UNIVERSIDAD TÉCNICA ESTATAL DE QUEVEDO

CIENCIAS DE LA INGENIERÍA

INGENIERÍA EN SOFTWARE

ECUADOR

Aritmética de la computadora y álgebra de Boole.

AUTORES:

ALMAGRO INTRIAGO LAINER PATRICIO

FUERTES ARRAES EDSON DANIEL

PEREZ RUIZ CARLOS ANDRES

ROCHINA ROCHINA FREDDY DAVID

DOCENTES:

ING. GUERRERO ULLOA GLEISTON CICERON

CURSO/PARALELO:

SEGUNDO SOFTWARE “B”

SPA 2025 – 2026

Índice

Objetivo

Aplicar conceptos de aritmética binaria y decimal, álgebra booleana y circuitos lógicos para comprender el funcionamiento de los microprocesadores, microcontroladores y sistemas digitales

Evolución y Arquitectura de los Microprocesadores.

Desde los primeros modelos de un solo bit como el MC14500B de Motorola en 1977, hasta los modernos de 128 bits, la evolución de los microprocesadores ha estado marcada por avances en integración, velocidad, eficiencia energética y complejidad arquitectónica. Los primeros microprocesadores estaban limitados por tecnologías PMOS o NMOS y velocidades de reloj bajas (~1 MHz), sin buses de dirección integrados, dependiendo de circuitos externos [1][2].

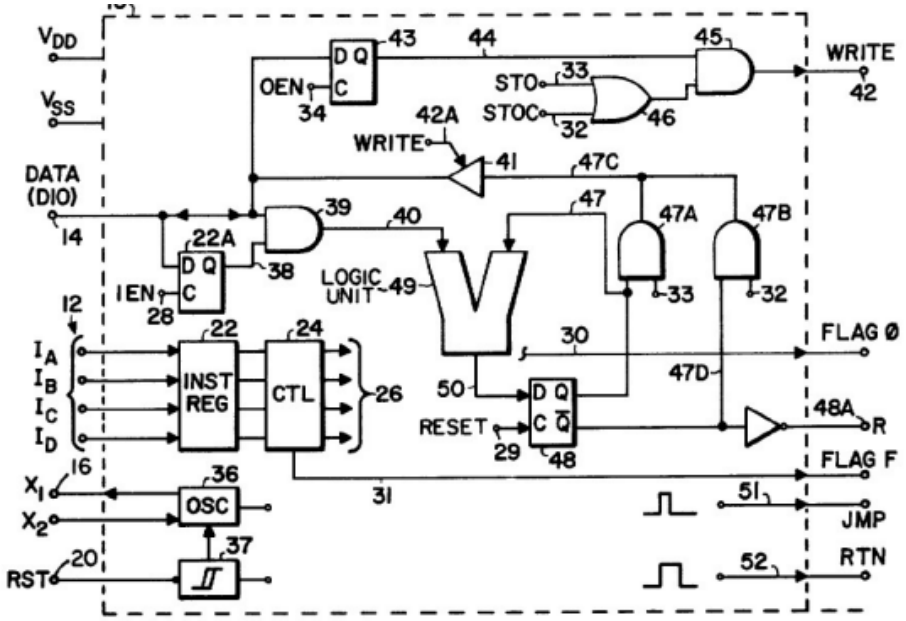


Figura 1 Diagrama de bloques del MC14500B (Gregory 1979)

Con el tiempo, la transición a CMOS permitió reducir el consumo y aumentar la densidad de transistores, acercándose a tamaños de grano de 0,2–0,1 µm a finales del siglo XX, con más de 80 millones de transistores en un solo chip y latencias de puerta inferiores a 0,2 ns [3].

Se consolidaron dos grandes filosofías arquitectónicas como la de Von Neumann, con programa y datos compartiendo bus y Harvard, con separación física de instrucciones y datos.

Paralelamente, la aparición de arquitecturas RISC a finales de los 80 triplicó el rendimiento (de 1 MIPS a 300 MIPS entre 1980 y 1995) gracias a su simplicidad de diseño y alta eficiencia, consolidando estrategias como la ejecución Superscalar, superpipelining y VLIW como mecanismos para extraer paralelismo a nivel de instrucción [3].

Buses y Espacios de Direccionamiento

Instrucciones y Modos de Direccionamiento

La evolución del set de instrucciones (ISA) ha sido crucial. Se pasó de 46 instrucciones (Intel 4004) a más de 150 en el Z80. Se introdujeron modos de direccionamiento sofisticados como indirecto, indexado y relativo, permitiendo más flexibilidad en programación y acceso a memoria [2].

Con RISC, las instrucciones se simplificaron, incrementando el paralelismo mediante técnicas como la predicción de saltos, la prelectura de instrucciones y pipelines extendidos, permitiendo alcanzar cientos de MIPS [3].

Control del Tiempo y Señalización

Los microprocesadores modernos dependen de clocks de alta frecuencia (>100 MHz) y sincronización precisa con señales como Read/#Write, #INT, #NMI, #Reset. También se requiere adaptación eléctrica entre señales mediante componentes como buffers, resistencias de pull-up o level shifters para asegurar compatibilidad lógica y eléctrica [1].

Tendencias Futuras y Diseño Sostenible

Integración de enlaces de comunicación, circuitos de sincronización y mecanismos de coherencia para soportar sistemas multiprocesador y clústeres de CPUs de propósito general. Investigación en microprocesadores ópticos y materiales con mayores gaps para mejorar la velocidad de señalización y reducir interferencias [3].

Microcontroladores

Los microcontroladores (MCUs) son componentes esenciales en muchos dispositivos electrónicos actuales. Integran en un solo chip un procesador, memoria y periféricos, lo que los hace ideales para tareas de control en sistemas embebidos como electrodomésticos, sensores, automóviles o equipos del Internet de las Cosas (IoT) [4].

Elegir un microcontrolador adecuado depende de varios factores, como su arquitectura (8, 16 o 32 bits), frecuencia de reloj, consumo de energía, periféricos integrados y herramientas de desarrollo. Para comparar su desempeño se utilizan benchmarks, que permiten evaluar su velocidad, eficiencia energética y rendimiento en tareas específicas [4].

En aplicaciones IoT, donde los dispositivos deben funcionar con baterías por largos periodos, la eficiencia energética es clave. Se usan técnicas como:

- **DVFS (Dynamic Voltage and Frequency Scaling):** reduce voltaje y frecuencia cuando la carga de trabajo es baja.
- **Modos de suspensión:** disminuyen el consumo al apagar componentes inactivos.
- **Agrupamiento de tareas (batching):** acumula procesos y los ejecuta juntos para ahorrar energía.
- **Optimización del código:** evita cálculos innecesarios o accesos frecuentes a memoria.

Hacia el futuro, los microcontroladores evolucionan con arquitecturas más complejas y eficientes: múltiples núcleos, soporte para inteligencia artificial, procesamiento en el borde (edge computing) y aceleradores específicos. Esto permitirá aplicaciones más potentes y autónomas sin comprometer el consumo [5].

Velocidad De Un Microprocesador

Fallas De Velocidad De Un Microprocesador

El alto rendimiento en microprocesadores mejora la rentabilidad, pero al aumentar la frecuencia también surgen fallas de velocidad. Estas fallas, aunque sean pocas, requieren un análisis detallado para identificar sus causas y mejorar el diseño [6].

Para detectar fallas de velocidad, se analizan rutas críticas del circuito que limitan la frecuencia. Este proceso incluye simulaciones lógicas y de temporización, además del uso de herramientas como el:

CAD que sirve para aislar errores en etapas específicas del reloj y entender su origen.

Las fallas de velocidad pueden deberse a factores como la caída de voltaje localizada, el acoplamiento cruzado o la conmutación de entradas. En el estudio del microprocesador P4 de 65 nm, se identificaron varias causas, siendo el evento dI/dt uno de los más frecuentes [6].

Velocidad Del Microprocesador En Sistemas IIot De Protección

La velocidad es un aspecto clave en los dispositivos de protección de relé, ya que mejora la respuesta del sistema, reduce el tamaño del gabinete y permite su integración con otros equipos. Este parámetro depende directamente del microprocesador y de tecnologías como IIoT, que significa “Industrial Internet of Things”, que en español se traduce como Internet Industrial de las Cosas [7].

El estudio evaluó la velocidad de procesamiento de dispositivos de protección mediante pruebas y simulaciones, revelando tiempos de respuesta eficientes. El uso de microprocesadores como el ESP32, o el XILINX Zynq7000.

Son plataformas que permitieron lograr la baja latencia y alto rendimiento. La velocidad no solo es un indicador de evaluación junto con fiabilidad y sensibilidad, sino también una métrica directa del rendimiento del sistema. El procesamiento rápido y la transmisión efectiva de datos nos muestran cómo el desempeño del microprocesador modifica la rapidez del funcionamiento [7].

Velocidad De Un Microprocesador Y Ruido Digital

Según Meloth, (2016). En el libro: “A case study on radiated emission in a high speed microprocessor based design, nos indica lo siguiente acerca la velocidad de un microprocesador y ruido digital [8].

El estudio se enfoca en el diseño de microprocesadores de alta velocidad, donde la velocidad de operación es un factor crucial que genera desafíos técnicos, especialmente en la gestión de emisiones electromagnéticas.

A medida que los microprocesadores y semiconductores funcionan a frecuencias más altas y con tasas de conmutación más rápidas, aumentan las perturbaciones en el espectro de frecuencias, afectando la integridad electromagnética del sistema.

La circuitería de reloj dentro de los sistemas microcontroladores es identificada como la principal fuente de ruido de banda ancha. En un ejemplo específico, un microprocesador de 32 bits con una frecuencia de núcleo de 266 MHz genera emisiones en su frecuencia base y en sus armónicos, llegando a frecuencias superiores a 1 GHz.

Esto evidencia la relación directa entre la velocidad del microprocesador y las emisiones radiadas, que pueden interferir con otros componentes o sistemas.

Para reducir el ruido digital de alta frecuencia, se recomienda colocar condensadores de desacoplamiento cerca del microprocesador. Es mejor usar materiales dieléctricos como COG o NPO en los condensadores pequeños, ya que son más efectivos para absorber el ruido en frecuencias elevadas.

Esto ayuda a evitar interferencias causadas por la conmutación digital y mejora la estabilidad del sistema. Además, las interfaces utilizadas en el desarrollo, como la interfaz BDM que opera a frecuencias derivadas de la PLL interna del microprocesador.

También contribuyen a las emisiones electromagnéticas. Aunque estas señales son necesarias para la depuración y la programación, su gestión adecuada es fundamental para reducir interferencias en el producto final y así poder garantizar un desempeño óptimo en entornos industriales o comerciales.

Mejora De La Velocidad De Microprocesador Mediante Mim On-Die

Según Sanchez et al. (2006), En el libro: Increasing Microprocessor Speed by Massive Application of On-Die High-K MIM Decoupling Capacitors, nos indica lo siguiente sobre la mejora de la velocidad de los microprocesadores [9].

Uno de los problemas que limita la velocidad del microprocesador es la caída de voltaje que ocurre dentro del chip cuando hay una demanda alta de corriente. Esto pasa por la propia estructura del encapsulado, y afecta más a diseños pequeños y rápidos.

On-die: significa “dentro del chip”. Esto se refiere a los componentes integrados directamente en el microprocesador.

Para mejorar la velocidad, se propuso usar capacitores especiales llamados MIM de alta constante dieléctrica, integrados directamente en el chip. Estos capacitores ayudaron a aumentar la frecuencia del núcleo del procesador hasta en un 10%.

MIM (Metal-Insulator-Metal): esto significa el tipo de capacitor formado por capas metálicas y materiales dieléctricos que permiten mejorar el rendimiento eléctrico.

Estos capacitores reducen las caídas de voltaje en la fuente principal del núcleo. Por ejemplo, una caída de 250 milivoltios bajó a 156 milivoltios, lo cual ayuda a que el procesador trabaje de forma más estable y rápida.

Vdd: es la fuente de alimentación principal del núcleo del microprocesador.

Además de mejorar la velocidad del núcleo, los capacitores MIM también ayudan en otras partes del chip, como las interfaces de alta velocidad y los circuitos PLL, ocupando menos espacio y mejorando la alimentación interna del sistema.

PLL (Phase-Locked Loop): circuito que genera señales de reloj sincronizadas y estables, muy usado en procesadores e interfaces digitales.

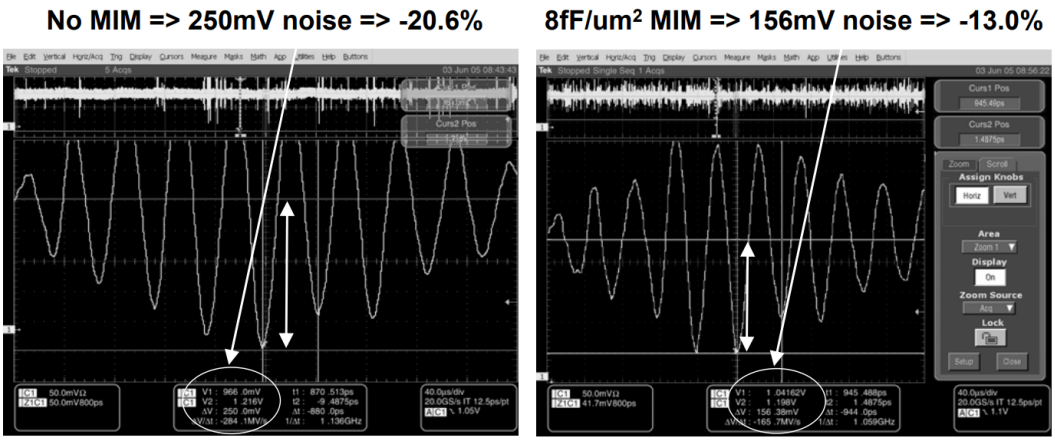


Figura 3 Caídas de voltaje

La figura 3 nos ayuda a entender el problema de la caída de voltaje y cómo los capacitores MIM lo solucionan. A la izquierda se muestra una medición del voltaje del núcleo del procesador sin los capacitores, donde se nota una caída de 250 mV. En cambio, en la imagen de la derecha, con capacitores MIM aplicados, esa caída se reduce a 156 mV. Esto significa que el ruido de voltaje baja bastante, lo que permite que el procesador trabaje de forma más estable y eficiente.

Ciclo De Instrucciones

¿Qué es el ciclo de instrucciones?

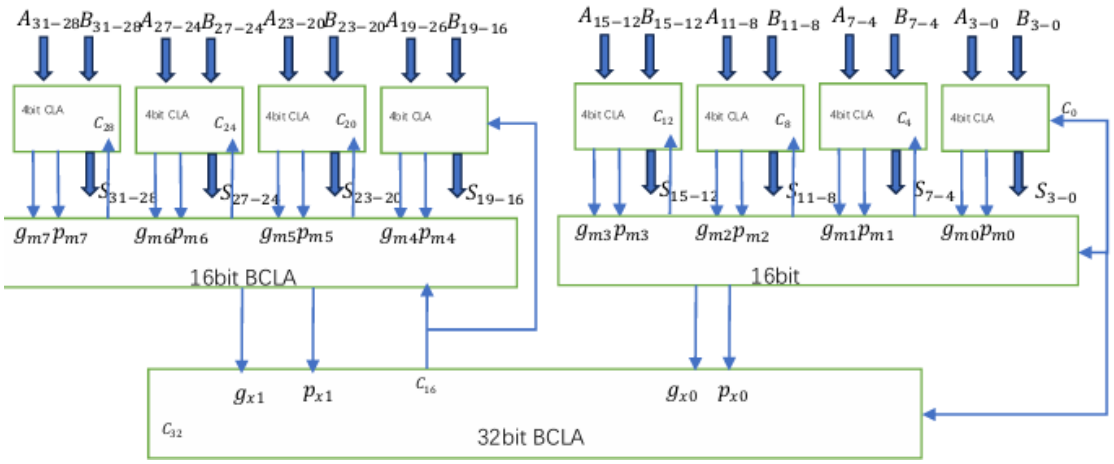
Según L. Deng, (). En el libro: “Design a 5-stage pipeline RISC-V CPU and optimise its ALU,” Applied and Computational Engineering” nos indica los siguiente acerca del ciclo de instrucciones [10].

El ciclo de instrucciones en un procesador es un proceso fundamental que permite que las computadoras puedan ejecutar diferentes tareas de manera ordenada y eficiente. Esto se logra gracias a una técnica llamada pipeline, que divide la ejecución de las instrucciones en varias etapas específicas que trabajan en paralelo para mejorar el rendimiento del CPU.

Estas etapas son generalmente cinco. La primera es fetching o recuperación de la instrucción, donde se obtiene la instrucción del programa. Luego está decoding o decodificación, en la cual el procesador interpreta qué debe hacer esa instrucción.

Después sigue la etapa de executing o ejecución, en la que se realiza la operación requerida, ya sea aritmética, lógica o de otro tipo. Posteriormente está la etapa de accessing memory o acceso a memoria, donde se pueden leer o escribir datos en la memoria. Finalmente, la etapa de writing back o escritura de resultados en registros, donde se almacenan los resultados finales de la operación.

Esto es importante porque, gracias a este diseño en pipeline, cada una de estas etapas puede realizar su tarea en cada ciclo del reloj, lo que reduce el tiempo total de ejecución de las instrucciones y permite que la CPU tenga una mayor velocidad y eficiencia en el procesamiento de tareas.



Etapas del ciclo de instrucciones.

M. Perotti et al. (2022) en el libro: “A ‘New Ara’ for Vector Computing: An Open Source Highly Efficient RISC-V V 1.0 Vector Processor Design, nos indica lo siguiente sobre las etapas ciclo de instrucciones [11].

El ciclo de instrucciones en un procesador se puede entender mejor si analizamos sus diferentes etapas, que se conectan entre sí para que el CPU funcione de manera eficiente. Estas etapas permiten que cada instrucción pase por todo el proceso de ejecución en varias fases específicas que trabajan en conjunto.

En primer lugar, está la etapa de *fetch* o recuperación, en donde el procesador obtiene la instrucción de la memoria principal. Esto se realiza a través del contador de programa, que indica la dirección de la siguiente instrucción a ejecutar.

Después sigue la etapa de decoding o decodificación en la cual el procesador interpreta qué tipo de instrucción es y qué recursos necesita para realizarla y esto es fundamental para saber si debe acceder a registros, a la memoria o realizar operaciones aritméticas y después de la decodificación viene la etapa de execute o ejecución en la que se llevan a cabo las operaciones propiamente dichas ya sean sumas restas multiplicaciones etcétera y esto se realiza con componentes como la ALU que es el módulo responsable de las operaciones aritméticas y lógicas y posteriormente en la etapa de memory o memoria el procesador realiza lecturas o escrituras en la memoria de datos si la instrucción lo requiere como en los casos de cargas o almacenamientos.

Finalmente es la etapa de write-back o escritura de resultados en los registros donde se almacenan los resultados finales para que puedan ser utilizados en futuras instrucciones y todos estos pasos se repiten en ciclo, permitiendo que las computadoras puedan procesar instrucciones de manera continua y rápida lo cual es esencial para el buen rendimiento de un CPU [10].

Ejemplo práctico.

Según R. Giorgi, (2022). En el artículo: *"FREESS: An Educational Simulator of a RISC-V-Inspired Superscalar Processor Based on Tomasulo's Algorithm"* [12], se presentan varios ejemplos prácticos que ayudan a ilustrar cómo funciona la simulación en FREESS y cómo los conceptos de procesamiento en paralelo se pueden analizar en un contexto educativo.

Por ejemplo, en uno de los casos se muestra un programa que realiza una suma vectorial, y se describe cómo el simulador visualiza en cada ciclo las etapas en las que se encuentran las instrucciones, incluyendo sus dependencias y posibles cuellos de botella.

Mientras que en otro ejemplo se trabaja con un programa más reducido y con menor ancho de banda del procesador, lo que demuestra cómo la limitación en el número de instrucciones que se pueden procesar en paralelo afecta directamente la eficiencia del *pipeline* y el IPC alcanzado.

Estos ejemplos son fundamentales porque permiten a los estudiantes comprender el impacto de diferentes parámetros arquitectónicos en el rendimiento del procesador y

facilitar la visualización de conceptos complejos como las dependencias en datos y las restricciones estructurales.

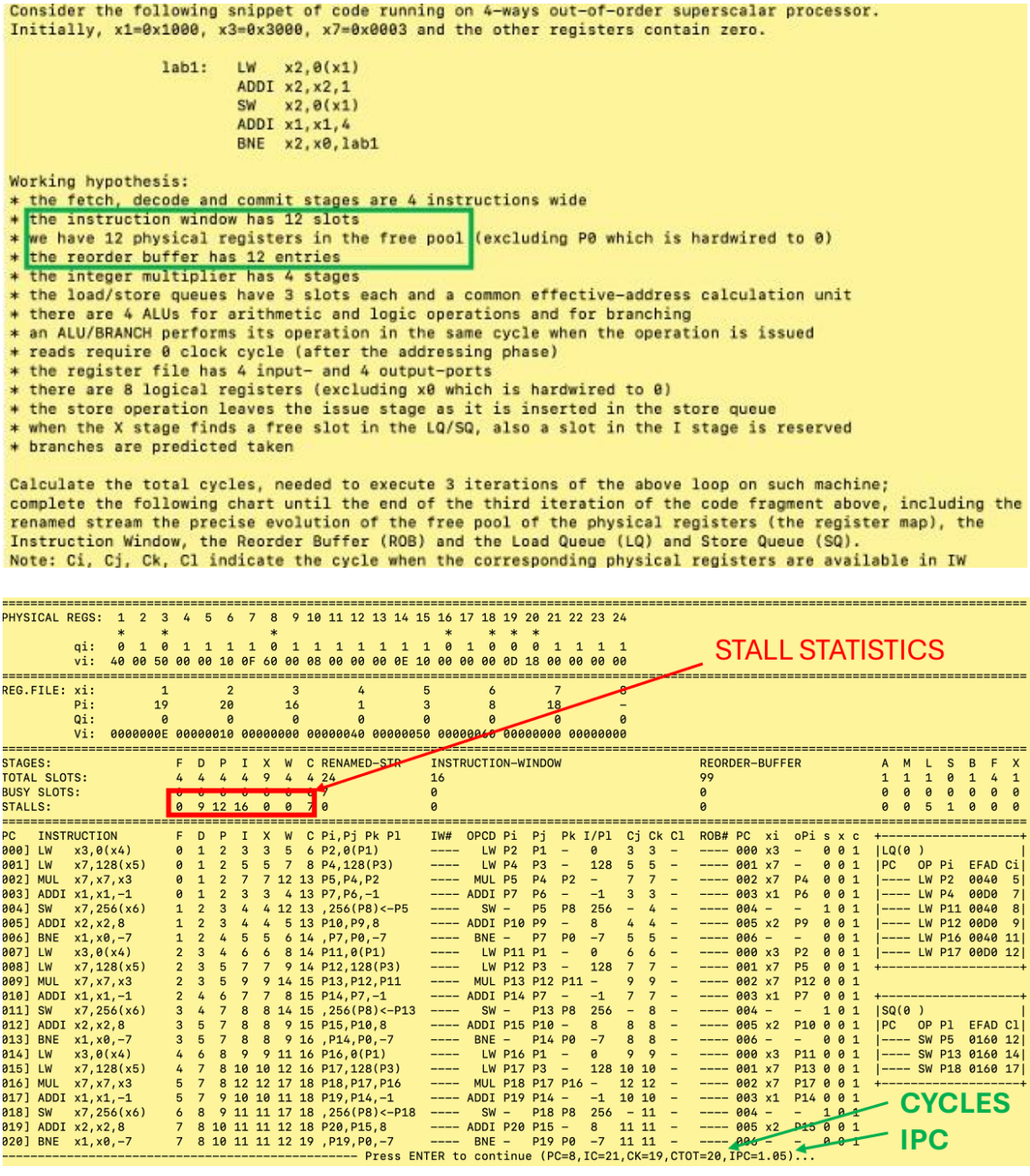


Figura 4 Parámetros arquitectónicos en el rendimiento del procesador

Importancia del ciclo de instrucciones.

Según A. Diavastos y T. E. Carlson (2021), en el artículo: *"Efficient Instruction Scheduling using Real-time Load Delay Tracking"* [13], se propone una técnica de programación de instrucciones que mejora la eficiencia del *pipeline* mediante el rastreo en tiempo real de los retardos causados por cargas.

El ciclo de instrucciones es muy importante en la arquitectura de los procesadores porque permite entender cómo se ejecutan varias instrucciones de manera eficiente en diferentes etapas del pipeline. Esto es clave para maximizar el rendimiento del sistema, ya que al analizar cómo pasa cada instrucción por fases como fetch, decodificación y ejecución, los ingenieros y estudiantes pueden detectar posibles fallas o cuellos de botella en el proceso.

Además, esto ayuda a mejorar el diseño del procesador y a implementar técnicas como la ejecución fuera de orden, que permite procesar varias instrucciones en paralelo y aprovechar mejor los recursos disponibles. Por otra parte, comprender el ciclo de instrucciones también ayuda a gestionar dependencias entre instrucciones, que pueden causar retardos o stalls en el pipeline, y por eso es fundamental en el desarrollo de procesadores modernos para mejorar su eficiencia y rendimiento.

En conclusión, el conocimiento del ciclo de instrucciones no solo es esencial para diseñadores y programadores, sino que también es la base para entender cómo funcionan los procesadores y cómo se pueden optimizar para lograr una mayor velocidad y eficiencia en el procesamiento de datos. Esto, en el contexto de este documento, se relaciona directamente con el uso de simuladores educativos como FREESS, que ayudan a visualizar y entender mejor este proceso.

Procedimientos

Procedimientos

Operaciones de Aritmética Binaria y Decimal

- Realizar ejercicios de suma, resta, multiplicación y división en binario y decimal
- Comparar resultados obtenidos en ambos sistemas para evaluar la precisión

Suma binaria y decimal

$$\begin{array}{r} 125_{10} \\ +\ 63_{10} \\ \hline 188_{10} \end{array}$$

$125_{10} \rightarrow 1111101_2$

$63_{10} \rightarrow 111111_2$

$188_{10} \rightarrow 10111100_2$

$$\begin{array}{r} 1111101_2 \\ +\ 111111_2 \\ \hline 10111100_2 \end{array}$$

Resta binaria y decimal

$$\begin{array}{r} 125_{10} \\ -\ 63_{10} \\ \hline 62_{10} \end{array}$$

$125_{10} \rightarrow 1111101_2$

$63_{10} \rightarrow 111111_2$

$62_{10} \rightarrow 111110_2$

$$\begin{array}{r} 1111101_2 \\ -\ 111111_2 \\ \hline 0111110_2 \end{array}$$

Multiplicación binaria y decimal

$$\begin{array}{r} 1150_{10} \\ \times \quad 56_{10} \\ \hline 6900_{10} \\ +5750_{10} \\ \hline 64400_{10} \end{array}$$

$1150_{10} \rightarrow 10001111110_2$

$56_{10} \rightarrow 111000_2$

$64400_{10} \rightarrow 1111101110010000_2$

$$\begin{array}{r} 10001111110_2 \\ \times \quad 111000_2 \\ \hline 00000000000_2 \\ 00000000000_2 \\ 10001111110_2 \\ 10001111110_2 \\ +\ 10001111110_2 \\ \hline 1111101110010000_2 \end{array}$$

División binaria y decimal

$$\begin{array}{r|l} 3415 & 15 \\ -30 & 227 \\ \hline 41 & \\ -30 & \\ \hline 115 & \\ -105 & \\ \hline (10) & \end{array}$$

$3415_{10} \rightarrow 110101010111_2$

$15_{10} \rightarrow 1111_2$

$227_{10} \rightarrow 11100011_2$

$10_{10} \rightarrow 1010_2$

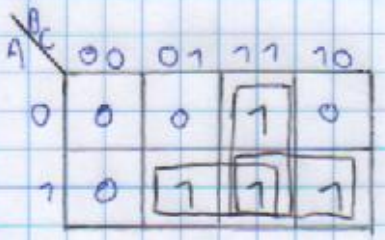
$$\begin{array}{r|l} 110101010111 & 1111_2 \\ -\ 1111 & 11100011_2 \\ \hline 010111 & \\ -\ 1111 & \\ \hline 010000 & \\ -\ 1111 & \\ \hline 000011011 & \\ -\ 1111 & \\ \hline 011001 & \\ -\ 1111 & \\ \hline (01010)_2 & \end{array}$$

Ejercicios de Algebra Booleana

- Simplificar expresiones usando mapas de Karnaugh.

1) $F_1(A,B,C) = A\bar{B}C + AB\bar{C} + ABC + \bar{A}BC$

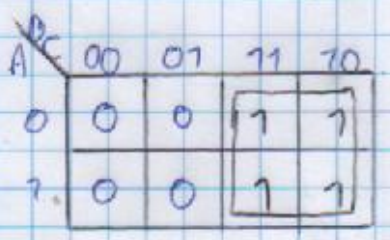
A	B	C	F_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



$F_1(A,B,C) = AC + BC + AB$

2) $F_2(A,B,C) = \bar{A}B\bar{C} + \bar{A}BC + AB\bar{C} + ABC$

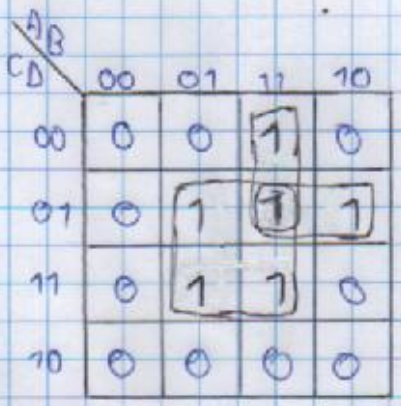
A	B	C	F_2
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



$F_2(A,B,C) = B$

3) $F_3(A,B,C,D) = \bar{A}B\bar{C}D + \bar{A}BCD + A\bar{B}\bar{C}D + ABCD + AB\bar{C}\bar{D} + AB\bar{C}D$

A	B	C	D	F_3
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1



$F_3(A,B,C,D) = BD + AB\bar{C} + A\bar{C}D$

Ejercicios de Algebra Booleana

Diseñar y probar circuitos lógicos básicos utilizando software de simulación

1) $F1(A,B,C) = A\bar{B}C + AB\bar{C} + ABC + \bar{A}BC$

   Simplificado: $F1(A,B,C) = AC + BC + AB$

La función Booleana F1(A, B, C), nos dice que será verdadera cuando A y C sean positivas, o cuando B y C sean positivas, o cuando A y B sean positivas. Entonces vamos a comprobarlo dibujando el circuito en un Software de Simulación como "Logisim Evolution".

Este es el circuito simplificado:

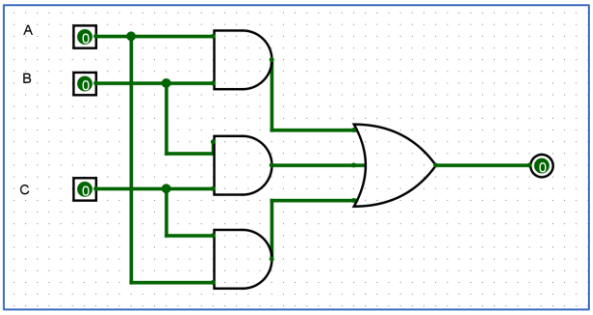


Figura 5: Circuito F1 Simplificado

Circuito cuando A y C son positivos:

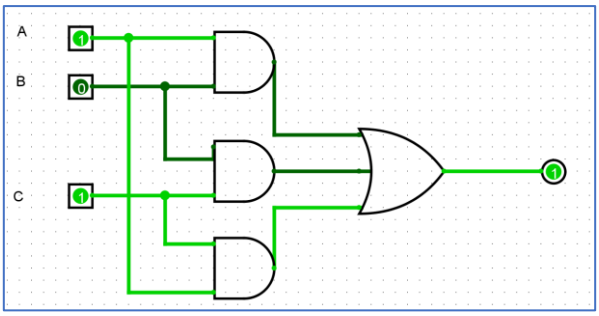


Figura 6 A y C positivos

Circuito cuando B y C son positivos:

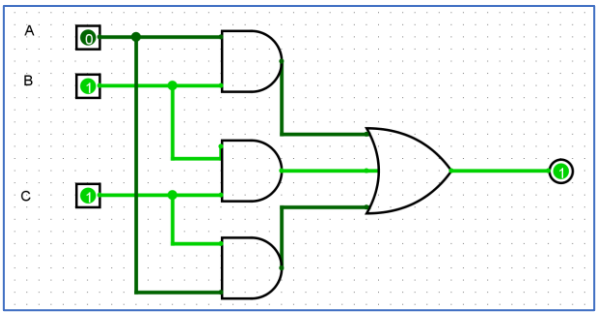


Figura 7 B y C positivos

Circuito cuando A y B son positivos:

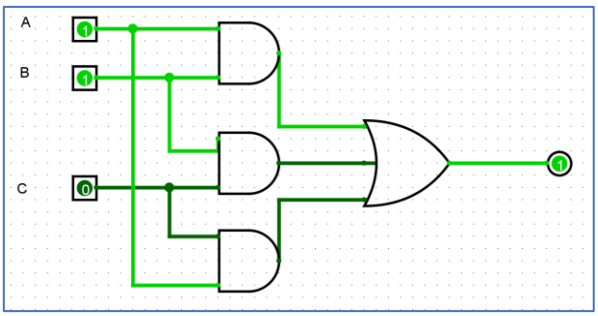


Figura 8 A y B positivos

2) $F2(A,B,C) = \bar{A}B\bar{C} + \bar{A}BC + AB\bar{C} + ABC$

Simplificado: $F1(A,B,C) = B$

La función booleana F2(A, B, C), es bastante sencilla, ya que nos indica que únicamente va a ser verdadera cuando B sea positiva. Entonces usando el mismo Software de Simulación quedaría de la siguiente forma:

Este es el circuito simplificado:

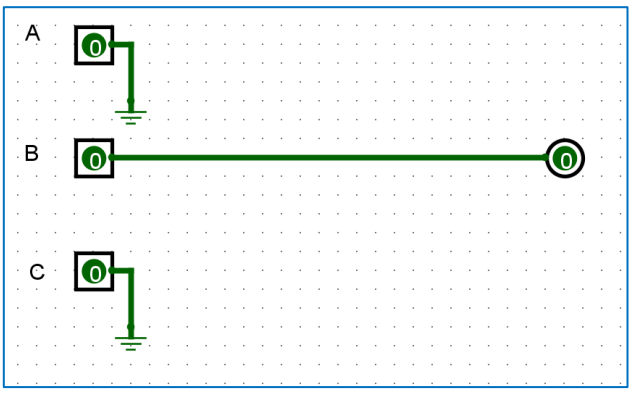


Figura 9 Circuito F2 Simplificado

Circuito cuando B es positiva:

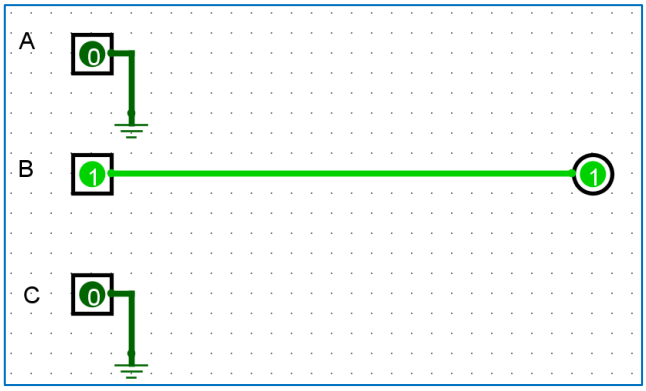


Figura 10 B positivo

3) $F3(A,B,C,D) = \bar{A}B\bar{C}D + \bar{A}BCD + A\bar{B}\bar{C}D + ABCD + AB\bar{C}\bar{D} + AB\bar{C}D$

Simplificado: $F1(A,B,C,D) = BD + AB\bar{C} + A\bar{C}D$

La función booleana F3(A, B, C, D), nos indica que solo será positiva cuando B y D sean positivas, o cuando Ay B sean positivas, y C sea negativa, o cuando A y D sean positivas, y C sea negativa. Si lo representamos en un Software de Simulación quedaría de la siguiente manera.

Este es el circuito simplificado:

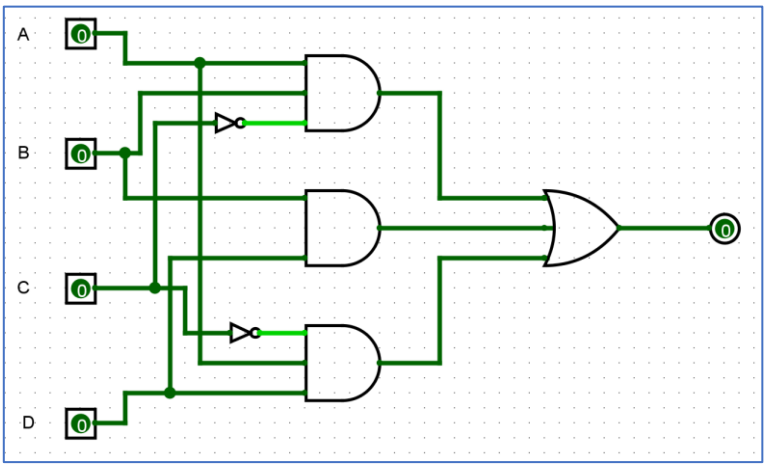


Figura 11 Circuito F3 Simplificado

Circuito cuando A y B, pero no C:

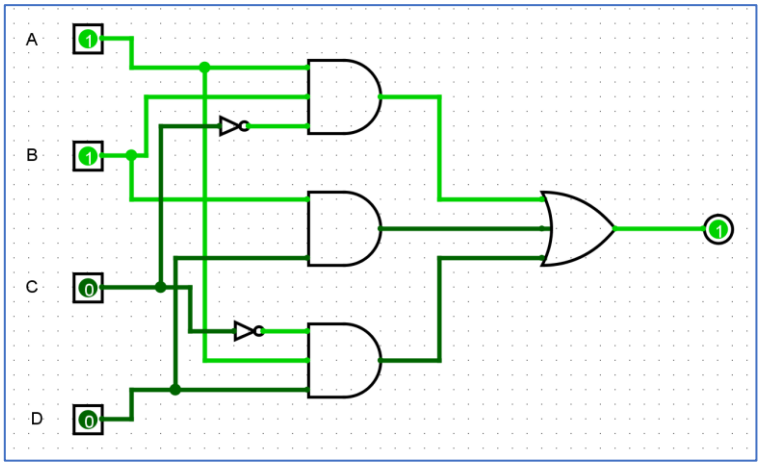


Figura 12 A y B positivas, y C negativa

Circuito cuando A y D, pero no C:

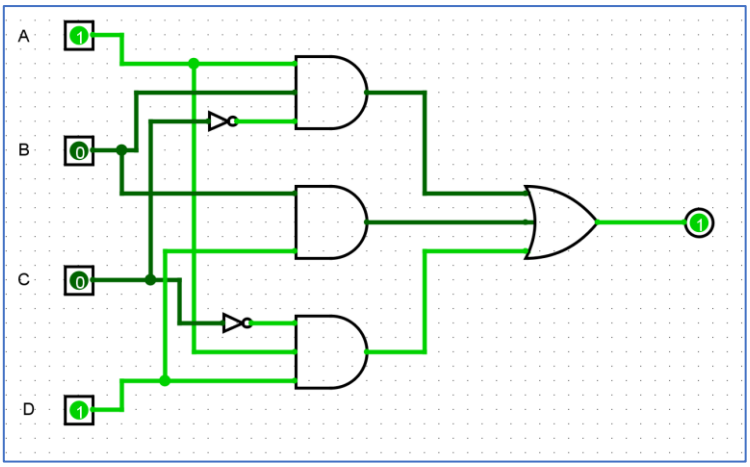


Figura 13 A y D positivas, y C negativa

Diseño de circuitos combinacionales

· Construcción de un sumador binario completo

A continuación se desarrollará un sumador completo de un bit. Este tendrá 3 entradas (A, B, C) y 2 salidas (S, F).

A	B	C	S	F
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Determinamos los valores de suma y acarreo

1 bit

$$\begin{array}{r} 0 \\ 0 \\ +\,1 \\ \hline 1 \end{array} \quad \begin{array}{r} 1 \\ 0 \\ +\,0 \\ \hline 1 \end{array} \quad \begin{array}{r} 0 \leftarrow A \\ 1 \leftarrow B \\ +\,0 \leftarrow C \\ \hline 1 \end{array}$$

2 bits

$$\begin{array}{r} 0 \\ 1 \\ +\,1 \\ \hline 10 \end{array} \quad \begin{array}{r} 1 \\ 0 \\ +\,1 \\ \hline 10 \end{array} \quad \begin{array}{r} 1 \leftarrow A \\ 1 \leftarrow B \\ +\,0 \leftarrow C \\ \hline 10 \end{array}$$

F = acarreo

3 bits

$$\begin{array}{r} 1 \leftarrow A \\ +\,1 \leftarrow B \\ \hline 10 \\ +\,1 \leftarrow C \\ \hline 11 \end{array}$$

S = Sumario

F = Acarreo

Salidas

(S)

A \ BC	00	01	11	10
0	0	1	0	1
1	1	0	1	0

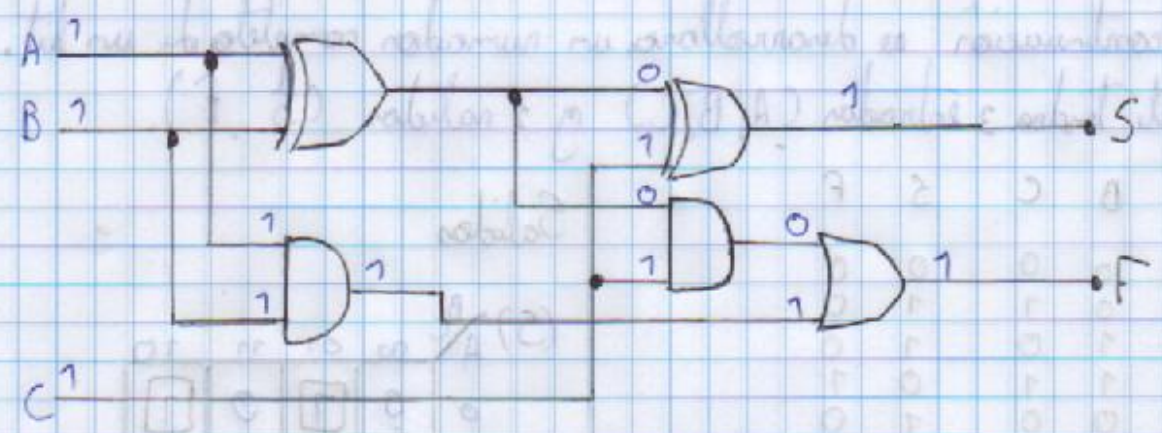
$$S = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$$
$$= \bar{A}(\bar{B}C + C\bar{B}) + A(\bar{B}\bar{C} + BC)$$
$$= \bar{A}(B \oplus C) + A\overline{(B \oplus C)}$$
$$= A \oplus (B \oplus C) = A \oplus B \oplus C$$

(F)

A \ BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$F = AB + AC + BC$$

- Representación gráfica del sumador completo.

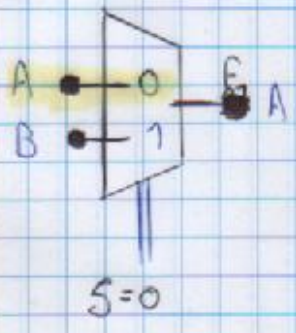


- Implementar un multiplexor simple para controlar el flujo de datos

Un multiplexor de dos entradas es como un interruptor que decide cual entrada enviar a la salida.

En un MUX 2:1 tenemos: dos entradas (A, B) y una señal de selección (S) y una salida F.

Si "S" es igual a cero, entonces elije "A", si "S" es 1, entonces elije "B"



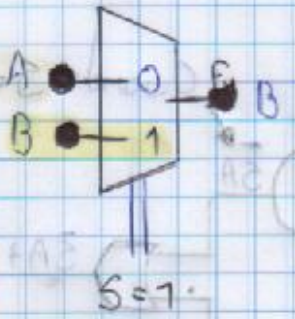


Tabla de verdad del multiplexor

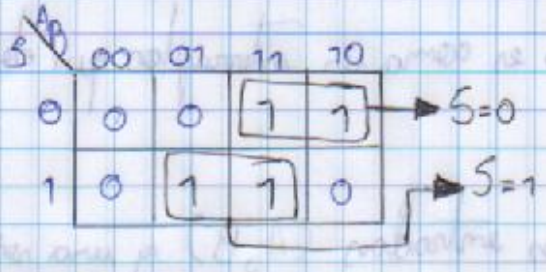
S	A	B	F
0	0	X	0
0	1	X	1
1	X	0	0
1	X	1	1

→ Si la extendemos →

S	A	B	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Si observamos la tabla de verdad extendida nos damos cuenta como toma los valores según si "S" es negativo o positivo.

Graficación del multiplexor usando Karnaugh.

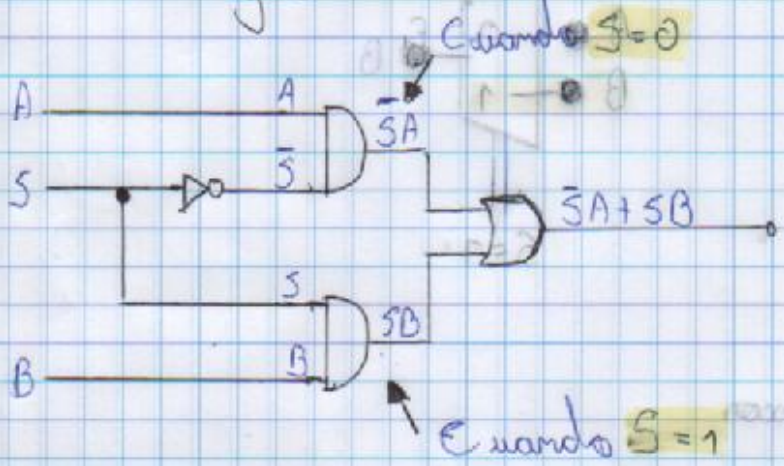


S	A	B	F_g
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

$F_1 = \bar{S}A\bar{B} + \bar{S}AB + S\bar{A}B + SAB$

$F_1 = \bar{S}A + SB$

Circuito Lógico:



Estudio de Microprocesadores y Microcontroladores

Investigar la arquitectura básica de un microprocesador y un microcontrolador

Arquitectura basica de un microprocesador

De acuerdo con Hübner y Becker (2011), los microprocesadores modernos están diseñados con múltiples núcleos, donde cada uno dispone de sus propias memorias caché L1 y L2. Sin embargo, los niveles de caché más altos son compartidos entre los núcleos utilizando protocolos distribuidos que garantizan la coherencia de los datos. Esta organización permite una transferencia eficiente de información entre los núcleos y la memoria principal, lo que contribuye significativamente a mejorar tanto el rendimiento como la capacidad de escalar en sistemas con múltiples núcleos. Un ejemplo claro de esta arquitectura se encuentra en los estudios sobre los sistemas multiprocesador en chip (MPSoC) como se observa en la figura 2, donde se detallan estos mecanismos de coherencia y distribución de memoria en entornos actuales [14].

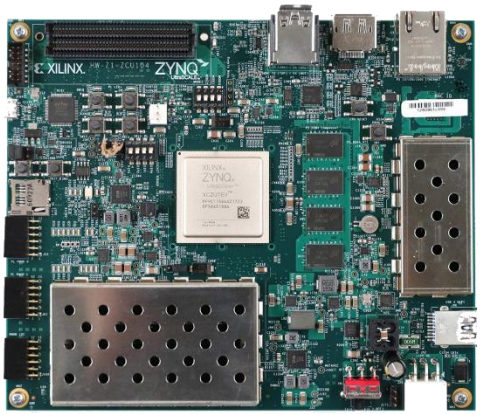


Figura 14 "MPSoC"

Arquitectura del microprocesador MPSoC

En el libro: Multiprocessor System-on-Chip, nos indica que: la arquitectura MPSoC descrita se basa en una plataforma modular compuesta por múltiples bloques de procesamiento y memoria interconectados mediante una red en chip. Las aplicaciones se dividen en tareas que se distribuyen de forma estática entre los procesadores, lo que significa que no hay migración de tareas entre núcleos [14].

Las tareas no críticas pueden comunicarse libremente a través de memoria compartida, mientras que las tareas en tiempo real siguen un patrón cíclico de lectura, procesamiento

y escritura. Estas tareas se comunican mediante colas FIFO con operaciones bloqueantes, siguiendo el protocolo C-HEAP, lo cual favorece aplicaciones de flujo continuo de datos y facilita el análisis temporal del sistema.

El acceso a recursos compartidos como procesadores, memorias o la red de comunicación se realiza mediante solicitudes, que pueden verse afectadas por interferencias debido a otros procesos. Esto impacta en el tiempo total de respuesta de cada solicitud. Para garantizar un comportamiento estable y verificable, se aplican los conceptos de componibilidad y predictibilidad.

Un sistema es componible si el comportamiento temporal de una tarea no se ve afectado por otras tareas externas, y es predecible si se pueden establecer límites claros para los peores tiempos de ejecución y respuesta (WCET y WCRT). Estas propiedades permiten diseñar sistemas más robustos, especialmente en entornos con aplicaciones en tiempo real, donde se requiere una ejecución fiable y puntual.

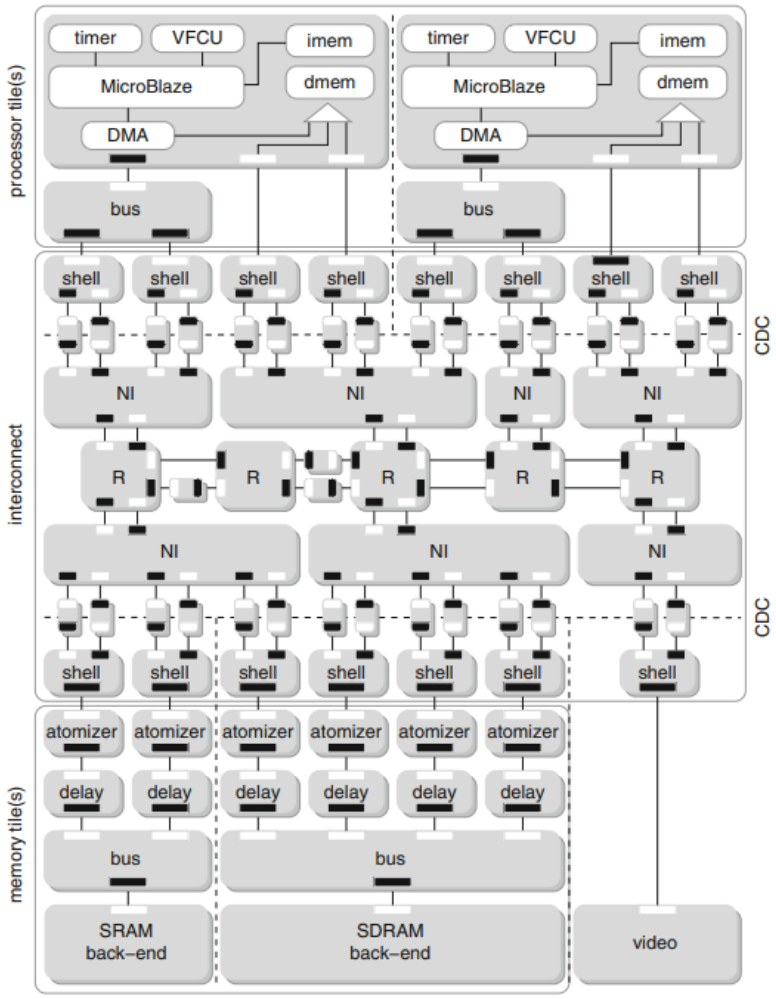


Figura 15 Arquitectura de la plataforma MPSoC

Arquitectura basica de un microcontrolador

En el libro: ICCPS ’23: Proceedings of the ACM/IEEE 14th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2023), redacta que los microcontroladores se caracterizan por integrar en un solo chip la unidad central de procesamiento, la memoria RAM, ROM, diversos periféricos de entrada y salida. Entre estos periféricos destacan los convertidores analógico-digital y módulos de temporización. Esta integración permite desarrollar soluciones muy compactas, ideales para aplicaciones en sistemas embebidos [15].

Por su parte señala que los diseños actuales incorporan técnicas de mapeo de memoria unificada, lo cual optimiza el acceso a los datos y minimiza la dependencia de interfaces externas.

MicroNas

En el artículo: MicroNAS for memory and latency constrained hardware aware neural architecture search in time series classification on microcontrollers, nos indica lo siguiente acerca de la arquitectura, basada en MPSoC, MicroNAS [16].

La arquitectura MicroNAS (Micro Network-on-Chip-based Architecture for MPSoCs) es una propuesta de diseño para sistemas multiprocesador en chip que combina múltiples núcleos de procesamiento con una red en chip altamente eficiente y ligera. Su enfoque principal es lograr una comunicación rápida y de bajo consumo entre núcleos, lo que la convierte en una solución ideal para aplicaciones embebidas que requieren eficiencia energética y alto rendimiento, como dispositivos móviles, sensores inteligentes o sistemas IoT.

A diferencia de arquitecturas tradicionales basadas en buses o interconexiones simples, MicroNAS utiliza una red personalizada y modular, lo que mejora la escalabilidad y el paralelismo del sistema. Cada procesador o componente (como memoria o periféricos) se conecta a través de esta red, permitiendo una transferencia de datos flexible y distribuida, sin cuellos de botella. La red está optimizada para minimizar latencia y consumo energético, utilizando rutas directas y simplificadas entre nodos.

Uno de los aspectos más relevantes de MicroNAS es su soporte para aplicaciones heterogéneas. Puede integrar distintos tipos de núcleos de, y está diseñada para facilitar tanto la comunicación local como global. Además, su estructura modular permite escalar

fácilmente el número de nodos sin afectar el rendimiento global, lo cual es esencial en entornos donde la demanda de procesamiento puede crecer con el tiempo .

Figura 16 MicroNAS

Analizar el ciclo de instrucciones y la velocidad del microprocesador con ejemplos prácticos

El objetivo de esta práctica es analizar el tiempo de ejecución de un conjunto de instrucciones utilizando un microprocesador (ESP32), midiendo su velocidad a través del uso de la función micros() y observando los resultados en el Monitor Serie del entorno de desarrollo Arduino IDE.

Para desarrollar este ejemplo práctico vamos a necesitar los siguientes materiales:

- Arduino ESP32 (Puede ser cualquier modelo).
- Cable USB para realizar la conexión con el Arduino.
- Nuestro computador con la aplicación Arduino IDE.

Procedimiento de la practica

En el Arduino IDE seleccionamos nuestro modelo de placa, en mi caso ESP32 DEV MODULE, el puerto COM al que conectamos nuestro Arduino, y pegamos el siguiente código:

```
unsigned long startTime;
unsigned long endTime;
volatile long resultado = 0;
void setup()
{
     Serial.begin(115200);
     startTime = micros();
     for (long i = 0; i < 1000; i++)
     {
```

```
            resultado = resultado + i;
        }
        endTime = micros();
        Serial.print("Tiempo de ejecución: ");
        Serial.print(endTime - startTime);
        Serial.println(" microsegundos");
        Serial.print("Resultado final: ");
        Serial.println(resultado);
}
void loop()
{
}
```

Explicación del código

En el código declaramos una variable startTime, la cual se encargará de almacenar el tiempo, antes de comenzar el bloque de instrucciones, en microsegundos. En este caso el bloque de instrucciones es un ciclo que realiza 1000 sumas.

Luego, para guardar el tiempo después de la ejecución de nuestro bloque de instrucciones, se declara una variable endTime, la cual se encarga de detener el tiempo después de completar nuestra instrucción.

Además, declaramos una variable volátil resultado, la cual se encargará de mostrar el resultado final de la suma, esto nos sirve para que nuestro Arduino ESP32 no se salte ninguna iteración.

Al final, para calcular el tiempo total se imprime entTime-startTime, de esta forma podemos calcular cuánto se tarda nuestro Arduino ESP32 en hacer 1000 sumas al momento de ejecutarse.

El tiempo total de la ejecución de nuestro bloque de instrucciones se mostrará a través del Monitor Serie.

Una vez cargado el código en nuestro Arduino, procedemos a proporcionar energía para ver cuál es el tiempo total de la ejecución del ciclo de instrucción.

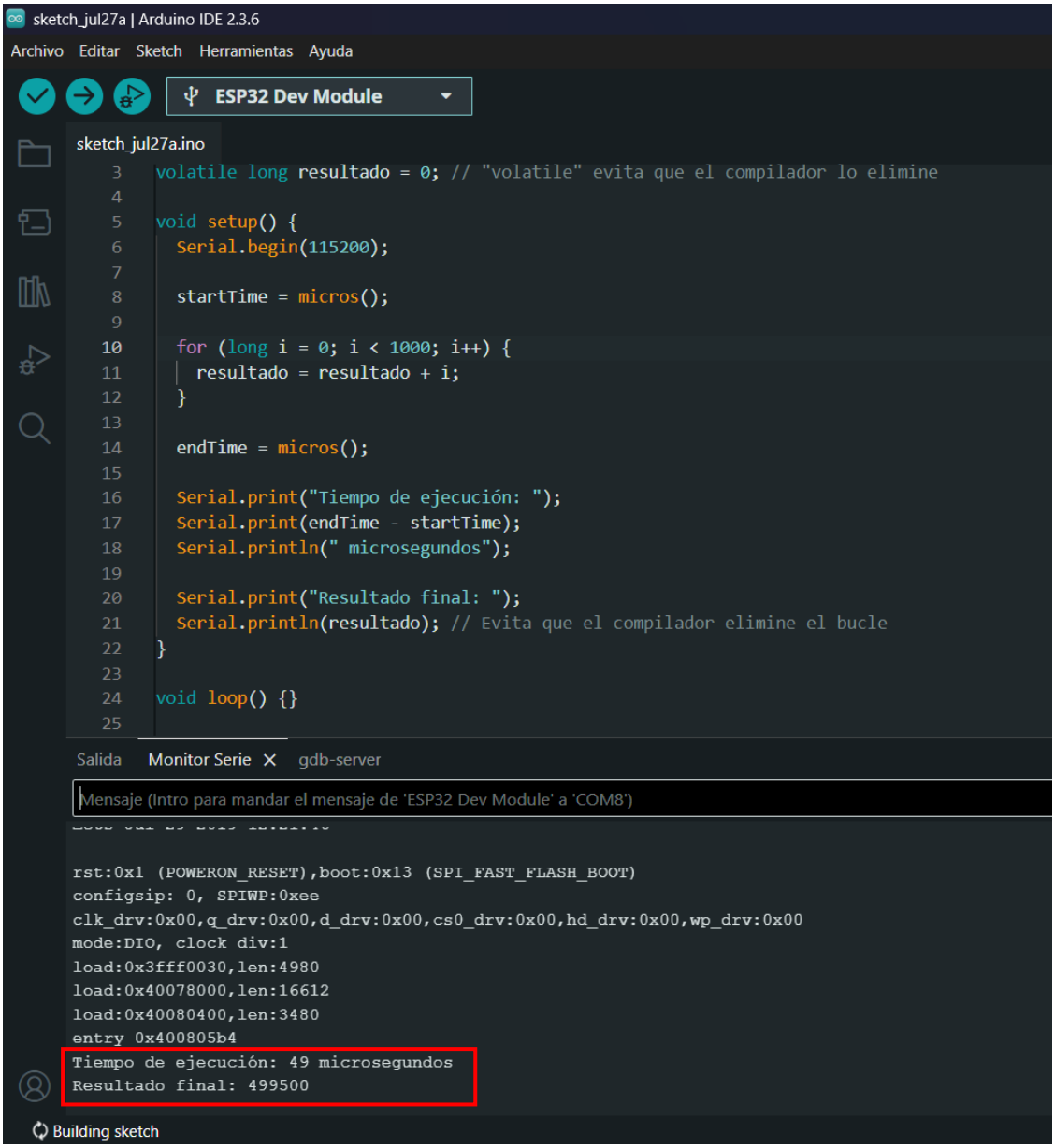


Figura 17 Ejecución del ciclo de instrucciones en el Arduino IDE

Conclusión acerca del tiempo de ejecución

Como podemos observar en la figura , tenemos un tiempo de ejecución de 49 microsegundos y un resultado de 499500, es decir, nuestro Arduino realizo 1000 sumas en 0.000049 segundos dando como resultado 499500. Esto se debe a que el Arduino ESP32 tiene un procesador Dual-Core de 240MHz, el cual es muy rápido.

Bibliografía

[1] Philippe Darche, “Commercial Microprocessors: From a Single Bit to 128 Bits,” in *Microprocessor 3*, Wiley, 2020, pp. 81–101. doi: 10.1002/9781119788010.ch4.

[2] Philippe Darche, “Microprocessor Interfacing,” in *Microprocessor 3*, Wiley, 2020, pp. 17–38. doi: 10.1002/9781119788010.ch2.

[3] Runde Zhou, “The trend of the microprocessor design toward the year of 2000,” in *Proceedings of 4th International Conference on Solid-State and IC Technology*, IEEE, pp. 685–687. doi: 10.1109/ICSICT.1995.503390.

[4] K.-D. Kramer, T. Stolze, and T. Banse, “Benchmarks to Find the Optimal Microcontroller-Architecture,” in *2009 WRI World Congress on Computer Science and Information Engineering*, IEEE, 2009, pp. 102–105. doi: 10.1109/CSIE.2009.928.

[5] R. Cheour, S. Khriji, M. abid, and O. Kanoun, “Microcontrollers for IoT: Optimizations, Computing Paradigms, and Future Directions,” in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, IEEE, Jun. 2020, pp. 1–7. doi: 10.1109/WF-IoT48130.2020.9221219.

[6] K. Killpack, S. Natarajan, A. Krishnamachary, and P. Bastani, “Case Study on Speed Failure Causes in a Microprocessor,” *IEEE Design & Test of Computers*, vol. 25, no. 3, pp. 224–230, May 2020, doi: 10.1109/MDT.2008.61.

[7] A. Neftissov, I. Kazambayev, L. Kirichenko, A. Aubakirova, D. Urazayev, and K. Zhakupova, “Development of microprocessor device of relay protection based on open architecture using Industrial Internet of Things technology,” in *Procedia Computer Science*, Elsevier B.V., 2024, pp. 672–677. doi: 10.1016/j.procs.2023.12.163.

[8] S. Meloth, “A case study on radiated emission in a high speed microprocessor based design,” in *2016 International Conference on ElectroMagnetic Interference & Compatibility (INCEMIC)*, IEEE, Dec. 2019, pp. 1–2. doi: 10.1109/INCEMIC.2016.7921500.

[9] H. Sanchez *et al.*, “Increasing Microprocessor Speed by Massive Application of On-Die High-K MIM Decoupling Capacitors,” in *2006 IEEE International Solid*

State Circuits Conference - Digest of Technical Papers, IEEE, 2006, pp. 2190–2199. doi: 10.1109/ISSCC.2006.1696280.

[10] L. Deng, "Design a 5-stage pipeline RISC-V CPU and optimise its ALU," *Applied and Computational Engineering*, vol. 34, no. 1, pp. 237–244, Feb. 2024, doi: 10.54254/2755-2721/34/20230334.

[11] M. Perotti, M. Cavalcante, N. Wistoff, R. Andri, L. Cavigelli, and L. Benini, "A 'New Ara' for Vector Computing: An Open Source Highly Efficient RISC-V V 1.0 Vector Processor Design," in *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors*, Institute of Electrical and Electronics Engineers Inc., 2022, pp. 43–51. doi: 10.1109/ASAP54787.2022.00017.

[12] R. Giorgi, "FREESS: An Educational Simulator of a RISC-V-Inspired Superscalar Processor Based on Tomasulo's Algorithm," Jun. 2025, [Online]. Available: http://arxiv.org/abs/2506.07665

[13] A. Diavastos and T. E. Carlson, "Efficient Instruction Scheduling using Real-time Load Delay Tracking," Sep. 2021, [Online]. Available: http://arxiv.org/abs/2109.03112

[14] M. Hübner and J. Becker, Eds., *Multiprocessor System-on-Chip*. New York, NY: Springer New York, 2011. doi: 10.1007/978-1-4419-6460-1.

[15] Sayan Mitra, Nalini Venkatasubramanian, Abhishek Dubey, Lu Feng, Mahsa Ghasemi, and Jonathan Sprinkle, "ICCPS '23: Proceedings of the ACM/IEEE 14th International Conference on Cyber-Physical Systems (with CPS-IoT Week 2023)," New York, NY, USA: Association for Computing Machinery, 2023. Accessed: Jul. 26, 2025. [Online]. Available: https://dl.acm.org/doi/proceedings/10.1145/3576841

[16] T. King, Y. Zhou, T. Röddiger, and M. Beigl, "MicroNAS for memory and latency constrained hardware aware neural architecture search in time series classification on microcontrollers," *Sci Rep*, vol. 15, no. 1, Dec. 2025, doi: 10.1038/s41598-025-90764-z.

Anexos

https://github.com/MiloSaurio4kHD/ARQCOM_GC_