr2 = M(addr2)
--06c 03DDC82A slt $25, $30, $29   -- (maddr2 < maddr1) ?
--
--070 13200001 beq $25, $0,  1     -- if no, proceed to the next data
--074 1000FFFF beq $0,  $0, -1     -- else, You're stuck here
--078 035FD020 add $26, $26, $31   -- addr1 = addr1 + 4
--07c 037FD820 add $27, $27, $31   -- addr2 = addr2 + 4
--
```

```
--
--080 137C0001 beq $27, $28, 1     -- if all tested, proceed to the next
program
--084 1000FFF7 beq $0,  $0, -9     -- else test next data
--088 00000020 add $0, $0, $0      -- noop
--08c 00000020 add $0, $0, $0      -- noop
--
--090 00000020 add $0, $0, $0    -- nop *** INITIALIZATION FOR SELECTION SORT
***
--094 00A01020 add $2, $5, $0    -- set min = 5
--098 00BF4820 add $9, $5, $31   -- $9  = 9
--09c 01215020 add $10, $9, $1   -- $10 = 10
--
--0A0 00003020 add $6, $0, $0    -- slt_result = 0
--0A4 00A01820 add $3, $5, $0    -- i = 5
--0A8 00612020 add $4, $3, $1    -- j = i+1   *** SELECTION SORT STARTS HERE
***
--0Ac 007F6819 mul $13, $3, $31  -- ai = i*4
--
--0B0 8DB70000 lw  $23, 0($13)   -- mi = M(ai)
--0B4 01A06020 add $12, $13, $0  -- amin = ai
--0B8 02E0B020 add $22, $23, $0  -- mmin = mi
--0Bc 009F7019 mul $14, $4, $31  -- aj  = j*4
--
--
--0C0 8DD80000 lw  $24, 0($14)   -- mj = M(aj)
--0C4 0316302A slt $6, $24, $22  -- (mj < mmin)
--0C8 10C00002 beq $6, $0, 2     -- if(no)
--0Cc 01C06020 add $12, $14, $0  -- amin = aj
--
--0D0 0300B020 add $22, $24, $0  -- mmin = mj
--0D4 00812020 add $4, $4, $1    -- j++
--0D8 108A0001 beq $4, $10, 1    -- (j = 10)
--0Dc 1000FFF7 beq $0, $0, -9    -- if(no)
--
--0E0 00000020 add $0, $0, $0    -- nop
--0E4 ADB60001 sw  $22, 0 ($13)  -- M(ai) = mmin // swap
--0E8 AD970001 sw  $23, 0 ($12)  -- M(amin) = mi // swap
--0Ec 00611820 add $3, $3, $1    -- i++
--
--0F0 00612020 add $4, $3, $1    -- j = i+1
--0F4 10690001 beq $3, $9, 1     -- (i==9)
--0F8 1000FFEC beq $0, $0, -20   -- if(no)
--0Fc 00000020 add $0,  $0,  $0  -- nop
--
--
--100 00000020 add $0,  $0,  $0    -- *** CHECKER FOR THE NEXT 5 ITEMS ***
--104 00BFD019 mul $26, $5,  $31   -- addr1 = num_of_items * 4
--108 035FD820 add $27, $26, $31   -- addr2 = addr1 + 4
--10c 00BFE019 mul $28, $5, $31    -- addr3 = num_of_items * 4
--
--110 039AE020 add $28, $28, $26   -- addr3 = addr3 + addr1
--114 8F5D0000 lw  $29, 0 ($26)    -- maddr1 = M(addr1)
--118 8F7E0000 lw  $30, 0 ($27)    -- maddr2 = M(addr2)
--11c 03BEC82A slt $25, $29, $30   -- (maddr1 < maddr2) ?
```

```
--
--120 13390001 beq $25, $25, 1     -- if yes, proceed to the next data
--124 1000FFFF beq $0,  $0, -1     -- else, You're stuck here
--128 035FD020 add $26, $26, $31   -- addr1 = addr1 + 4
--12c 037FD820 add $27, $27, $31   -- addr2 = addr2 + 4
--
--130 137C0001 beq $27, $28, 1     -- if all tested, proceed to the next
program
--134 1000FFF7 beq $0,  $0, -9     -- else test next data
--138 00000020 add $0, $0, $0      -- noop
--13c 00000020 add $0, $0, $0      -- noop
--
--
--REG FILE USED BY BUBBLE SORT
--Initilaly, the content of a register is assumed to be same as its register
number.
--
--$0   ----> 0        constant
--$1   ----> 1        constant
--$2   ----> ak       address of k
--$3   ----> ai       address of i
--$4   ----> aj       address of j
--$5   ----> 5        num_of_items (items at location 0~4 will be sorted)
--$6   ----> result_of_slt
--$13  ----> mi       M(ai)
--$14  ----> mj       M(aj)
--$25~$30 -> RESERVED for the checker
--$31  ----> 4        conatant for calculating word address
--
--REG FILE USED BY SELECTION SORT
--
--$0   ----> 0       constant
--$1   ----> 1       constant
--$2   ----> min     index of the minimum value
--$3   ----> i       index i
--$4   ----> j       index j
--$5   ----> 5       num_of_items (items at location 5~9 will be sorted)
--$6   ----> result of slt
--$9   ----> 9       constant
--$10  ----> 10      constant
--$12  ----> amin    address of min
--$13  ----> ai      address of i
--$14  ----> aj      address of j
--$15~$21 -> don't care
--$22  ----> mmin    M(amin)
--$23  ----> mi      M(ai)
--$24  ----> mj      M(aj)
--$25~$30 -> RESERVED for checker
--$31  ----> 4       for calculating word address
--
--REG FILE USED BY CHECKER
--
--$26  ----> addr1    starting point
--$27  ----> addr2    ending point
--$28  ----> addr3    bound
```

```
--$29  ----> maddr1   M(addr1)
--$30  ----> maddr2   M(addr2)

Programa Dos
-- THE GIVEN PROGRAM FINDS MINIMUM NUMBER FROM THE FIRST 10 LOCATION OF DATA
MEM
-- ADD $31, $0, $0  --$31 is set to X?00000000?.
-- ADD $11, $10, $0 --
-- ADD $30, $14, $0 --
-- ADD $10, $0, $0  --
-- LW $2, 0($31)    --Content of memory location 0 goes to $2
-- BEQ $A, $B, GOTO2-- $A is counter to track 10 numbers. -- INSTRUCTION WITH
LABEL LOOP
-- ADD $31, $31, $4 --$31 is incremented by 4 so it can point to the next
memory location
-- LW $3, 0($31)    --Content of Memory location 2 goes to $3
-- SLT $5, $3, $2   --Check if ($3) < ($2)
-- ADD $10, $10, $1 --Increment counter $10
-- BEQ $5, $1, GOTO1--  If ($3) < ($2) then
-- JMP LOOP         --If ($3) ? ($2) then
-- ADD $2, $3, $0   --Move ($3)-->($2) IF $3< $2  --INSTRUCTION WITH LABEL
GOTO1
-- JMP LOOP         --Jump to loop
-- SW  $2,0($30)    --Store ($2) at memory location 11. ($30)-->4 --
INSTRUCTION WITH LABEL GOTO2
-- LW $6, 0($16)    --Content of memory location 12 goes to $6 --> THIS BRANCH
IS NEVER TAKEN
-- BEQ $2, $6, -4 --If ($2) = ($6) then ? this branch is always taken
-- JMP HERE         -- LOCATION WITH LABEL HERE
```

DISEÑO DEL
Instruction Fetch Queue (IFQ).

Following with the MAS specification this is the resulted code.
Architecture proposed

As you can see in the picture below the architecture proposed complies with the IFQ MAS document. This contains the description data flow and how these registers have been treated to generate the stimulus to control data from the I-Cache module and also describes how the signal controls for the Dispatch Unit are generated.

Control Unit
These are the signals needed to control the flow data from I-cache

```
input jmp_brn_valid_in;       // Jump Branch Valid from Dispatch Unit
input [31:0]jmp_brn_addr;
output [31:0] pc_ic;          // Program Counter Address In for I-Cache
output rden_ic;               // Read Enable Data for I-Cache
output abort_ic;              // Abort Instruction Fetch signal for I-Cache //Wait for jmp address
```

Data path from the I-cache aspect view

First of all, a 128bits data address comes from I-cache, so every single clock cycle if Rd_en signal is present the I-cache module resolved the address petition it delivers the required data to IFQ so this data contains 4 instruction for every address this 4 instructions of 128bits are been storage in queue. For write pointer view this is a 4x128 bits array memory register. Because the architecture has a limited space available to storage data instructions from I-cache. A full queue signal is needed to indicate the I-cache stop sending data this signal this means full_q goes high at time to wp[3:2] are equal to 2'b11.

If Registers descriptions.

```
input [127:0] dataOut_ic;                 // Instruction Data bus from I-cache
input data_valid_ic;                      // Instruction Data Valid from I-cache
reg [127:0] instruc_dataq_reg [3:0];      // Instruction Data Queue Regiter
reg [31:0] instruc_bp_reg;                // Instruction Bypass Regiter
reg [127:0] instruc_rp_reg;               // Instruction read pointer Regiter
reg [31:0] instruc_rp2_reg;               // Instruction read pointer Regiter
```

From the Dispatch view
Verilog Code

```
//`timescale 1ns/100ps
//////////////////////////////////////////////////////////////////////////////////
// Institution: ITESO
// Engineer: Enrique Hernandez , Mario Ruvalcaba
// Subjet: Microprocessors Design
// Instructor: German Fabila
// Create Date:    19:41:38 02/15/2010
// Design Name:
// Module Name:  IFQ_Data_In.v
// Project Name:  Instruction Fetch Queue
// Target Devices: I- Cache, Dispatch Unit
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.02 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module IFQ_Data_In(dataOut_ic, data_valid_ic, clk, rst, jmp_brn_valid_in, jmp_brn_addr, rden_du,
pc_ic, rden_ic, abort_ic, pc_du, instruc_du, empty_du);

input [127:0] dataOut_ic;      // Instruction Data bus from I-cache
input data_valid_ic;           // Instruction Data Valid from I-cache
input clk;                     //  General Clock
input rst;                     //  General Reset
input jmp_brn_valid_in;        // Jump Branch Valid from Dispatch Unit
input [31:0]jmp_brn_addr;      // Jump Branch Address from Dispatch Unit
input rden_du;                 // Read Enable Instruction from Dispatch Unit

output [31:0] pc_ic;           // Program Counter Address In for I-Cache
output rden_ic;                // Read Enable Data for I-Cache
output abort_ic;               // Abort Instruction Fetch for I-Cache //Wait for jmp address

output [31:0] pc_du;           // Program Counter for Dispatch Unit
output [31:0] instruc_du;      // Instruction Data for Dispatch Unit
output empty_du;               // Empty Queue Instruction Fetch for Dispatch Unit

reg rden_ic;                // Read Enable for I-Cache
```

```
reg bypass_ctrl;              // Bypass Signal control
reg [31:0] pcic_cnt;          // Program Counter out for I-Cache
reg [31:0] pcdu_cnt;          // Program Counter out for Dispatch Unit
reg [127:0] instruc_dataq_reg [3:0];     // Instruction Data Queue Regiter
reg [31:0] instruc_bp_reg;     // Instruction Bypass Regiter
reg [127:0] instruc_rp_reg;     // Instruction read pointer Regiter
reg [31:0] instruc_rp2_reg;   //Instruction read pointer Regiter
wire [31:0] pc_du;          // Program Counter to Dispatch Unit
reg [31:0] pc_ic;           // Program Counter out for I-Cache
reg [4:0] wp_cntr;             // Write Pointer counter
reg [4:0] rp_cntr;             // Read Pointer counter
wire pcic_cnt_crtl;          // Program Counter Control out for I-Cache
reg rd_ctrl;               // Read Enable control
reg wr_ctrl;               // Write Enable control
reg abort_ctrl;               // Abort signal control
wire Full_q;               // Bypass Signal control
reg[2:0] current;
reg[2:0] next;
parameter [2:0]s0=3'b000, s1=3'b001, s2=3'b010, s3=3'b011, s4=3'b100;
// ControlPATH begin
always@(posedge clk, posedge rst)begin
  if(rst) begin wr_ctrl <= 1'b0; rd_ctrl <= 1'b0; abort_ctrl <= 1'b0; pcdu_cnt <= 32'h00;
          rden_ic <= 1'b0; bypass_ctrl <= 1'b0; rp_cntr <= 4'b0;
          wp_cntr <= 2'b00; next <= s0;end
   else
   current <= next;
end

always@(*)begin
  case (current)
     s0:  begin pc_ic = pcic_cnt; rden_ic = 1'b1; rp_cntr = 5'h00; wp_cntr = 5'h00;
           pcic_cnt = 32'h00; pcdu_cnt = 32'h00; next = s1; end
     s1: begin  bypass_ctrl = 1'b1; rd_ctrl = 1'b1; wr_ctrl = 1'b1; pcic_cnt = 32'h00000010;
            pc_ic = pcic_cnt; next = s2;end
     s2: if (jmp_brn_valid_in) begin   rden_ic = 1'b1; pc_ic = {jmp_brn_addr[31:4],4'h0};
           pcic_cnt = {jmp_brn_addr[31:5],5'h10}; next = s4;end
         else if(Full_q) begin wr_ctrl = 1'b0; pc_ic = pcic_cnt; bypass_ctrl = 1'b0; rden_ic = 1'b0;
            next = s3;end
           else begin rd_ctrl = 1'b1; wr_ctrl = 1'b1; pc_ic = pcic_cnt;  bypass_ctrl = 1'b0;
           next = s2; end
     s3: if(pcic_cnt_crtl)begin rp_cntr = 5'h1f; wp_cntr = 5'h00; pcic_cnt = pcic_cnt + 16;
         rden_ic = 1'b1; wr_ctrl = 1'b1; rd_ctrl = 1'b1; next = s2; end
          else next = s2;
     s4:  begin wp_cntr = 5'h00; bypass_ctrl = 1'b1; wr_ctrl = 1'b1; rden_ic = 1'b1;
```

```
            rp_cntr = {2'b0,jmp_brn_addr[3:2]}; next = s2;end

       default next = s2;
    endcase
end

 // Write pointer counter
always@(posedge clk)begin
  if(wr_ctrl && data_valid_ic)begin
     wp_cntr <= wp_cntr + 4;
     pcic_cnt <= pcic_cnt + 16; end
   else begin
   wp_cntr <= wp_cntr;
   pcic_cnt <= pcic_cnt; end
end
// Read pointer counter
always@(posedge clk)begin
  if(rd_ctrl && rden_du)
     rp_cntr <= rp_cntr + 1;
    else
   rp_cntr <= rp_cntr;
end

// Program Counter
always@(posedge clk)begin
  if(rp_cntr[1:0]==2'b11)
     pcdu_cnt <= pcdu_cnt + 16;
   else
   pcdu_cnt <= pcdu_cnt;
end



// ControlPath End

/// DataPath begin
/// Instruction Data Queue driven by wr_cntr[3:2], enable by Data out Valid from I-Cache
always @(*)begin
    if (data_valid_ic)
    instruc_dataq_reg[wp_cntr[3:2]][127:0] = dataOut_ic[127:0];
end
/// Instruction Data Queue driven by read pointer rd_ptr[3:2], enable by rden_du
always@(*)begin
    if (rden_du && rd_ctrl )begin
```

```
      instruc_rp_reg[127:0] = instruc_dataq_reg[rp_cntr[3:2]][127:0];
      case(rp_cntr[1:0])
      2'b00: instruc_rp2_reg = instruc_rp_reg[31:0];
      2'b01: instruc_rp2_reg = instruc_rp_reg[63:32];
      2'b10: instruc_rp2_reg = instruc_rp_reg[95:64];
      3'b11: instruc_rp2_reg = instruc_rp_reg[127:96];
      endcase
    /// Instruction Data Register driven by read pointer rd_ptr[1:0]
    case(rp_cntr[1:0])
     2'b00: instruc_bp_reg [31:0] = dataOut_ic[31:0];
     2'b01: instruc_bp_reg [31:0] = dataOut_ic[63:32];
     2'b10: instruc_bp_reg [31:0] = dataOut_ic[95:64];
     3'b11: instruc_bp_reg [31:0] = dataOut_ic[127:96];
endcase
end
   else
      instruc_rp_reg = instruc_dataq_reg[rp_cntr[3:2]][127:0];
  end
  /// DataPath End
  /// Instruction Control Data to Dispatch Unit
 assign  instruc_du = (!bypass_ctrl) ? instruc_rp2_reg : instruc_bp_reg;
 assign  jmp_brn_valid = (jmp_brn_valid_in)?1:0;
 assign  empty_du = (((wp_cntr[4:2] - rp_cntr[4:2])== 0)&& !Full_q) ? 1 : 0;
 assign  Full_q  = (wp_cntr[4:2] == 3'b011)? 1 : 0;
 assign  pcic_cnt_crtl = (rp_cntr == 5'h0f)? 1 : 0;
 assign pc_du = (rden_du && rd_ctrl)? pcdu_cnt:pc_du;
endmodule
```

Test Bench

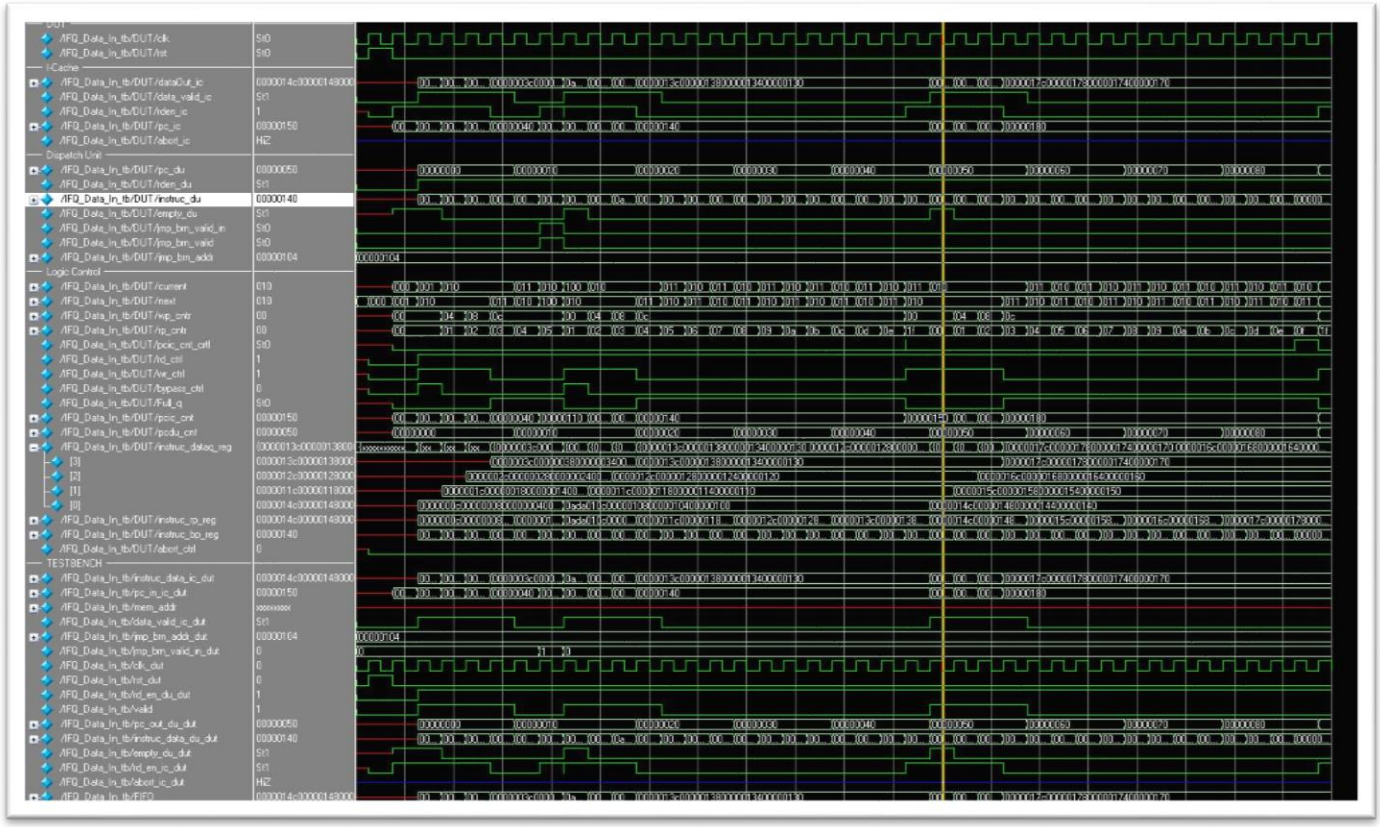
```
`timescale 1ns/100ps
//////////////////////////////////////////////////////////////////////////////
// Institution: ITESO
// Engineer: Enrique Hernandez , Mario Ruvalcaba
// Subjet: Microprocessors Design
// Instructor: German Fabila
// Create Date:    19:41:38 02/15/2010
// Design Name:
// Module Name:  IFQ.v
// Project Name:  Instruction Fetch Queue Test Bench
// Target Devices: I- Cache, Dispatch Unit
// Tool versions:
// Description:
//
// Dependencies:
```

```
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module IFQ_Data_In_tb();

wire [127:0] instruc_data_ic_dut;      // Instruction Data bus In from I-cache
wire data_valid_ic_dut;                // Instruction Data Valid from I-cache
reg clk_dut;               //  General Clock
reg rst_dut;               //  General Reset
reg jmp_brn_valid_in_dut;
reg [31:0]jmp_brn_addr_dut;
reg rd_en_du_dut;
reg valid;
wire [31:0] pc_in_ic_dut;       // Program Counter Address In for I-Cache
wire [31:0] pc_out_du_dut;
wire [31:0] instruc_data_du_dut;
wire empty_du_dut;
wire rd_en_ic_dut;       // Read Enable for I-Cache
wire abort_ic_dut;       // Abort Instruction Fetch for I-Cache

reg[31:0] Inst0;
reg[31:0] Inst1;
reg[31:0] Inst2;
reg[31:0] Inst3;
reg[31:0] mem_addr;
reg[127:0] FIFO;
integer i;

initial begin
    clk_dut = 0;
 forever clk_dut = #5~clk_dut;

end

initial begin
   rst_dut = 1'b0;
   jmp_brn_valid_in_dut = 1'b0;

   valid = 1'b0;
```

```
  #5 rst_dut = 1'b1;
  #10 rst_dut = 1'b0;

end


initial begin
  rd_en_du_dut = 1'b0;
 #25 rd_en_du_dut = 1'b1;
 #380 rd_en_du_dut = 1'b0;
// #20 rd_en_du_dut = 1'b1;
// #140 rd_en_du_dut = 1'b0;


end

initial begin
 jmp_brn_addr_dut = 32'h0000_0104;
 #75 jmp_brn_valid_in_dut = 1'b1;
 #10 jmp_brn_valid_in_dut = 1'b0;
end


always@(posedge clk_dut, posedge rst_dut)begin

  if(rst_dut)  valid <= 1'b0;
  else if (rd_en_ic_dut)begin

 // Instruction Memory
  case (pc_in_ic_dut)
  32'h00000000: FIFO <= 128'h0000000c_00000008_00000004_0000F1F0;
  32'h00000010: FIFO <= 128'h0000001c_00000018_00000014_00000010;
  32'h00000020: FIFO <= 128'h0000002c_00000028_00000024_00000020;
  32'h00000030: FIFO <= 128'h0000003c_00000038_00000034_00000030;
  32'h00000040: FIFO <= 128'h0000004c_00000048_00000044_00000040;
  32'h00000050: FIFO <= 128'h0000005c_00000058_00000054_00000050;
  32'h00000060: FIFO <= 128'h0000006c_00000068_00000064_00000060;
  32'h00000070: FIFO <= 128'h0000007c_00000078_00000074_00000070;
  32'h00000080: FIFO <= 128'h0000008c_00000088_00000084_00000080;
  32'h00000090: FIFO <= 128'h0000009c_00000098_00000094_00000090;
  32'h000000A0: FIFO <= 128'h000000Ac_000000A8_000000A4_000000A0;
  32'h000000B0: FIFO <= 128'h000000Bc_000000B8_000000B4_000000B0;
  32'h000000C0: FIFO <= 128'h000000Cc_000000C8_000000C4_000000C0;
  32'h000000D0: FIFO <= 128'h000000Dc_000000D8_000000D4_000000D0;
```

```
   32'h000000E0: FIFO <= 128'h000000Ec_000000E8_000000E4_000000E0;
   32'h000000F0: FIFO <= 128'h000000Fc_000000F8_000000F4_000000F0;
   32'h00000100: FIFO <= 128'h0ADA010c_00000108_00000104_00000100;
   32'h00000110: FIFO <= 128'h0000011c_00000118_00000114_00000110;
   32'h00000120: FIFO <= 128'h0000012c_00000128_00000124_00000120;
   32'h00000130: FIFO <= 128'h0000013c_00000138_00000134_00000130;
   32'h00000140: FIFO <= 128'h0000014c_00000148_00000144_00000140;
   32'h00000150: FIFO <= 128'h0000015c_00000158_00000154_00000150;
   32'h00000160: FIFO <= 128'h0000016c_00000168_00000164_00000160;
   32'h00000170: FIFO <= 128'h0000017c_00000178_00000174_00000170;
   32'h00000180: FIFO <= 128'h0000018c_00000188_00000184_00000180;
   32'h00000190: FIFO <= 128'h0000019c_00000198_00000194_00000190;
   32'h000001A0: FIFO <= 128'h000001ac_000001a8_000001a4_000001a0;
   32'h000001B0: FIFO <= 128'h000001bc_000001b8_000001b4_000001b0;
   32'h000001C0: FIFO <= 128'h000001cc_000001c8_000001c4_000001c0;
   32'h000001D0: FIFO <= 128'h000001dc_000001d8_000001d4_000001d0;
   32'h000001E0: FIFO <= 128'h000001ec_000001e8_000001e4_000001e0;
   32'h000001F0: FIFO <= 128'h000001fc_000001f8_000101f4_000001f0;

   //default :FIFO <= 128'hBEBE0000_BAD00000_CAFE0000_ABAD0000;
  endcase
  valid <= 1'b1;end
   else
  valid <=1'b0;
end
assign instruc_data_ic_dut = valid ? FIFO:instruc_data_ic_dut;
 assign data_valid_ic_dut = valid ? 1:0;

IFQ_Data_In DUT(
.dataOut_ic(instruc_data_ic_dut),
.data_valid_ic(data_valid_ic_dut),
.clk(clk_dut),
.rst(rst_dut),
.jmp_brn_valid_in(jmp_brn_valid_in_dut),
.jmp_brn_addr(jmp_brn_addr_dut),
.rden_du(rd_en_du_dut),
.pc_ic(pc_in_ic_dut),
.rden_ic(rd_en_ic_dut),
.abort_ic(abort_ic_dut),
.pc_du(pc_out_du_dut),
.instruc_du(instruc_data_du_dut),
.empty_du(empty_du_dut));
Endmodule
```

Simulation Results.



C. PROPUESTA PARA DETERMINAR EL CÁLCULO DEL TAMAÑO IDEAL DE BLOQUE DE CACHÉ

ITESO

Universidad Jesuita
de Guadalajara

Proyecto Final:
Arquitectura de Microprocesadores

Propuesta para determinar el Cálculo del tamaño ideal de bloque de caché

Alumnos:
Enrique Israel Hernández Meza
Pablo Márquez

Propuesta para determinar el Cálculo del tamaño ideal de bloque de caché

Abstracto:

En el presente trabajo mostraremos el método usado para determinar el tamaño ideal para el bloque de memoria cache utilizado en un microprocesador experimental. Dicha propuesta se sustenta en la medición del tamaño de los datos que son leídos de la memoria cache de datos, en cada ocasión que estos son requeridos y la frecuencia con la que una dirección de memoria es utilizada a lo largo de la ejecución de cualquier aplicación de cómputo.

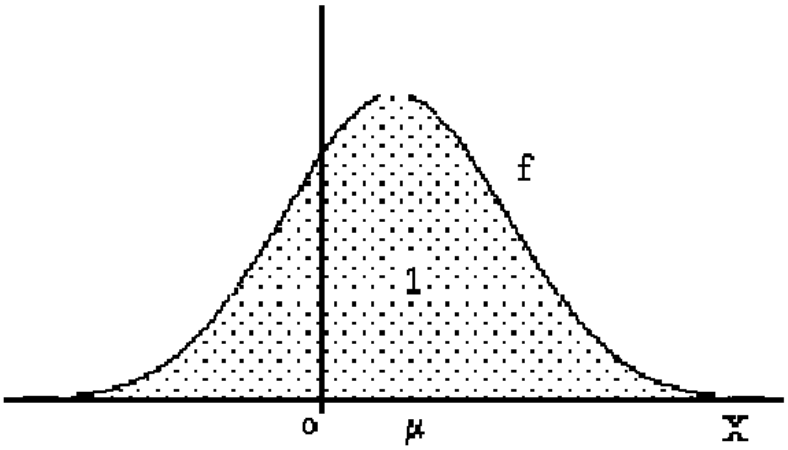
Así mismo comprobaremos en el estado del arte los resultados que obtuvimos de los análisis que se hicieron al comportamiento de las lecturas de cache como fundamento para determinar el tamaño ideal de cache y mostraremos los resultados de las diferentes herramientas de benchmarks que se aplicaron al simulador una vez modificado el tamaño de la cache. Demostrando que la tendencia del rendimiento con nuestro tamaño de cache propuesto es similar o mejor para ambos tamaños de cache, las ventajas de determinar un tamaño idóneo de línea de cache se verían reflejadas en el hecho de optimizar la cantidad de transferencias de datos de memoria principal a las memorias caches.

Marco Teórico:

El tamaño de bloque de caché típicamente usado es de 64 bytes, por estándar. Sin embargo, es posible que para algunas aplicaciones ese tamaño no sea ideal. Por eso es importante estudiar las características de las aplicaciones para determinar cuál es el tamaño ideal en cada caso y de forma general.

La distribución de Gaussiana es una herramienta que se utiliza para determinar el comportamiento que está siguiendo un patrón determinada de muestras, si la cantidad de estas muestras son demasiado grandes, es factible que describa eventos físicos. Nuestra propuesta contempla el análisis de una cantidad considerable de muestras, obtenidas por el simulador, para lo cual decidimos aplicar el análisis estadístico de la distribución gaussiana, como método para determinar el tamaño óptimo que se requiere tener en la memoria de cache de datos, para un marco de aplicaciones determinadas. Logrando con ello un acertado resultado en el cumplimiento de nuestro objetivo.

Usando diferentes aplicaciones como estímulo para nuestro sistema de simulación y en base a las características que proporciona cada una de estas aplicaciones, obtendremos los datos que nos permitirán graficar la distribución de la campana gauss, usando la frecuencia de aparición del área bajo su curva como valor óptimo para determinar el tamaño de la cache.



Nuestra propuesta está basada en las siguientes suposiciones, enumerando las que nos permitirán determinar cuál es el comportamiento que tienen las operaciones de lectura de datos una aplicación a lo largo de su ejecución en el microprocesador.

Los accesos a memoria de Datos cache son a través de lecturas de 64bits, 32 bits, 16bits, 8 bits, 0bits.

El tamaño del bloque de memoria principal es igual al tamaño del bloque de memoria cache, esto implica que una localidad de memoria cache incluye una y solo una localidad de memoria principal, (este hecho es relevante para soportar la manera en que trabaja nuestro algoritmo de determinación de tamaño de lecturas).

Basamos nuestra hipótesis en el supuesto de que los accesos a memoria cache de datos tendrán que tener un comportamiento que puede ser modelado bajo el análisis estadístico de la distribución gaussiana. Donde la mayor cantidad de ocurrencias determinara el tamaño de los datos que se leyeron de la memoria.

La instrumentación en la que estamos basando la propuesta, versa en muestrear el número de veces que ocurre la ejecución de la micro instrucción del tipo load para acceder a los datos a memoria cache, diferenciar el número de eventos en los que ocurrieron los accesos a la misma

dirección de memoria cache contabilizando cada uno los diferentes tamaños de longitud posible, por dirección de memoria leída.

Instrumentación.

La instrumentación comprende el desarrollo de un algoritmo que permita la inclusión en la etapa de issue, de decodificación y de writeback; para la captura de las micros operaciones que se utilizan para acceder a los datos que se encuentran alojados en memoria cache y que son requeridos por las macro operaciones para la realización de sus operaciones. Este algoritmo debe de contar con los siguientes requerimientos:

1. Ser capaz de identificar la micro operación de load.

2. Mantener el registro de la dirección de memoria que está siendo leída.

3. Mantener el tamaño del dato que está siendo leído de la dirección de memoria.

4. Acumular el tamaño total de datos que se leyeron de una misma dirección de memoria.

5. Almacenar los datos en un reporte para su posterior análisis e interpretación de resultados.

6. El código debe de ser intrusivo pero no debe de afectar el rendimiento actual del simulador

Etapa Issue.

Para la etapa de issue se hicieron incorporaciones de código para instrumentar el comportamiento de las micro operaciones de load y effective address, con el objetivo determinar la dirección física que necesita para traer el dato de la memoria cache, así como el número de secuencia de la instrucción. Los datos que se obtienen son utilizados como base de post análisis para corroborar que la veracidad de los datos usados para determinar la frecuencia con la que una dirección de memoria de cache de datos es utilizada, en la etapa de writeback.

```
//eihernan
FILE *regfile;
regfile = fopen("issueReport.log","a");
if (!encabezadopresent1){
printf("\nissueReport debug file created\n");
fprintf(regfile,"uop: mem_phadd: load: load_ptr:\n");
encabezadopresent1++;
```

```
}
if (sample1<maxsamples1){
fprintf(regfile,"%x %x %x\n",load->uop,load->mem_phaddr,load,*load);
sample1++;
}
if (sample1==maxsamples1){
printf("\nEnd collection of data from issueReport\n");
sample1++;
}
```

Esta instrumentación nos permitió identificar él como algunas direcciones de memoria son accedidas de manera secuencial.

Etapa de Decodificación.

En esta etapa se realizó la modificación del archivo uop.c para incorporar la instrumentación del código necesario que permita la identificación de la micro operación de lectura a cache (load) que va a ser utilizada en etapas posteriores, también se incluyó en la estructura "uop_t" (estructura de datos de las micro operaciones) un nuevo elemento que nos proporciona el tamaño de la memoria que se leerá de la cache, este elemento además nos permitirá identificar en etapas posteriores el tamaño del dato leído de la memoria cache de datos.

Además trabajamos en la inclusión de las dependencias (idep y odep) correspondiente a la operación y la extracción de la información de la dirección efectiva para el proceso de análisis, como base para determinar la mejor aproximación en la solución que proponemos.

A continuación se anexa el código que se utilizó.

```
uop: mem_phadd: data:
14 1484 15ffaef0
14 1484 15ffbc10
14 2888 15ffd7d0
14 4484 15ffec00
14 4484 15fff920
14 5888 160014e0
14 5888 1603dae0
14 7484 16002960
14 7484 16003680
14 8888 16005240
14 a484 16006700
```

Etapa de Writeback.

En esta etapa se instrumentó la parte fuerte de nuestro algoritmo de captura y procesamiento de datos para reducir en un máximo posible el tamaño del archivo que contiene los datos necesarios para su posterior análisis y procesamiento. Y con ello demostrar en la distribución gaussiana el comportamiento de los accesos a memoria cache de datos.

Básicamente lo que el código hace en esta etapa, es la selección de micro operaciones que hacen acceso a memoria cache de datos, el procesamiento de la dirección de donde se obtuvo el dato, el almacenamiento de la longitud del dato leído; el cual nos es proporcionado por la modificación realizada en la etapa de decodificación, el agrupamiento en eventos de la ocurrencia del tamaño de esos datos, la contabilización y la frecuencia con la que se fueron utilizados por dirección de memoria cache.

A continuación se incluye el código que se utilizó:

```
//prmarque
//crear archive para correlacionar el calculo de la direccion para uop pareadas (Effaddress-Load)
// uop->datasize=dep;
FILE *regfile;
regfile = fopen("ExamLog.log","a");
if (!encabezadopresent2){
        fprintf(regfile,"UOP SEQ Adduopeip seg:base:index Datalenght\n");
        encabezadopresent2++;
        }
if(sample1<maxsamples1){
        fprintf(regfile,"EffAdd:%x %x %x %x:%x:%x %x\n",uop->uop,uop->seq, uop->eip,uop-
        >idep[0],uop->idep[1],uop->idep[2],dep);
        sample1++;
        }
//end prmarque
//prmarque
// uop->datasize=dep;
FILE *regfile;
regfile = fopen("ExamLog.log","a");
if (!encabezadopresent2){
        fprintf(regfile,"UOP SEQ Adduopeip seg:base:index Datalenght\n");
        encabezadopresent2++;
        }
if(sample1<maxsamples1){
        fprintf(regfile,"EffAdd:%x %x %x %x:%x:%x %x\n",uop->uop,uop->seq, uop->eip,uop-
        >idep[0],uop->idep[1],uop->idep[2],dep);
        sample1++;
        }
//end prmarque

//prmarque - ihernan
if(uop->uop== 0x14){
```

```
	// uop->datasize = dep;
	int data_valor;
	value = isa_load_reg(reg_eax);
	//printf("reg_eax=%x\n",value);
	FILE *regfile;
	regfile = fopen("Fromwriteback.log","a");
	if (!encabezadopresent){
		printf("\nwriteback debug file created\n");
		// fprintf(regfile,"UOP SEQ AddMacroIns DAUXidep[0] idep[1] idep[2] Data=odep[0]
		Data=odep[1] odep[2]
		mem_phadd EAXvalue Datasize\n");
		fprintf(regfile," memaddress totalcache cache_reads\n");
		encabezadopresent++;
	}
if (sample0<maxsamples){
	data_valor = uop->datasize;
	//printf("datasize %x\n",uop->datasize);
	memactual = uop->mem_phaddr;
	if (memactual == mempass) {
		switch( data_valor ){
			case 0x100 :
			readsize = 1;
			break;
			case 0x101:
			readsize = 2;
			break;
			case 0x102:
			readsize = 4;
			break;
			default:
			readsize = 0;
			}
		totalcache += readsize;
		flagequal ++;
	}
else if(mempass!=0)
{
	fprintf(regfile,"%x %x %x\n",mempass,totalcache,flagequal);
totalcache=0;
flagequal=0;
switch(data_valor){
		case 0x100:
		readsize = 1;
		break;
		case 0x101:
		readsize = 2;
		break;
		case 0x102:
		readsize = 4;
		break;
		default:
		readsize = 0;
		}
totalcache += readsize;
```

```
flagequal ++;
mempass = memactual;
}
else {
        switch(data_valor){
                case 0x100:
                readsize = 1;
                break;
                case 0x101:
                readsize = 2;
                break;
                case 0x102:
                readsize = 4;
                break;
                default:
                readsize = 0;
                }
totalcache = readsize;
flagequal ++;
mempass = memactual;
sample0++;
}
if (sample0==maxsamples){
        printf("\nEnd collection of data from WRITEBACK\n");
        sample0++;
}
fclose(regfile);
}
```

Esta instrumentación nos permitió obtener los datos necesarios para hacer la caracterización del comportamiento de los accesos a memoria cache de los diferentes benchmarks utilizados, los datos obtenidos se muestran a continuación, para cada uno de los benchmarks tenemos un par de graficas que describen los datos obtenidos, estas graficas son:

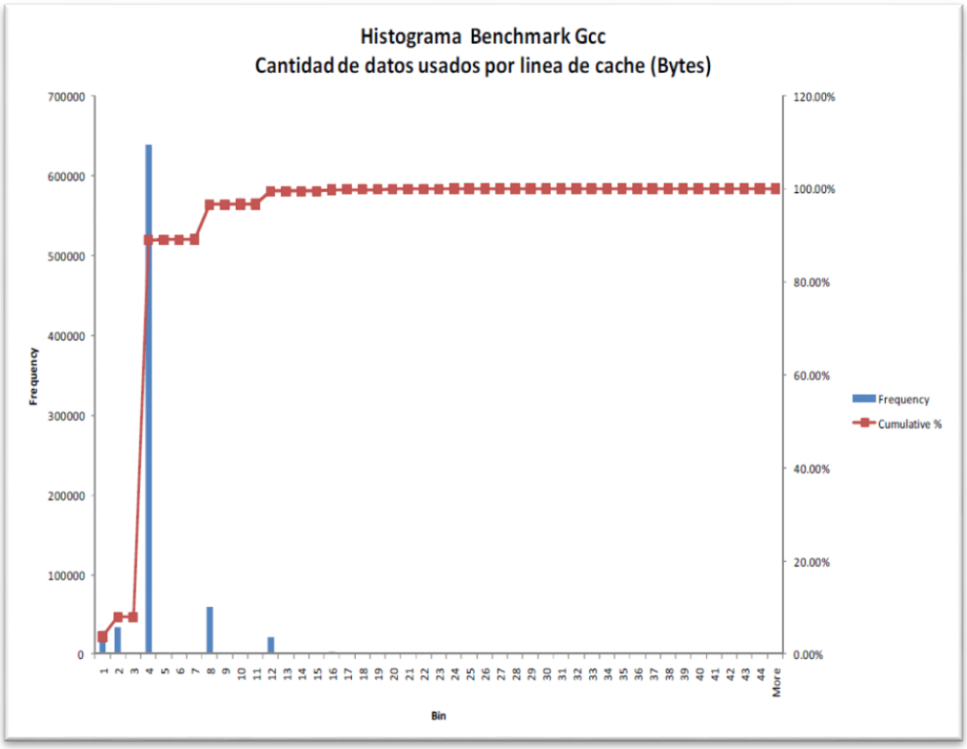
1. Distribución de las lecturas de cache por ocurrencia (datos agrupados por dirección de memoria que corresponden a una misma línea de cache.)

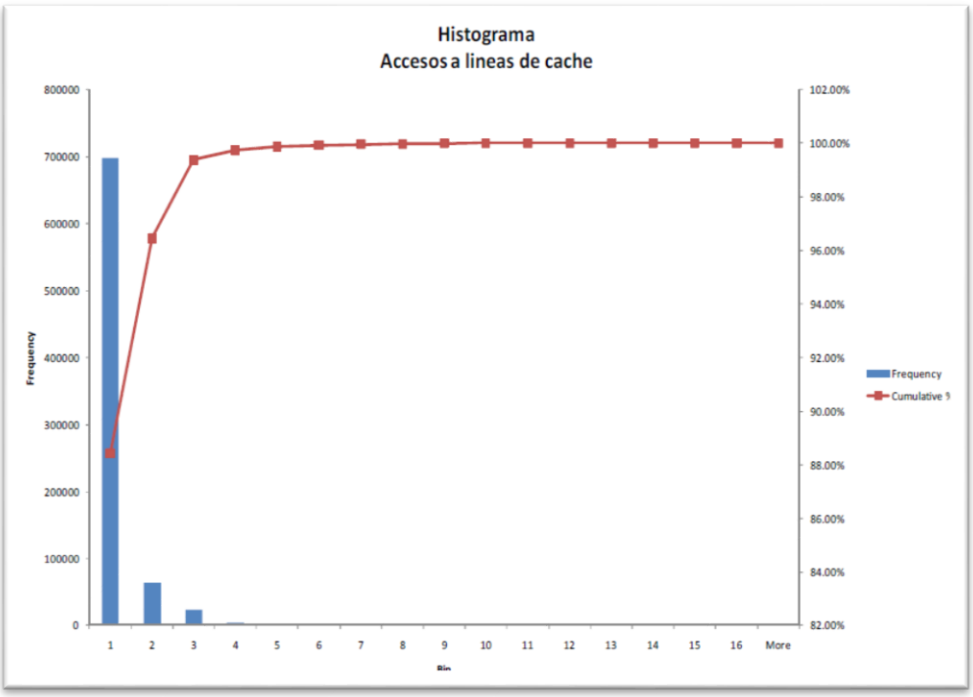
2. Distribución de la cantidad de accesos múltiples a una misma línea de cache.

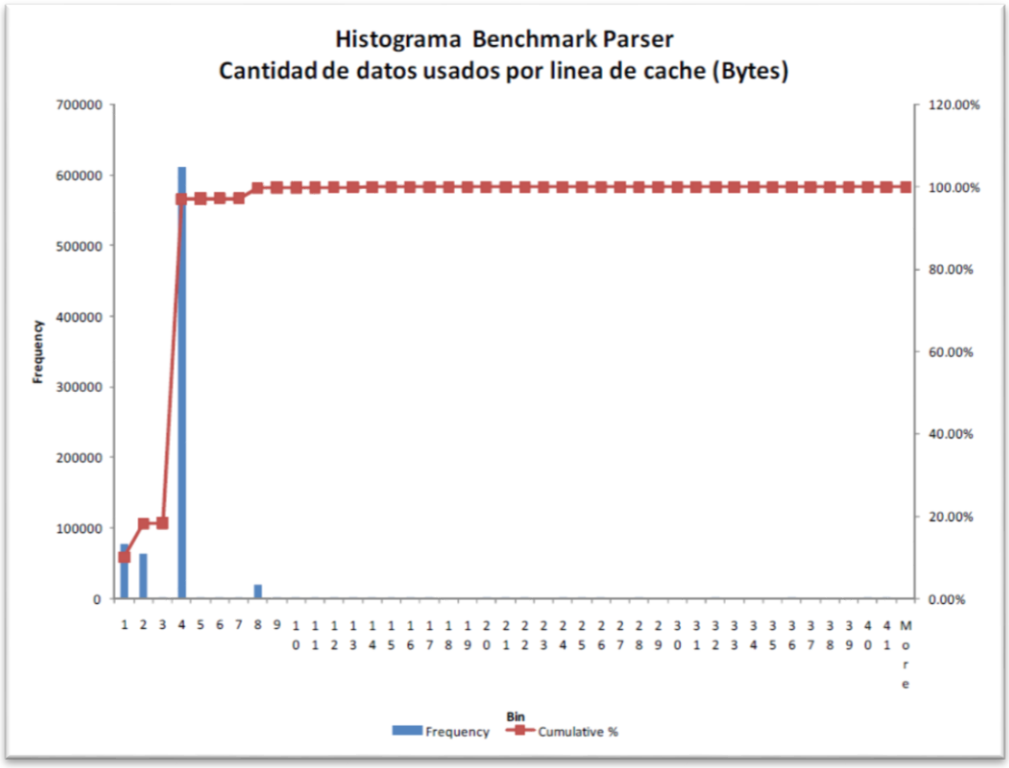
Los bencharks utilizados para este estudio fueron los siguientes:

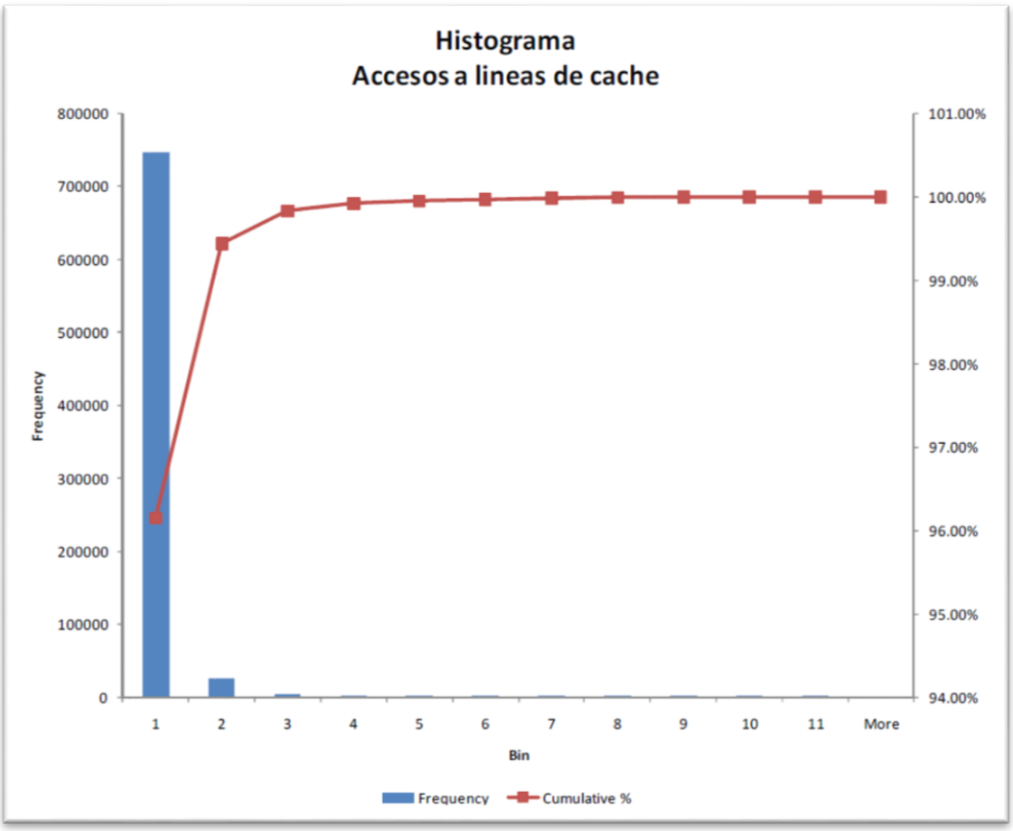
- Gcc
- Pearl
- Bzip2
- Gzip
- Parser

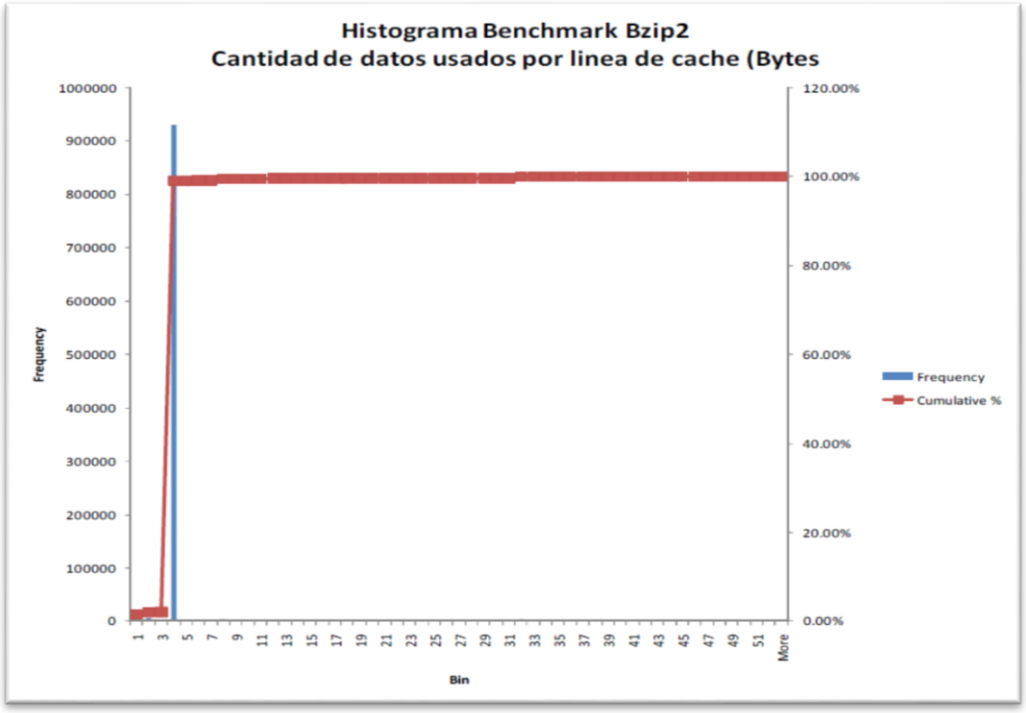
Los resultados obtenidos se muestran a continuación:

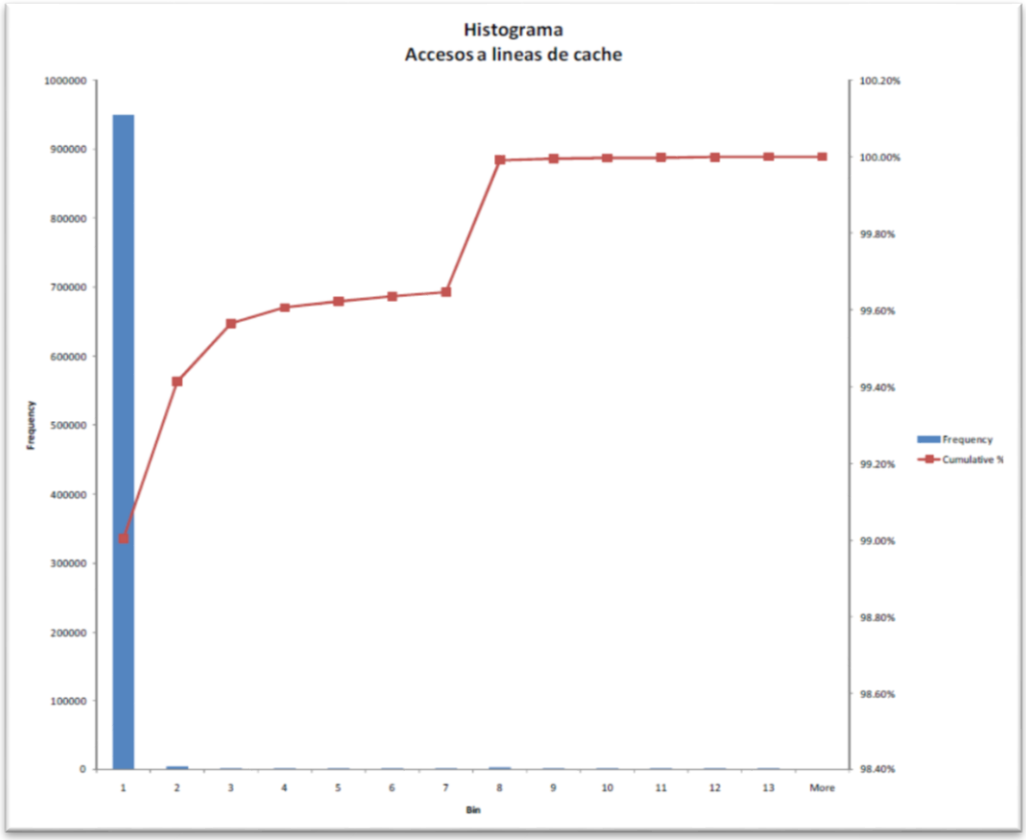

Histograma Benchmark Gcc
Cantidad de datos usados por linea de cache (Bytes)
Frequency
Bin
Frequency
Cumulative %

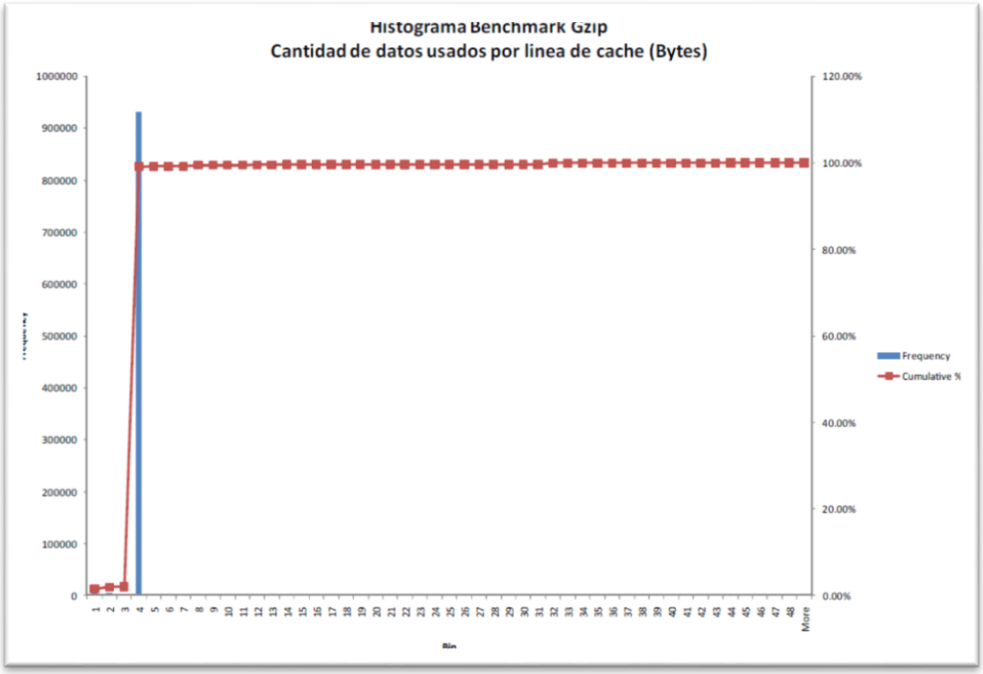

Histograma
Accesos a lineas de cache
Frequency
Bin
Frequency
Cumulative %

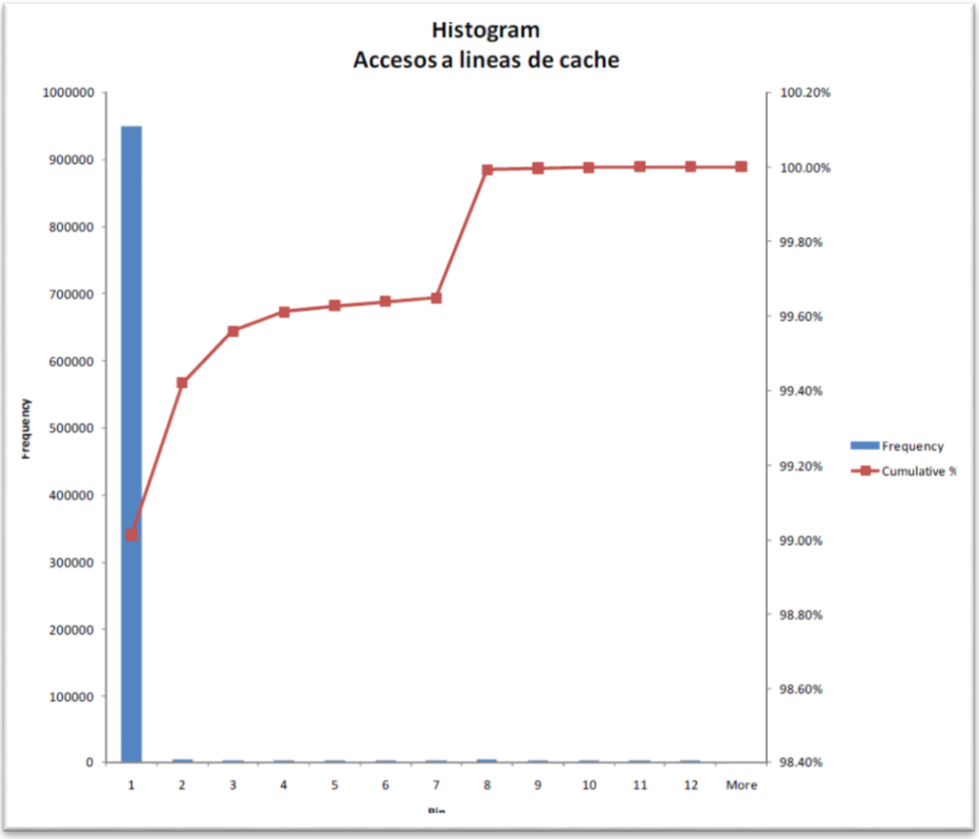

Histograma Benchmark Parser
Cantidad de datos usados por linea de cache (Bytes)
Frequency
Bin
Frequency
Cumulative %

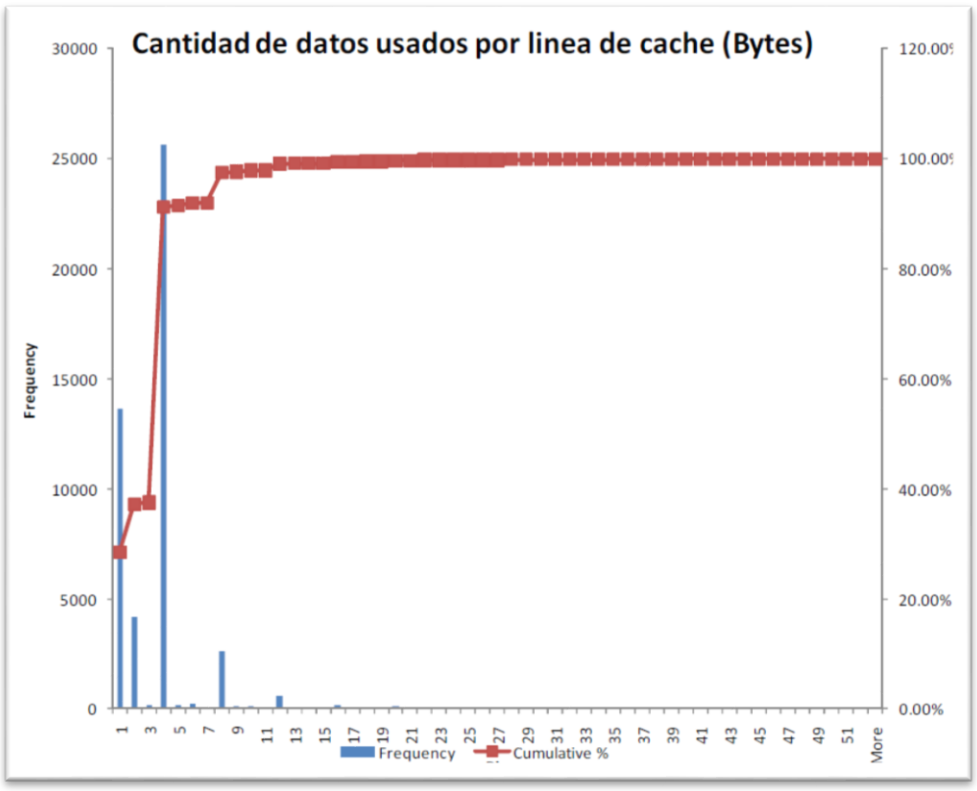

Histograma
Accesos a lineas de cache
Frequency
Bin
1
2
3
4
5
6
7
8
9
10
11
More
Frequency
Cumulative %

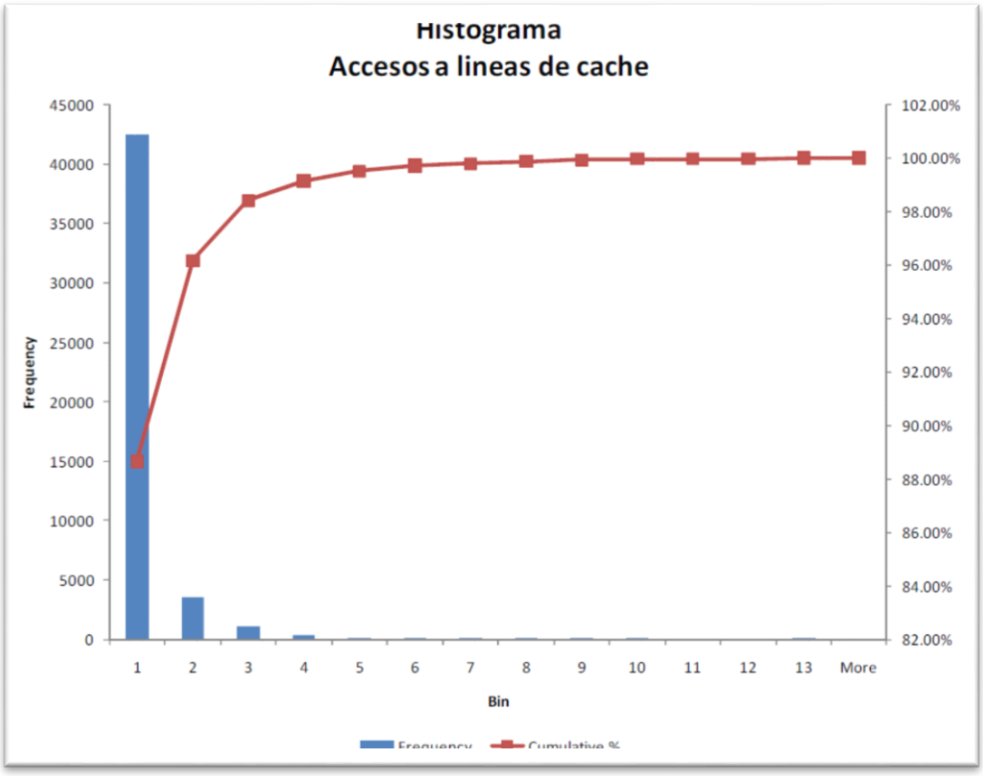

Histograma Benchmark Bzip2
Cantidad de datos usados por linea de cache (Bytes
Frequency
Bin
Frequency
Cumulative %
1000000
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
31
33
35
37
39
41
43
45
47
49
51
More


Histograma
Accesos a lineas de cache
Frequency
Bin
Frequency
Cumulative %
1000000
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
100.20%
100.00%
99.80%
99.60%
99.40%
99.20%
99.00%
98.80%
98.60%
98.40%
1
2
3
4
5
6
7
8
9
10
11
12
13
More


Histograma Benchmark Gzip
Cantidad de datos usados por linea de cache (Bytes)
1000000
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
Frequency
Cumulative %
More


Histogram
Accesos a lineas de cache
Frequency
1000000
900000
800000
700000
600000
500000
400000
300000
200000
100000
0
100.20%
100.00%
99.80%
99.60%
99.40%
99.20%
99.00%
98.80%
98.60%
98.40%
1
2
3
4
5
6
7
8
9
10
11
12
More
Frequency
Cumulative %


Cantidad de datos usados por linea de cache (Bytes)
Frequency
30000
25000
20000
15000
10000
5000
0
120.00%
100.00%
80.00%
60.00%
40.00%
20.00%
0.00%
1
3
5
7
9
11
13
15
17
19
21
23
25
27
29
31
33
35
37
39
41
43
45
47
49
51
More
Frequency
Cumulative %


Histograma
Accesos a lineas de cache
Frequency
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
102.00%
100.00%
98.00%
96.00%
94.00%
92.00%
90.00%
88.00%
86.00%
84.00%
82.00%
1
2
3
4
5
6
7
8
9
10
11
12
13
More
Bin

Basado en el análisis de los datos, podemos observar que la mayor ocurrencia de tamaño de lecturas de cache se dan en tamaños de 4 bytes (basados en el histograma), y estos datos son obtenidos mayormente en una sola transferencia (segunda grafica para cada benchmark), la siguiente tabla muestra los datos relevantes de este comportamiento, además esta tabla muestra la diferencia entre los IPC usando líneas de cache de 64 Bytes y de 32 Bytes, cabe resaltar que nuestra propuesta de mejor tamaño de línea de cache se basa en el hecho de que aproximadamente el 95% de los accesos a memoria cache implican la utilización de tamaño de datos de 4 a 8 bytes.

Nuestra propuesta implica una reducción del tamaño de las líneas de cache hasta un mínimo de 32 o 16 bytes, este tamaño de línea estaría sujeto a determinarla mediante un mayor número de corridas de simulación, que por la naturaleza de este proyecto no es posible hacerlo.

Benchmark	Tamano de lectura de Mayor %		Lecturas cubiertas al 95%	IPC@64 Bytes cache line	IPC@32 Bytes cache line (25 000 000 inst)
Gcc	4 bytes	81%	8 Bytes	16.43	25.21
Pearl	4 bytes	54%	8 Bytes	5.331	5.117
bzip2	4 bytes	97%	4 bytes	2.586	5.864
gzip	4 bytes	97%	4 bytes	2.439	6.284
parser	4 bytes	79%	4 bytes	22.69	5.338

Conclusiones:

Basado en los datos obtenidos, inferimos que el tamaño comúnmente usado de la línea de cache, pareciera que esta de alguna manera siendo sub utilizado, esta inferencia está sujeta a verificación mediante corridas de simulación más extensas y con mayor número de benchamrks.

La sub utilización que mencionamos en el párrafo anterior es asumiendo que esta arquitectura seria para estas aplicaciones en específico, tal subutilización no sería real para máquinas de propósito general en donde no se conoce el tipo de aplicaciones que estarían corriendo sobre estas.

Como trabajo a futuro que pretendemos desarrollar seria la conclusión de las simulaciones con diferentes tamaños de línea de cache (menores a 64 bytes) hasta llegar a un tamaño de 16 bytes el cual nosotros determinamos seria el ideal para esta aplicaciones que se utilizaron.