



Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Eléctrica

**INTRODUCCIÓN DEL LENGUAJE ENSAMBLADOR PARA LA ARQUITECTURA
ARM UTILIZANDO EL ENTORNO DE DESARROLLO KEIL UVISION4, BAJO EL MODELO
CONSTRUCTIVISTA DE EDUCACIÓN, COMO VIRTUALIZACIÓN DE LOS TEMAS
IMPARTIDOS EN EL CURSO DE ELECTRÓNICA 5 EN LA ESCUELA DE INGENIERÍA
MECÁNICA ELÉCTRICA, FACULTAD DE INGENIERÍA, UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA**

Miguel Bernabé Tavico Laynez

Asesorado por la Inga. Ingrid Rodríguez de Loukota

Guatemala, octubre de 2021

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA



FACULTAD DE INGENIERÍA

**INTRODUCCIÓN DEL LENGUAJE ENSAMBLADOR PARA LA ARQUITECTURA ARM
UTILIZANDO EL ENTORNO DE DESARROLLO KEIL UVISION4, BAJO EL MODELO
CONSTRUCTIVISTA DE EDUCACIÓN, COMO VIRTUALIZACIÓN DE LOS TEMAS
IMPARTIDOS EN EL CURSO DE ELECTRÓNICA 5 EN LA ESCUELA DE INGENIERÍA
MECÁNICA ELÉCTRICA, FACULTAD DE INGENIERÍA, UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA**

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

MIGUEL BERNABÉ TAVICO LAYNEZ

ASESORADO POR LA INGA. INGRID SALOMÉ RODRÍGUEZ DE LOUKOTA

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO ELECTRÓNICO

GUATEMALA, OCTUBRE DE 2021

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA



NÓMINA DE JUNTA DIRECTIVA

DECANA	Inga. Aurelia Anabela Cordova Estrada
VOCAL I	Ing. José Francisco Gómez Rivera
VOCAL II	Ing. Mario Renato Escobedo Martínez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Br. Kevin Vladimir Armando Cruz Lorente
VOCAL V	Br. Fernando José Paz González
SECRETARIO	Ing. Hugo Humberto Rivera Pérez

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Pedro Antonio Aguilar Polanco
EXAMINADOR	Ing. Carlos Eduardo Guzmán Salazar
EXAMINADOR	Ing. Byron Odilio Arrivillaga Méndez
EXAMINADOR	Ing. Armando Alonso Rivera Carrillo
SECRETARIA	Inga. Lesbia Magalí Herrera López

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

**INTRODUCCIÓN DEL LENGUAJE ENSAMBLADOR PARA LA ARQUITECTURA ARM
UTILIZANDO EL ENTORNO DE DESARROLLO KEIL UVISION4, BAJO EL MODELO
CONSTRUCTIVISTA DE EDUCACIÓN, COMO VIRTUALIZACIÓN DE LOS TEMAS
IMPARTIDOS EN EL CURSO DE ELECTRÓNICA 5 EN LA ESCUELA DE INGENIERÍA
MECÁNICA ELÉCTRICA, FACULTAD DE INGENIERÍA, UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA**

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, con fecha 03 de junio de 2019.

Miguel Bernabé Tavico Laynez

Guatemala 15 de junio 2021

Ingeniero
Julio César Solares Peñate
Coordinador del Área de Electrónica
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería, USAC.

Apreciable Ingeniero Solares,

Me permito dar aprobación al trabajo de graduación titulado **"Introducción del lenguaje ensamblador para la arquitectura ARM utilizando el entorno de desarrollo Keil uVision4, bajo el modelo constructivista de educación, como virtualización de los temas impartidos en el curso de Electrónica 5 en la Escuela de Ingeniería Mecánica Eléctrica, Facultad de Ingeniería, Universidad de San Carlos de Guatemala"**, del señor **Miguel Bernabé Tavico Laynez**, por considerar que cumple con los requisitos establecidos.

Por tanto, el autor de este trabajo de graduación y, yo, como su asesora, nos hacemos responsables por el contenido y conclusiones de este.

Sin otro particular, me es grato saludarle.

Atentamente,



Inga. Ingrid Rodríguez de Loukota
Colegiada 5,356
Asesora

**Ingrid Rodríguez de Loukota
Ingeniera en Electrónica
colegiada 5356**



FACULTAD DE INGENIERIA

Guatemala, 20 de julio de 2021

Señor director
Armando Alonso Rivera Carrillo
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería, USAC

Estimado Señor director:

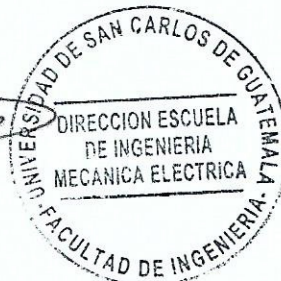
Por este medio me permito dar aprobación al Trabajo de Graduación titulado: **INTRODUCCIÓN DEL LENGUAJE ENSAMBLADOR PARA LA ARQUITECTURA ARM UTILIZANDO EL ENTORNO DE DESARROLLO KEIL UVISION4, BAJO EL MODELO CONSTRUCTIVISTA DE EDUCACIÓN, COMO VIRTUALIZACIÓN DE LOS TEMAS IMPARTIDOS EN EL CURSO DE ELECTRÓNICA 5 EN LA ESCUELA DE INGENIERÍA MECÁNICA ELÉCTRICA, FACULTAD DE INGENIERÍA, UNIVERSIDAD DE SAN CARLOS DE GUATEMALA**, desarrollado por el estudiante **Miguel Bernabé Tavico Laynez**, ya que considero que cumple con los requisitos establecidos.

Sin otro particular, aprovecho la oportunidad para saludarlo.

Atentamente,

ID Y ENSEÑAD A TODOS

Ing. Julio César Solares Peñate
Coordinador de Electrónica





REF. EIME 116. 2021.

El Director de la Escuela de Ingeniería Mecánica Eléctrica, después de conocer el dictamen del Asesor, con el Visto Bueno del Coordinador de Área, al trabajo de Graduación del estudiante; MIGUEL BERNABÉ TAVICO LAYNEZ titulado: INTRODUCCIÓN DEL LENGUAJE ENSAMBLADOR PARA LA ARQUITECTURA ARM UTILIZANDO EL ENTORNO DE DESARROLLO KEIL uVISION4, BAJO EL MODELO CONSTRUCTIVISTA DE EDUCACIÓN, COMO VIRTUALIZACIÓN DE LOS TEMAS IMPARTIDOS EN EL CURSO DE ELECTRÓNICA 5 EN LA ESCUELA DE INGENIERÍA MECÁNICA ELÉCTRICA, FACULTAD DE INGENIERÍA, UNIVERSIDAD DE SAN CARLOS DE GUATEMALA, procede a la autorización del mismo.


Ing. Armando Alonso Rivera Carrillo



GUATEMALA, 12 DE AGOSTO 2,021.



USAC
TRICENTENARIA
Universidad de San Carlos de Guatemala

Decanato
Facultad de Ingeniería
24189101 - 24189102
secretariadecanato@ingenieria.usac.edu.gt

DTG. 545-2021

La Decana de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Mecánica Eléctrica, al Trabajo de Graduación titulado: **INTRODUCCIÓN DEL LENGUAJE ENSAMBLADOR PARA LA ARQUITECTURA ARM UTILIZANDO EL ENTORNO DE DESARROLLO KEIL UVISION4, BAJO EL MODELO CONSTRUCTIVISTA DE EDUCACIÓN, COMO VIRTUALIZACIÓN DE LOS TEMAS IMPARTIDOS EN EL CURSO DE ELECTRÓNICA 5 EN LA ESCUELA DE INGENIERÍA MECÁNICA ELÉCTRICA, FACULTAD DE INGENIERÍA, UNIVERSIDAD DE SAN CARLOS DE GUATEMALA**, presentado por el estudiante universitario: **Miguel Bernabé Tavico Laynez**, y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

Inga. Anabela Cordova Estrada
Decana



Guatemala, octubre de 2021

AACE/cc

ACTO QUE DEDICO A:

Dios	Por darme la fortaleza y sabiduría durante toda mi carrera.
Mis padres	Antonia Laynez y Bernabé Tavico por ser los pilares de mi vida.
Mis hermanas	Dra. Ingrid, Dra. Marycruz y Kimberly Tavico por su apoyo.
Mis abuelos	Por sus sabios consejos, aprecio y cariño
Mi familia	Especialmente a las que me apoyaron y por decisión de la vida ya no están.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por abrirme las puertas y brindarme una educación de alta calidad.
Facultad de Ingeniería	Por brindarme todos los conocimientos necesarios para ejercer como profesional.
Mis amigos de la Facultad	Fabiola España y Manuel Fernández por su apoyo incondicional durante y al final de la carrera.
Mis amigos de la Facultad	Lesther Meoño, Luis Álvarez, José Monroy, Jonathan Medina, Steve Contreras, Ricardo Ball, Elizabeth Lux, Estuardo Chirix, Kevin Ayrton, Ronald Sandoval, Kelvin Garcia, Fernando Reyes, Luis Estrada, Luis Herrera y Esvin Paredes, por su apoyo durante la carrera.
Mis amigos de la Facultad	Luis Calderón, Rafa Toj, Dennis Fuentes, Kevin Franco, Victor Tórtola, Andrés Días, Juan Silva y Betuel Flores por su apoyo al inicio y durante la carrera.
Otros	Personas que por decisión de la vida ya no están presentes y otras por decisión propia.

Miguel Tavico

Por tener la fortaleza y perseverancia para culminar esta etapa de la vida.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES	IX
LISTA DE SÍMBOLOS	XVII
GLOSARIO	XIX
RESUMEN.....	XXIII
OBJETIVOS.....	XXV
INTRODUCCIÓN	XXVII
1. APRENDIZAJE EN LA EDUCACIÓN.....	1
1.1. La educación	1
1.1.1. Enseñanza y Aprendizaje	3
1.1.1.1. Teorías del aprendizaje	3
1.2. Precursores de las teorías modernas del aprendizaje.....	5
1.2.1. Teoría del aprendizaje	5
1.2.2. Racionalismo	6
1.2.3. Empirismo.....	6
1.3. El constructivismo en la educación como método de enseñanza.....	7
1.3.1. Jean Piaget.....	7
1.3.1.1. El origen del pensamiento humano	8
1.3.2. Lev Vygotsky	8
1.3.2.1. Método de enseñanza	9
1.3.3. David Ausubel.....	10
1.3.4. ¿Qué es el constructivismo?.....	10
1.4. Aspectos básicos según Piaget, Vygotsky y Ausubel.....	11
1.5. Ambientes de aprendizaje constructivistas.....	12

1.6.	La enseñanza dentro del constructivismo	14
1.6.1.	Enseñanza por el descubrimiento	14
1.6.2.	Enseñanza por indagación	14
1.6.3.	Aprendizaje asistido por los pares.....	15
1.6.4.	Aprendizaje cooperativo	15
1.7.	Evaluar el aprendizaje.....	15
1.7.1.	Observación directa	16
1.7.2.	Examen escrito.....	17
1.7.3.	Exámenes Orales.....	17
1.7.4.	Calificación de terceros	18
1.7.5.	Autoevaluaciones	18
1.8.	Tecnologías de la información y la comunicación (TIC)	19
1.8.1.	E-learning	21
1.8.1.1.	Comunicación Síncrona	22
1.9.	Herramientas usadas con la comunicación síncrona	22
1.9.1.	Chat.....	22
1.9.2.	Webinars	23
1.9.3.	Videollamadas.....	23
1.10.	Comunicación Asíncrona	23
2.	KEIL® MDK UTILIZANDO LENGUAJE ENSAMBLADOR PARA PROCESADORES ARM® CORTEX®-M4.....	25
2.1.	SoC (System on a Chip).....	28
2.2.	Procesador y Microprocesador	29
2.2.1.	Microprocesador.....	29
2.3.	Endianness	31
2.3.1.	Big-Endian.....	31
2.3.2.	Little-Endian	31
2.4.	Rango de direcciones para Cortex-M4.....	32

2.5.	Arquitectura de los procesadores	35
2.5.1.	RISC	37
2.5.2.	CISC	37
2.6.	Von Neumann & Harvard.....	38
2.6.1.	Arquitectura Von Neumman.....	39
2.6.2.	Arquitectura Harvard.....	40
2.6.3.	Arquitectura Harvard modificada	40
2.7.	La Familia Cortex	41
2.7.1.	Cortex A.....	42
2.7.2.	Cortex R	42
2.7.3.	Cortex M	43
2.8.	Cortex M4	45
2.8.1.	Instrucciones DSP	48
2.8.2.	Unidad de punto flotante.....	49
2.9.	Lenguaje de bajo nivel.....	50
2.9.1.	Bits a Comandos	50
2.10.	Lenguaje ensamblador	52
2.10.1.	Jerarquía de una computadora.....	52
2.11.	Arquitectura y lenguaje ensamblador	55
2.11.1.	Sistema numérico	55
2.12.	Sistema de numeración	56
2.12.1.	Sistema de numeración decimal.....	56
2.12.2.	Sistema de numeración binario	57
2.12.3.	Sistema de numeración hexadecimal	58
2.13.	Software de desarrollo para microcontroladores basados en ARM.....	59
2.13.1.	Keil uVision®	61
2.13.2.	Componentes básicos de Keil uVision®	62
2.13.3.	Instalación del software de desarrollo Keil® MDK ..	64

2.13.4.	Instalación de Keil Uvision®4	65
2.13.5.	Instalación de Keil Uvision®5	70
2.13.6.	Instalación de los paquetes para el procesador Cortex®M TM4C123GH6PM.....	74
2.13.7.	Instalación de drivers	77
2.13.8.	Instalación de Debug Stellaris® ICDI	85
3.	PROGRAMACIÓN EN LOS PROCESADORES CORTEX-M UTILIZANDO ENSAMBLADOR	89
3.1.	Registros	89
3.1.1.	Stack Memory	90
3.1.2.	R0-R12	91
3.1.3.	R13, Stack Pointer (SP)	91
3.1.4.	R14, Link Register (LR)	92
3.1.5.	R15, Program Counter (PC)	93
3.1.6.	Registros especiales	93
3.1.7.	Registro de estado del programa	93
3.1.8.	Principios básicos en ensamblador	96
3.1.8.1.	Sintaxis usada en lenguaje ensamblado.....	96
3.1.8.2.	Constantes	97
3.1.8.3.	Insertar valores en el programa.....	98
3.1.8.4.	Datos en programa.....	100
3.1.8.5.	Directivas.....	101
3.1.9.	Lenguaje ensamblador unificado o en inglés UAL.	105
3.1.10.	Set de instrucciones	106
3.1.10.1.	Mover datos dentro del procesador....	107

3.1.10.2.	Instrucciones de acceso a la memoria.....	110
3.1.10.2.1.	PUSH y POP	110
3.1.10.3.	Operaciones Aritméticas.....	114
3.1.10.4.	Operaciones lógicas	117
3.1.10.5.	Prueba y compara	118
3.1.10.6.	Control del flujo del programa.....	119
3.1.10.6.1.	Saltos (<i>Branch</i>).....	120
3.1.10.6.2.	Saltos (<i>Branches</i>) condicionados.....	120
3.1.10.6.3.	Llamada de función	123
3.1.10.6.4.	Compara y salto condicional.....	123
3.1.10.7.	Otro tipo de instrucciones	125
3.1.10.7.1.	NOP.....	126
3.1.11.	Instrucciones específicas para Cortex®-M4 basadas en DSP mejorado	126
3.1.11.1.	Multiplicación e instrucciones MAC ...	127
3.1.11.2.	Instrucciones de punto flotante	128
3.1.11.3.	Números con punto flotante de precisión simple	129
3.1.11.4.	Números con punto flotante de precisión media.....	130
3.1.11.5.	Números con punto flotante de doble precisión	131
3.1.12.	Unidad de punto flotante en Cortex®-M4	132
3.1.12.1.	Registro CPACR.....	136
3.1.12.2.	Banco de registros para punto flotante.....	137

3.1.12.3.	FPSCR	139
3.1.12.4.	Opciones de la línea de comandos del compilador	145
4.	PROBLEMAS PROPUESTOS EN LENGUAJE ENSAMBLADOR.....	147
4.1.	Crear un proyecto nuevo en Keil® uVision.....	147
4.2.	Constructor y Depurador	154
4.3.	Problema propuesto 1	155
4.3.1.	Análisis del problema	156
4.3.2.	Solución del problema 1	157
4.3.3.	Análisis de la solución	157
4.4.	Problema propuesto 2	158
4.4.1.	Análisis del problema	159
4.4.2.	Solución del problema 2	159
4.4.3.	Análisis de la solución	160
4.5.	Problema propuesto 3	161
4.5.1.	Análisis del problema	161
4.5.2.	Solución de problema 3.....	162
4.5.3.	Análisis de la solución	163
4.6.	Problema Propuesto 4.....	165
4.6.1.	Análisis del problema	165
4.6.2.	Solución de problema 4.....	166
4.6.3.	Análisis de la solución	167
4.7.	Problema Propuesto 5.....	167
4.7.1.	Análisis del problema	168
4.7.2.	Solución de problema 5.....	168
4.7.3.	Análisis de la solución	169
4.8.	Problema Propuesto 6.....	170
4.8.1.	Análisis del problema	170

4.8.2.	Solución del problema 6	171
4.8.3.	Análisis de la solución	171
4.9.	Problema Propuesto 7	173
4.9.1.	Análisis del problema.....	173
4.9.2.	Solución del problema 7	174
4.9.3.	Análisis de la solución	175
4.10.	Problema Propuesto 8	177
4.10.1.	Análisis del problema.....	177
4.10.2.	Solución de problema 8	179
4.10.3.	Análisis de la solución	179
4.11.	Problema Propuesto 9	181
4.11.1.	Análisis del problema.....	181
4.11.2.	Solución del problema 9	184
4.11.3.	Análisis de la solución	184
4.12.	Problema propuesto 10	186
4.12.1.	Análisis del problema.....	186
4.12.2.	Solución del problema 10	188
4.12.3.	Análisis de la solución	188
4.13.	Problema Propuesto 11	189
4.13.1.	Análisis del problema.....	190
4.13.2.	Solución del problema 11	191
4.13.3.	Análisis de la solución	191
4.14.	Problema Propuesto 12	193
4.14.1.	Análisis del problema.....	193
4.14.2.	Solución de problema 12	194
4.14.3.	Análisis de solución	194
4.15.	Problema Propuesto 13	197
4.15.1.	Análisis del problema.....	198
4.15.2.	Solución de problema 13	198

4.15.3.	Análisis de la Solución	199
4.16.	Problema Propuesto 14.....	200
4.16.1.	Análisis del problema	201
4.16.2.	Solución del problema 14	201
4.16.3.	Análisis de la solución	202
4.17.	Problema Propuesto 15.....	204
4.17.1.	Análisis del problema	204
4.17.2.	Solución del problema 15.....	205
4.17.3.	Análisis de la solución	205
4.18.	Problema propuesto 16.....	207
4.18.1.	Análisis del problema	207
4.18.2.	Solución del problema 16.....	208
4.18.3.	Análisis de la solución	208
4.19.	Problema Propuesto 17.....	210
4.19.1.	Análisis del problema	210
4.19.2.	Solución del problema 17	213
4.19.3.	Análisis de la solución	214
4.20.	Problema Propuesto 18.....	216
4.20.1.	Análisis del problema	216
4.20.2.	Solución del problema 18.....	218
4.20.3.	Análisis de la Solución	219
CONCLUSIONES.....		221
RECOMENDACIONES		223
BIBLIOGRAFÍA.....		225

ÍNDICE DE ILUSTRACIONES

FIGURAS

1.	Versiones de procesadores ARM y sus respectivas familias	27
2.	Encapsulado de un SoC y su vista interna	28
3.	Configuración interna básica de un procesador	31
4.	Almacenamiento de los bits en memoria.....	32
5.	Mapeado de la memoria y direcciones de un procesador	33
6.	Mapa de memoria para procesador Cortex-M4.....	34
7.	David Patterson and Carlos Séquina	36
8.	Arquitectura RISC vs CISC	38
9.	Diagrama de la arquitectura Von Neumman	39
10.	Diagrama de la arquitectura Harvard	40
11.	Diagrama de la arquitectura Harvard modificada	41
12.	Instrucciones en la familia Cortex-M	44
13.	Diagrama de bloques para Cortex®-M4 y M3.....	46
14.	Diagrama de un Cortex-M3 y M4	47
15.	Instrucción SIMD	49
16.	Tarjeta perforada.....	50
17.	Jerarquía de las computadoras	54
18.	Regiones de operación del transistor	55
19.	Representación del sistema decimal.....	57
20.	Representación del sistema binario	57
21.	Representación del sistema hexadecimal	58
22.	Sistema decimal, binario y hexadecimal	59
23.	Equipos Ulink	60

24.	Versiones de <i>MDK</i> disponibles para descargar	61
25.	Diagrama del funcionamiento de Keil® MDK-ARM.....	63
26.	Página web de Keil versión 4.....	66
27.	<i>Wizard</i> de instalación.....	67
28.	Formulario de datos de usuario	67
29.	Formulario completo de usuario.....	68
30.	Agregar ejemplos básicos en Keil.....	68
31.	Finalizar instalación de Keil uVision® 4	69
32.	Keil uVision® 4 instalado correctamente	69
33.	Búsqueda de Keil uVision® 5	70
34.	Selección de MDK5 -Texas Instruments para descargar.....	71
35.	Selección de MDK versión 5 para descargar.....	71
36.	Descarga de MDK v5.....	72
37.	Selección de ruta para instalar Keil uVision® 5	72
38.	Formulario de datos de usuario	73
39.	Instalación de <i>Ulink Drivers</i>	73
40.	Keil uVision® 5 instalado correctamente	74
41.	Instalación de paquetes en Keil uVision®5.....	75
42.	Búsqueda de paquetes para <i>Tiva C Series</i>	76
43.	Selección del paquete <i>TM4C123x Series</i>	76
44.	Instalación de Keil::TM4C_DFP.....	77
45.	Página oficial de Texas Instruments	78
46.	Búsqueda de ARM CORTEX.....	78
47.	Búsqueda del software para <i>ARM CORTEX</i>	79
48.	Búsqueda de <i>Stellaris® ICDI Drivers</i>	80
49.	Descarga de <i>Stellaris® ICDI Drivers</i>	80
50.	Conexión del cable USB A.....	81
51.	Conexión del cable USB en la Tiva C.....	81
52.	Conexión Tiva C y computadora.....	82

53.	<i>Device Manager</i> de Windows.....	83
54.	<i>Update Drivers</i> en <i>Device Manager</i>	83
55.	Ubicación de los drivers dentro de Windows.....	84
56.	Selección de la ubicación de los drivers.....	84
57.	Actualización exitosa de drivers para <i>Stellaris® ICDI</i>	85
58.	Búsqueda de <i>Stellaris® ICDI Debug Adapter Support</i>	86
59.	Instalación de <i>Stellaris® ICDI Debug Adapter Support</i>	86
60.	Verificación de la correcta instalación de <i>Stellaris® ICDI</i>	87
61.	Banco de registros del núcleo	90
62.	Registros APSR, EPSR e IPSR	94
63.	Bits dentro del registro xPSR	94
64.	Registro xPSR.....	95
65.	Acceso al registro PSR	95
66.	Acceso individual al registro PSR	95
67.	Campos para utilizar en <i>assembler</i>	96
68.	Sintaxis en lenguaje ensamblador	97
69.	Sintaxis de código	98
70.	Implementación de sintaxis en <i>assembler</i>	98
71.	Instrucción LDR Y MOV	99
72.	Uso de las instrucciones LDR y STR	99
73.	Datos en programa.....	100
74.	Ejemplo de código con UAL y pre-UAL.....	106
75.	Valor de 32 bit en R0.....	109
76.	Carga y descarga de la Pila mediante PUSH y POP	112
77.	Instrucción POP y PUSH.....	113
78.	Variantes de la instrucción ADD.....	115
79.	Ejemplo de uso instrucciones de salto	122
80.	Ejemplo de CBZ en código C	124
81.	Ejemplo de la instrucción CBZ	124

82.	Ejemplo de CBNZ en Java.....	125
83.	Ejemplo de la instrucción CNBZ	125
84.	Ejemplo del uso de NOP.....	126
85.	Datos SIMD posibles en un registro de 32 bits	127
86.	Diagrama del procesador Cortex®-M4	128
87.	Formato de datos de precisión simple	129
88.	Formato normalizado para datos con precisión simple.....	129
89.	Formato de datos de precisión media.....	130
90.	Formato normalizado para datos de precisión media	131
91.	Formato de datos de precisión doble.....	131
92.	Formato normalizado para datos con precisión doble	132
93.	Concepto de pipeline en coprocesador.....	134
94.	Registro de control de acceso al coprocesador	136
95.	Configuración del registro CPACR.....	137
96.	Registros para datos con punto flotante	138
97.	Bits en el registro FPSCR	139
98.	Aproximaciones basadas en el estándar IEEE 754	144
99.	Uso de FPU en Keil MDK	145
100.	Nuevo Proyecto	147
101.	Selección del nombre de archivo	148
102.	Selección de CPU.....	149
103.	Copiar archivo Startup	149
104.	Nuevo ítem	150
105.	Nuevo ítem al grupo.....	151
106.	Archivo <i>Startup</i>	152
107.	Opción llamada <i>Target</i>	152
108.	Selección de <i>Use Micro Lib</i>	153
109.	Selección de <i>Use Simulator</i>	153
110.	<i>Rebuild</i>	154

111.	Debug	155
112.	Código fuente del problema 1	157
113.	Resultados del problema 1	158
114.	Código fuente del problema 2	159
115.	Datos en memoria	160
116.	Diagrama de flujo del problema 3	162
117.	Código fuente del problema 3	162
118.	Resultado del problema 3	164
119.	Diagrama de flujo del problema 4	166
120.	Código fuente del problema 4	166
121.	Resultado del problema 4	167
122.	Ecuación propuesta del problema 5	168
123.	Código fuente del problema 5	168
124.	Resultados del problema 5	169
125.	Ecuación propuesta del problema 6	170
126.	Código fuente del problema 6	171
127.	Cambio de vista de la memoria	172
128.	Resultados del problema 6	172
129.	Código fuente del problema 7	174
130.	Resultado del problema 7	176
131.	LSL y LSR	177
132.	Multiplicación indirecta con LSL	178
133.	División indirecta con LSR	178
134.	Código fuente del problema 8	179
135.	Resultados del problema 8	180
136.	Resultados matemáticos del problema 8	180
137.	Diagrama de flujo del problema 9	183
138.	Código fuente del problema 9	184
139.	Valores en memoria del problema 9	185

140.	Resultado del problema 9	185
141.	Factorial de un número	186
142.	Diagrama de flujo del problema 10	187
143.	Código fuente del problema 10	188
144.	Resultado del problema 10	189
145.	Configuración del registro CPACR.....	190
146.	Código fuente del problema 11	191
147.	Valores dentro de FPU	192
148.	Resultados del problema 11	193
149.	Código fuente del problema 12	194
150.	Valores cargados en los registros.....	195
151.	Valores cargados en memoria	195
152.	Manipulación de valores en los registros	196
153.	Resultado del problema 12	197
154.	Energía cinética	197
155.	Código fuente del problema 13.....	198
156.	Conversión a decimal	199
157.	Resultado del problema 13	200
158.	Perímetro de un círculo.....	200
159.	Código fuente del problema 14.....	201
160.	Valores almacenados en registros.....	202
161.	Resultado del problema 14	203
162.	Área de un triángulo rectángulo	204
163.	Código fuente del problema 15.....	205
164.	Resultado del problema 15	206
165.	Área de un triángulo equilátero	207
166.	Código fuente del problema 16.....	208
167.	Solución del problema 16	209
168.	Diagrama de flujo del problema 17	211

169.	Código fuente del problema 17	213
170.	Datos del problema 17	214
171.	Resultado del problema 17	215
172.	Aproximación de $\text{Sen}(x)$	216
173.	Aproximación de $\text{Sen}(x)$ con $n = 3$	217
174.	Código fuente del problema 18	218
175.	Resultado del problema 18	219

TABLAS

I.	Criterios del aprendizaje.....	3
II.	Diferencia entre el racionalismo y empirismo.....	6
III.	Unidades de medidas en los procesadores	30
IV.	Rango de memorias para procesadores Cortex M3 y M4.....	35
V.	Interfaces de bus en procesadores Cortex® M3 y M4	47
VI.	Representación de mnemónicos para Intel i386	51
VII.	Directivas para insertar datos en programa	101
VIII.	Directivas importantes.....	102
IX.	Sufijos para lenguaje ensamblador para Cortex®-M.....	105
X.	Instrucciones para transferencia de datos.....	107
XI.	Transferir datos entre la FPU y los registros centrales.....	108
XII.	Instrucciones de acceso a memoria	110
XIII.	PUSH y POP	111
XIV.	VPUSH y VPOP	114
XV.	Instrucciones para operaciones aritméticas	116
XVI.	Instrucciones para multiplicar y MAC	117
XVII.	Instrucciones para álgebra booleanas.....	118
XVIII.	Instrucciones de prueba y compara	119
XIX.	Instrucciones de salto sin condición	120

XX.	Estado de los bits de las banderas del registro APSR.....	121
XXI.	Instrucciones para salto condicionado	121
XXII.	Sufijos condicionales	122
XXIII.	Instrucción para llamar o invocar a una función.....	123
XXIV.	Instrucciones de multiplicación y MAC.....	127
XXV.	Ejemplo para valores con punto flotante.....	130
XXVI.	Registros FPU adicionales.....	135
XXVII.	Configuración CP10 y CP11	136
XXVIII.	Descripción de bits en FPSCR	139
XXIX.	Banderas N, Z, C, y V dentro de FPSCR.....	141
XXX.	Operaciones para datos con punto flotante	141
XXXI.	Valores para operar	155
XXXII.	Operaciones matemáticas	156
XXXIII.	Dirección en memoria	159
XXXIV.	Dirección y valor	161
XXXV.	Valores propuestos	165
XXXVI.	Registros y valores del problema 7.....	173
XXXVII.	Banderas del registro xPSR.....	174
XXXVIII.	Valores del problema 9	181
XXXIX.	Registros renombrados con RN.....	181
XL.	Valores para problema 11.....	190
XLI.	Vueltas y tiempos.....	210

LISTA DE SÍMBOLOS

Símbolo	Significado
cm	Centímetro
/	División
Ghz	Gigahertz
Hz	Hertz
=	Igualdad
J	Julios
Kg	Kilogramos
C	Lenguaje de programación
Mhz	Megahertz
*	Multiplicación
-	Resta
S	Segundos
+	Suma
®	Símbolo de marca registrada
Vi	Valor inmediato

GLOSARIO

AND	Operación lógica en donde el resultado es 1 si solo si todas las entradas también son 1 de lo contrario el resultado es 0.
ARM	Arquitectura de procesadores tipo RISC de 32 bits o 64 bits.
Bit	Unidad básica para el campo de las computadoras y comunicaciones, el cual puede tener dos valores cero o uno.
Bus	Sistema digital para transferir datos entre componentes o dispositivos electrónicos, compuesto por un conjunto de cables o pistas en un circuito.
CMC	Cualquier tipo de comunicación entre personas mediante el uso de dos o más dispositivos electrónicos.
Compilador	Software o programa que traduce un código fuente escrito en un lenguaje de alto nivel a un lenguaje de bajo nivel.
DSP	Sistema basado en un microprocesador especializado para procesar señales digitales.

Educación	Disciplina que se basa en analizar los métodos de enseñanza para la adquisición de conocimiento.
IEEE	El Instituto de Ingenieros electrónicos y eléctricos es una asociación encargada de normalizar nuevas tecnologías y protocolos de comunicación.
Lenguaje de máquina	Sistema de código directamente interpretable por un microprocesador conformado por unos y ceros.
Memoria Flash	Memoria eléctricamente programable, este tipo de memoria no volátil.
Memoria RAM	Memoria de acceso aleatorio de tipo volátil, es utilizada cuando la velocidad de acceso a la información es muy importante.
Memoria ROM	Memoria de solo lectura, este tipo de memoria es no volátil.
MMU	Circuitos integrados encargados de manejar el acceso a la memoria.
MPU	Unidad de protección de memoria, permite definir rangos con diferentes permisos y puede leer información de fallas.

NOT	Operación lógica la cual da como resultado el valor invertido de la entrada.
OR	Operación lógica en la cual el resultado es 1 si solo si una de las entradas es 1 de lo contrario el resultado es 0.
Saturación	Estado de un circuito electrónico en donde la magnitud de voltaje de salida posee una tención próxima a la fuente con la que se está alimentado.
Sensores	Dispositivo electrónico capaz de detectar magnitudes físicas o químicas e interpretarlas de forma digital.
SIMD	Componentes electrónicos capaces de operar múltiples datos de forma paralela al mismo tiempo.
USB	Bus de comunicación con interfaz de 4 cables que interconectan dispositivos electrónicos con una computadora.

RESUMEN

El presente trabajo de graduación brinda material digital bajo el modelo constructivista de educación para el curso de Electrónica 5 de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, en la Escuela de Ingeniería Mecánica Eléctrica. Se estudian los inicios del lenguaje ensamblador, así como su correcta implementación en los procesadores de la familia Cortex®-M.

En el primer capítulo se estudia el aprendizaje en la educación, se explica cómo las personas mediante el constructivismo van aprendiendo y construyen su propio conocimiento. Además, se habla sobre la importancia de las TICs para brindar educación de calidad a distancia. El segundo capítulo explica los inicios del lenguaje de programación ensamblador y expone a la familia de procesadores llamados Cortex, los cuales pueden ser programados en ensamblador. Se explica la correcta instalación del software de desarrollo Keil® MDK, en el cual será utilizado para programar el procesador Cortex®-MF.

En el tercer capítulo se enuncian todas las características internas del procesador Cortex®-M, como lo son: registros, Pila, sintaxis, directivas, set de instrucciones para valores enteros y decimales. El cuarto y último capítulo se proponen y resuelven varios ejemplos en lenguaje ensamblador, esto con la finalidad de que el lector pueda comprender totalmente el uso del lenguaje.

OBJETIVOS

General

Diseñar e implementar material digital bajo el modelo constructivista en la educación para el curso de Electrónica 5.

Específicos

1. Explicar claramente el uso del constructivismo para adquirir nuevo conocimiento.
2. Conocer los inicios del lenguaje ensamblador.
3. Presentar la familia de procesadores Cortex®-M así como su arquitectura.
4. Explicar el uso correcto de las instrucciones, directivas, registros, operaciones y saltos en el lenguaje ensamblador utilizando Keil® uVision MDK.
5. Diseñar varios problemas para ser resueltos en el lenguaje ensamblador utilizando el microcontrolador TM4C123GH6PM.

INTRODUCCIÓN

Con el avance de la tecnología los dispositivos electrónicos han disminuido de tamaño y aumentado su velocidad de procesamiento. Todos los dispositivos electrónicos necesitan de un procesador, como los procesadores de la familia Cortex. La familia de procesadores Cortex son de alto rendimiento, bajo consumo energético y fáciles de programar en lenguaje ensamblador.

Arm Holdings es una compañía la cual lejos de producir masivamente procesadores sólo se encarga de diseñar las distintas arquitecturas de los procesadores y posteriormente vender los derechos para que otras empresas se encarguen de producirlos e implementarlos en dispositivos electrónicos

Texas Instruments implementa el Cortex®-MF dentro del microcontrolador TM4C123GH6PM. El procesador cuenta con 12 registros de propósito general y 3 registros de uso especial dentro del núcleo (con un tamaño de 32 bits). También cuenta con una unidad de punto flotante o FPU basada en el estándar IEEE 754 la cual brinda la posibilidad de poder operar datos con decimales de precisión simple. La FPU cuenta con 32 registros con un tamaño de 32 bits o al agruparlos en pares se obtiene 16 registros con tamaño de 64 bits

La jerarquía de una computadora es necesaria conocerla, esto ayuda a saber qué es lo que sucede al momento de programar en un lenguaje de bajo nivel. En el nivel llamado ISA se encuentran el conjunto de especificaciones e instrucciones necesarias para poder programar un microprocesador.

Existen varios tipos de ISAs las cuales se diferencian unas ante otras con base en la cantidad de instrucciones que posee, la longitud y cómo se procesa esa información. Dentro de las ISAs se pueden mencionar a la arquitectura CISC y RISC. Cabe resaltar que los procesadores Cortex-M se basan puramente en la arquitectura RISC

Para evitar lidiar con el lenguaje de máquina (basada en 0 y 1) se implementa el uso del lenguaje ensamblador ya que dispone de código simbólico (nemónicos) para reemplazar los largos sets de instrucciones escritos puramente con 0 y 1. Para programar el procesador Cortex®-MF se tiene que tomar en cuenta el entorno de desarrollo a usar, en qué software se va a realizar el código. Para el presente trabajo se estará trabajando con Keil® MDK-Lite.

1. APRENDIZAJE EN LA EDUCACIÓN

La educación consiste en una disciplina que analiza los métodos de enseñanza y facilita el aprendizaje o la adquisición de conocimiento. Por otro lado, el aprendizaje causa un cambio en la forma de ser o pensar de la persona en función de la experiencia individual o grupal. El término aprendizaje y educación se ha utilizado como sinónimo lo cual es incorrecto, son procesos completamente distintos.

Una buena educación en la persona depende plenamente de un buen aprendizaje. Actualmente, se tienen problemas respecto a la educación sin un aprendizaje, en donde sólo se hace repetir y memorizar al estudiante. El problema anterior también está presente en la educación superior en donde lo enseñado no tiene relación con lo que se encuentra en el ámbito laboral.

Aunque se logre memorizar todo el contenido o temas de estudio requeridos no es suficiente, no se trata de aprender sino también de aprender a aprender y razonar lo aprendido. Se necesita comprender que el aprendizaje no es algo impuesto y romper el paradigma de que el estudiante no puede aprender o enseñar por sí solo, sólo hay que tener en cuenta que la educación siempre tiene que ir acompañada de la supervisión de un docente o tutor o un trabajo constante por parte del estudiante.

1.1. La educación

La educación nos concierne a todos desde el momento de nacer. El término “educación” puede tener distinta interpretación, se puede decir que es el proceso

de facilitar la adquisición de nuevos conocimientos, sin embargo, la educación puede ocurrir indirectamente sin ninguna ayuda externa. La educación inicia desde el momento en que nuestro cerebro empieza a retener información y la empieza a asociar con acciones. Conceptualmente se usa con frecuencia el vocablo para otorgar un significado a acontecimientos que se relacionan con lo educativo.

El ser humano basa su desarrollo social y psicológico en la educación, la educación brinda un incesante aprendizaje también llamado proceso de Humanización. Este proceso brinda a las personas cualidades de ser empáticas, afectivas, amorosas, racionales y comunicativas. García Carrasco (1987), destaca el hecho de que la educación no se refiere a una actividad más bien a un conjunto de ella.

Brindar una definición muy precisa del significado del término “Educación” es muy complicado, por tal se procederá a establecer límites para poder establecer un significado correcto. Se describe las características más sobresalientes que logran conformar el término “Educación”:

- La humanización, acción y efecto por la cual las personas se dotan de cualidades humanas, estas ayudan a una persona a tener actitudes relevantes.
- Los elementos básicos de la educación:
 - Emisor, persona que brinda nuevo conocimiento, puede ser pasivo o activo.
 - Receptor, persona que recibe o capta la información de forma directa o indirecta.

- La acción brindada por el educador y el educando en una situación educativa. La enseñanza brindada es de forma sistematizada y gradual. El aprendizaje se estudia mediante la manipulación, la interacción y la formación.

1.1.1. Enseñanza y Aprendizaje

La enseñanza dentro del constructivismo puede ser implementada mediante distintos métodos, en los cuales el estudiante logra adquirir nuevo conocimiento de forma correcta. Ahora bien, el término aprendizaje difiere en su significado dependiendo del punto de vista, ya que no existe una definición plenamente aceptada por todos los teóricos, investigadores, pedagogos y otros profesionales.

1.1.1.1. Teorías del aprendizaje

Una definición básica para el término aprendizaje es:

“Adquisición del conocimiento de algo por medio del estudio, el ejercicio o la experiencia, en especial de los conocimientos necesarios para aprender algún arte u oficio.”¹

Tabla I. **Criterios del aprendizaje**

1.º	El aprendizaje implica un cambio en la forma de ser y actuar
2.º	El aprendizaje es adquirir y compartir conocimientos
3.º	El aprendizaje puede ocurrir mediante varios métodos

Fuente: elaboración propia, empleando Microsoft Word.

¹ EDU GESTORES. *Conocimiento y Aprendizaje*. 2017. <https://www.edugestores.pe/docs/conocimiento-y-aprendizaje/history/?revision=14025>. Consulta: 1 de enero de 2020.

El aprendizaje conlleva un cambio en la forma de ser de una persona. Las personas aprenden cuando adquieren nuevo conocimiento y logran realizar distintas actividades lógicas y físicas de distinta manera, de una forma más eficiente.

El aprendizaje es inferencial, la persona aprende y comprende nuevos conocimientos sin darse cuenta. Al obtener nuevo conocimiento el aprendizaje en las personas inicia, luego del aprendizaje estas personas adquieren nuevas habilidades, nuevos conocimientos, creencias o conductas las cuales no se perciben inmediatamente.

Un segundo criterio consiste en adquirir nuevo conocimiento, independientemente de la forma en que se obtuvo el nuevo conocimiento las personas logran transmitir lo ya aprendido hacia otras personas. Compartir el conocimiento de una persona a otra puede ser por distintos medios.

Un tercer criterio es el aprendizaje mediante la experiencia. Esto se logra obtener mediante la práctica o al observar algún evento que ocurra a su alrededor. Lo anterior cambia la forma de ser y pensar de la persona.

El aprendizaje permite a la persona construir su propio pensamiento mediante su propia experiencia, le brinda nuevas habilidades y logra mejorar sus valores como persona. Algunos de los beneficios del aprendizaje mediante la experiencia son:

- Permite un crecimiento personal a partir del conocimiento previo
- Mejora la estructura cognitiva del estudiante
- Ayuda a mejorar las actitudes de la persona de manera positiva
- Permite enfocar el aprendizaje

1.2. Precursores de las teorías modernas del aprendizaje

El origen de la teoría y postulado del aprendizaje se remontan a varios años. Muchos temas o problemas que se estudian hoy en día no son nuevos. Entre los orígenes de las teorías contemporáneas del aprendizaje, las posturas filosóficas sobre el origen del conocimiento y su aprendizaje se puede mencionar lo siguiente.

1.2.1. Teoría del aprendizaje

Desde el punto de vista filosófico, el aprendizaje podría definirse con el término epistemología. Esta es la parte de la filosofía encargada de estudiar los principios de la naturaleza y métodos del conocimiento humano, así como el origen de ese conocimiento.

El racionalismo como el empirismo son las posturas que intentan explicar la forma en que las personas adquieren conocimiento, tanto el racionalismo como el empirismo tienen fundamentos filosóficos opuestos.

En la tabla II, se brinda una breve explicación de las diferencias entre el racionalismo y el empirismo.

Tabla II. **Diferencia entre el racionalismo y empirismo**

	Racionalismo	Empirismo
Base fundamental	La razón es la base del conocimiento humano	La experiencia es la base del conocimiento
Ideas	Las personas poseen conocimiento innato	Las personas no poseen conocimiento innato
El conocimiento	Se obtiene de la razón y lógica propia de cada persona	Se obtiene de la experiencia y en la propia experimentación durante la vida de la persona.

Fuente: elaboración propia, empleando Microsoft Word.

1.2.2. Racionalismo

Postura filosófica que defiende lo que se debe conocer de la verdad, para poder tener la razón. La razón es la generadora del conocimiento lo cual es innato del ser, los individuos al nacer ya poseen determinados conocimientos. La mente logra percibir el mundo exterior mediante los sentidos y los manipula con leyes innatas. La persona aprende del mundo mediante su propia percepción, por esa razón no se tiene un conocimiento absoluto. El conocimiento es empírico ya que la información se toma del mundo exterior y la mente posteriormente lo interpreta. Por lo anterior se puede decir que el conocimiento surge mediante lo que percibe la mente, las ideas se generan en función de la mente.

1.2.3. Empirismo

En contraste con el racionalismo, el empirismo se basa en que la única fuente de conocimiento es mediante la experiencia y niega totalmente la idea innata del racionalismo.

La experiencia puede ser interna o externa, por esa razón se puede decir que es producto de los sentidos. De acuerdo con el empirismo la base, origen y límite del conocimiento es la experiencia. Se puede resumir como corriente filosófica basada en la práctica, experiencia y en la observación de los hechos.

1.3. El constructivismo en la educación como método de enseñanza

El constructivismo es una postura psicológica y pedagógica. Existen varias teorías las cuales son las bases del constructivismo, esas teorías fueron postuladas por Piaget, Vygotsky y Ausubel. Lo publicado por cada autor no se denomina postulado totalmente constructivista pero sus ideas son las guías de una corriente puramente constructivista.

1.3.1. Jean Piaget

Jean William Fritz Piaget fue un epistemólogo y biólogo suizo, se dedicó al estudio y análisis de las respuestas erróneas las cuales son comunes en las personas al momento del desarrollo del conocimiento, estudió la razón de esas respuestas “erróneas”. Para Piaget no importa la autoridad sobre el estudiante durante su interacción, se interesa sencillamente en saber por qué ellos piensan de la forma en que lo hacen ya sea un pensamiento correcto o erróneo.

Piaget reinventa la Psicología del Desarrollo, ese Desarrollo no es más que una adición del nuevo conocimiento. Las etapas para obtener un nuevo conocimiento suceden ante conflictos, accidentes y eventos importantes en la vida de la persona. El estudiante en cada etapa elabora teorías acerca del mundo circundante sin importar su veracidad.

1.3.1.1. El origen del pensamiento humano

Piaget estudiaba cómo surge el pensamiento en los niños, estaba convencido de que el modo en que evoluciona el pensamiento infantil servía para comprender el pensamiento racional en su estado más puro, el cual es el pensamiento científico. Piaget habla sobre la utilidad de la comprensión del pensamiento racional como resultado de una evolución. Él acumuló pruebas que sostienen un nuevo modo de comprender la evolución de la inteligencia, el proceso del desarrollo y cómo se va construyendo el conocimiento es llamado *constructivismo*.

Piaget demostró que la actividad psíquica del niño no es totalmente gráfica, demostró que nociones tan simples como el espacio, casualidad y otros eventos eran resultado de las experiencias adquiridas por el niño en los primeros años de su vida.

Para Piaget la inteligencia y capacidad para aprender se encuentra ligado al medio físico y social al que el niño o niña esté expuesto. La lógica es base fundamental para el entendimiento y la capacidad cognitiva, la cual se desarrolla en los primeros días de vida y continúa paulatinamente con el transcurrir de los años.

1.3.2. Lev Vygotsky

Lev Semiónovich Vygotsky fue un psicólogo, fundador de la psicología histórico-cultural y precursor de la neuropsicología soviética. Vygotsky plantea su modelo de aprendizaje sociocultural, sostiene que los eventos socioculturales proporcionan nuevos conocimientos. Se explica el aprendizaje como una forma de socialización. El conocimiento es causa del desarrollo cultural.

El concepto básico es la zona de desarrollo próximo (ZDP) en la cual cada estudiante es capaz de aprender una serie de habilidades o nuevas formas de pensar lo cual se basa en su nivel de desarrollo, pero existen otros conceptos o habilidades que pueden ser comprendidos mediante la ayuda de adultos. En resumen, lo que el estudiante puede aprender por sí solo y con ayuda de otros se le conoce como ZDP. En este postulado el docente tiene un papel muy importante en el desarrollo de las estructuras mentales debido a que facilita el aprendizaje hacia en el estudiante, el estudiante logra construir conocimiento mucho más complejo.

Para Vygotsky existen dos tipos de funciones mentales: inferiores y superiores. Las inferiores son las funciones mentales con las que se nace y están determinadas genéticamente. Las funciones mentales superiores se adquieren y desarrollan a través de la interacción sociocultural, estas funciones están determinadas por la forma de ser de la sociedad en la que el estudiante se encuentra involucrado.

1.3.2.1. Método de enseñanza

Los estudiantes se colocan en situaciones en las cuales su mente se ve forzada a atender el problema, los estudiantes disponen del apoyo de otros compañeros o docentes. Se toma mucho en cuenta otro estudiante porque puede ser que recién haya resuelto el problema y será más sencillo explicar la solución. Los docentes deben guiar a los estudiantes con explicaciones, demostraciones, ejemplos similares y demás.

1.3.3. David Ausubel

David Paul Ausubel fue un psicólogo y pedagogo de gran importancia para el constructivismo, su teoría es cognitiva. Ausubel ponía mucho énfasis en la enseñanza la cual según él se logra a partir de los conocimientos que ya posee el alumno. Por lo anterior el primer paso para garantizar un buen aprendizaje es averiguar lo que el estudiante ya sabe.

Aprendizaje significativo se le llama al momento en el cual la información adquiere un significado comprensible para el estudiante a través de la asociación de conceptos existentes en su memoria. “La teoría de la Asimilación” permite comprender claramente el aprendizaje significativo, ayuda a comprender cómo los nuevos conocimientos se integran a los existentes. La asimilación ocurre cuando una nueva información logra ser integrada al conocimiento, se genera una asociación entre ellas, y esa asociación permite expandir el conocimiento antiguo.

Una de las ventajas del aprendizaje significativo implica una construcción del conocimiento intencional, por esa razón la información aprendida significativamente será asociada y retenida más tiempo en la mente del estudiante. El conjunto de conceptos adquiridos por el estudiante en la estructura cognitiva será único. Aunque se haya usado la misma tarea de aprendizaje cada persona asociara a su propia manera el nuevo conocimiento.

1.3.4. ¿Qué es el constructivismo?

No existe una única definición del término constructivismo, el término no es una teoría sino una epistemología o explicación filosófica acerca de la naturaleza

del aprendizaje. en concreto propone que las personas crean o van construyendo su propio conocimiento.

Hay muchas verdades, pero ninguna debe ser considerada más correcta que las otras. El constructivismo no propone que existan principios del aprendizaje que se deban descubrir y poner a prueba, sino que las personas crean su propio aprendizaje.

Los principales exponentes constructivistas rechazan la idea de que existan verdades científicas absolutas, ya que se espera el descubrimiento y la verificación. Ninguna afirmación se puede considerar como verdadera. En lugar de considerar el conocimiento como verdadero, los constructivistas lo definen como una hipótesis de trabajo el cual da inicio cuando los seres humanos empiezan a adquirir conocimiento.

El conocimiento no inicia de forma externa ante una persona, sino que se genera dentro de ellas. La construcción de ese conocimiento para una persona es plenamente verdadera, pero no necesariamente para los demás. Lo anterior debido a que cada persona construye su conocimiento con base en sus creencias y experiencias, entonces todo el conocimiento es subjetivo y personal.

1.4. Aspectos básicos según Piaget, Vygotsky y Ausubel

Los aspectos más relevantes del constructivismo según los autores son:

- Piaget

Reinventa la Psicología del Desarrollo, ese desarrollo se da mediante la adición de nuevo conocimiento. Para que las personas logren adquirir nuevo

conocimiento debe existir conflictos, accidentes o eventos importantes en la vida de ellas.

- Vygotsky

Enuncia el aprendizaje sociocultural, eventos socioculturales que ocurren en la vida de una persona proporcionando nuevos conocimientos. Lo que la persona puede aprender por sí solo y con ayuda de otros se conoce como ZDP. Los profesionales deben guiar a los estudiantes con explicaciones claras, demostraciones, ejemplos similares y demás.

- David Ausubel

Basando su postulado en la experiencia, las personas estructuran el mundo a través de las percepciones de sus experiencias. La teoría de la Asimilación permite comprender claramente el aprendizaje significativo el cual ayuda a comprender cómo los nuevos conocimientos se integran a los existentes.

La asimilación ocurre cuando una nueva información se integra al antiguo conocimiento, se genera una asociación entre ellas y esa asociación permite expandir el conocimiento. Aunque se haya usado la misma tarea de aprendizaje cada persona asocia a su manera el nuevo conocimiento.

1.5. Ambientes de aprendizaje constructivistas

Aprender en un ambiente constructivista no quiere decir que los estudiantes pierdan el tiempo, en este tipo de ambiente se debe crear actividades y material estimulante que fomente el aprendizaje sin interrupciones.

Las aulas constructivistas se basan en los conceptos más importantes, las actividades suelen incluir fuentes de los datos y materiales extras. Los docentes interactúan con los estudiantes averiguando lo que les interesa y sus puntos de vista. La clave es estructurar el ambiente de aprendizaje de forma que los estudiantes puedan construir de una forma eficiente nuevos conocimientos y habilidades.

En una clase tradicional, durante una lección nueva, las suposiciones y las respuestas brindadas por los estudiantes pueden ser incorrectas, el catedrático debe de corregir la respuesta o pensamiento erróneo. En una enseñanza constructivista se reta a los mismos estudiantes a deducir la información correcta durante una lección.

En los ambientes de estudio constructivistas se genera un aprendizaje estructurado, significativo y profundo. Por lo anterior los exámenes de verdadero y falso no son muy objetivos para evaluar los resultados en un aprendizaje constructivista. Las evaluaciones eficaces para medir la calidad del aprendizaje del estudiante son mediante la redacción de textos sobre lo aprendido o que ellos demuestren y apliquen las nuevas habilidades adquiridas.

A las evaluaciones constructivistas no le interesa tanto las respuestas correctas ni las incorrectas sino las etapas posteriores a la emisión de las respuestas. En este tipo de evaluaciones se tienen el reto en el cual los profesores diseñen actividades que estimulen la retroalimentación del estudiante para que converjan en una respuesta correcta y que logren modificar lo aprendido en caso sea necesario.

1.6. La enseñanza dentro del constructivismo

Se presentan varios escenarios en los cuales el estudiante va adquiriendo de forma eficiente nuevos conocimientos mediante el constructivismo.

1.6.1. Enseñanza por el descubrimiento

Ocurre cuando el estudiante obtiene conocimientos por sí mismo. El estudiante plantea y prueba hipótesis, no solo lee y escucha lo que el catedrático expone.

El razonamiento es de tipo inductivo ya que los mismos estudiantes realizan reglas, conceptos y principios generales de lo que se está aprendiendo. Este tipo de aprendizaje también suele ser llamado aprendizaje basado en problemas, aprendizaje por indagación, aprendizaje de experiencia y aprendizaje constructivista.

La enseñanza dentro del descubrimiento plantea: preguntas, problemas o situaciones complejas. Se reta a los estudiantes a formular conjeturas y llegar a una idea o concepto totalmente correcto.

1.6.2. Enseñanza por indagación

Este tipo de enseñanza es un tipo de aprendizaje por descubrimiento. La meta es lograr que los estudiantes razonen y luego apliquen lo comprendido a situaciones nuevas. Este tipo de enseñanza fue diseñado para el aprendizaje individual, con algunos cambios se podrá usar en grupos pequeños de estudiantes. Un detalle muy importante con respecto a las preguntas es que éstas

deben ser planteadas por expertos en el tema y se deben realizar de acuerdo con el nivel del pensamiento del estudiante.

1.6.3. Aprendizaje asistido por los pares

Este tipo de enseñanza utiliza mucho los principios de la enseñanza constructivista. Los estudiantes participan de forma activa en el proceso de aprendizaje, el estudiante y profesor participan con plena libertad. En el contexto individual en el cual se desarrolla este tipo de aprendizaje los estudiantes suelen hacer muchas preguntas, preguntas que no se atreverían a hacer en grupos grandes de estudiantes. Este tipo de enseñanza fomenta la cooperación entre estudiantes y ayuda a diversificar la estructura del grupo.

1.6.4. Aprendizaje cooperativo

El objetivo de este tipo de aprendizaje es desarrollar la habilidad de los estudiantes para trabajar en colaboración con otros estudiantes. Se debe aplicar de forma oportuna en tareas que son demasiado extensas para un solo estudiante. Existen principios que ayudan a que los grupos cooperativos tengan éxito. Uno de los criterios o principios es formar grupos con estudiantes que puedan trabajar bien juntos y que también puedan desarrollar habilidades de cooperación.

1.7. Evaluar el aprendizaje

El aprendizaje no se puede observar de manera directa, se observa mediante sus productos y resultados. Los investigadores y profesionales que educan a los estudiantes podrían asumir que, si se ha aprendido, pero la única

forma de garantizar el aprendizaje es mediante una evaluación de los productos y los resultados del aprendizaje.

La evaluación implica determinar la calidad del nuevo conocimiento brindado por los profesionales hacia los estudiantes. Los profesionales e investigadores desean saber si ha ocurrido un aprendizaje exitoso, para determinar lo anterior se tienen varios métodos aparte de las pruebas escritas, con las cuales es posible obtener evidencia de la calidad del aprendizaje adquirido por el estudiante. Lo que se evalúa en las pruebas son las nuevas habilidades o nuevos interés y motivación personal. Las evaluaciones se realizan al finalizar de estudiar el contenido propuesta.

Existen muchos métodos para evaluar el resultado del aprendizaje, los métodos abarcan la observación directa, los exámenes escritos, exámenes orales, la calificación por terceros y las autoevaluaciones.

1.7.1. Observación directa

Consiste en observar la conducta del estudiante, observar si el comportamiento del estudiante cambió con base en lo enseñado. Este método es usado con mucha frecuencia por el docente. En un laboratorio, el docente requiere que los estudiantes utilicen los instrumentos de forma correcta, él procede a enseñar sobre el buen uso de los instrumentos. El docente observará cómo los estudiantes utilizan los instrumentos. Este es un método válido si las observaciones son claras y sin suposiciones por parte del observador.

1.7.2. Examen escrito

La forma más común de evaluar el aprendizaje es mediante los exámenes escritos. En este tipo de examen el instrumento de evaluación consiste en pruebas objetivas en donde las respuestas son breves, opciones múltiples, respuestas verdaderas y falsas, y desarrollar el tema propuesto.

El estudiante responderá a las interrogantes con base en los conocimientos adquiridos. Una de las ventajas por parte del estudiante es responder a su propio ritmo, tiene más tiempo para reflexionar. El estudiante puede revisar cuidadosamente sus respuestas antes de entregar la prueba. Al finalizar la evaluación, cuando el docente califique las respuestas puede darse cuenta si el estudiante ha adquirido de forma correcta el nuevo conocimiento. El resultado concluye si se adquirió el conocimiento o si se requiere reforzar temas puntuales.

1.7.3. Exámenes Orales

Este tipo de exámenes evalúa el conocimiento adquirido por el estudiante mediante un diálogo abierto entre el docente y el alumno. Se basa en numerosas preguntas por parte del docente hacia el alumno referente al tema a evaluar. El estudiante debe responder a las preguntas de forma clara y con coherencia de esta forma demostrar que posee el conocimiento correcto sobre el tema.

Una de las desventajas del examen oral ante el examen escrito es el tiempo que el estudiante posee al responder. En un examen escrito el estudiante puede responder las preguntas sobre los temas que recuerda con facilidad y dejar las preguntas difíciles para el final, durante el examen oral el docente pregunta de forma consecutiva y el estudiante debe de poder responder las preguntas una tras otras.

1.7.4. Calificación de terceros

Otra forma de evaluar la calidad del conocimiento adquirido por los estudiantes es mediante la intervención de otros individuos diferentes a los alumnos y docentes. Los nuevos individuos pueden ser padres de familia, otros alumnos, investigadores, administradores, entre otros. Estos individuos verifican el nuevo conocimiento adquirido por los estudiantes. Se tiene una ventaja ante este tipo de calificación ya que los observadores podrían ser más objetivos al momento de realizar la evaluación, aunque se requiere más inferencia que las observaciones directas. Los evaluadores deben tener en cuenta lo que aprenden los estudiantes y tener el conocimiento claro de los temas a evaluar.

1.7.5. Autoevaluaciones

El estudiante evaluará su propio aprendizaje. Es un método en el cual el estudiante toma conciencia de su progreso individual durante el proceso de enseñanza y aprendizaje mediante su propia evaluación.

Las autoevaluaciones pueden tener diversos formatos como: cuestionarios, entrevistas y ejercicios resueltos.

- En un cuestionario se puede encontrar preguntas con base en el tema enseñado con el fin de averiguar si el estudiante logró aprender y comprender correctamente. Típicamente este tipo de pruebas son de respuestas con opción múltiples, por lo que el estudiante tendrá un grupo de posibles respuestas. Cada una de las posibles respuestas ya poseen una ponderación. Al terminar de responder todas las preguntas, el mismo estudiante realizará una calificación en una escala numérica (en un rango de puntos en donde 1 podría ser el más bajo, respuesta incorrecta, y el 10

el valor más alto para la respuesta totalmente correcta) dependiendo de la respuesta seleccionada. El estudiante se puede autoevaluar observando el resultado de puntos obtenido con base en las respuestas correctas.

- Las entrevistas son un tipo de cuestionario, el entrevistado plantea y el entrevistador responde de manera oral. La entrevista puede ser tanto individual como grupal. Por parte del entrevistador se intenta obtener respuestas largas y claras.
- Ejercicios resueltos, cuando se tienen problemas lógicos como un problema matemático se requiere llegar a un resultado exacto. El estudiante puede usar el método incorrecto o correcto y aun así lograr obtener una respuesta válida. Por esa razón se le brinda al estudiante los problemas y sus respectivas soluciones. Claramente esto no sucede al mismo tiempo, se pretende que el estudiante realice toda la prueba y luego pueda tener acceso a la solución de cada uno de los ejercicios. El estudiante podrá comparar los resultados, técnicas y métodos empleados para la solución de cada problema. El estudiante evaluará en que parte ha tenido problemas o en donde ha cometido errores graves y de esta forma evaluar la calidad del conocimiento adquirido.

1.8. Tecnologías de la información y la comunicación (TIC)

La forma de aprender y consultar información ha cambiado debido al gran progreso que ha tenido la tecnología. Con el avance de la tecnología el acceso a la información es cada vez más sencilla. La mayoría de información se ha digitalizado, hoy en día se tiene muchas publicaciones mediante medios digitales. La evolución que se ha tenido en el despliegue de información se ha logrado mediante las Tecnologías de la información y la comunicación.

TIC es un término que abarca tanto las tecnologías de las comunicaciones (TC) y las tecnologías de la información (TI). TC está constituida principalmente por equipos de radio, televisión, telefonía e internet. TI se les llama a las aplicaciones en computadoras o dispositivos inteligentes (como pueden ser celulares, tablets y televisores) capaces de almacenar, consultar, transmitir, y manipular datos.

Las TIC son un conjunto de herramientas, estas permiten consultar, revisar, almacenar y presentar información. Los medios más típicos en donde se pueden encontrar estas herramientas son la computadora y dispositivos inteligentes, pueden ser mediante el uso de la internet.

El uso de las TIC ha mostrado un cambio notable en la sociedad y un gran cambio en la educación debido a la forma de compartir y adquirir nuevos conocimientos. Mediante dispositivos como una computadora podemos acceder a información de nuestro interés desde cualquier parte del mundo, esa información puede estar previamente almacenada en la computadora o descargada de internet. Ahora ya no es necesario contar con material físico para poder adquirir nuevo conocimiento lo cual facilita el aprendizaje.

Los beneficios brindados por las TIC son:

- Facilita el acceso a la información
- Acceso a información totalmente gratuita
- Herramientas para procesar texto y datos
- Almacenamiento de nuestra propia información
- Interconectividad

1.8.1. E-learning

E-learning es un término abreviado para *electronic learning*, lo que se refiere a la enseñanza, aprendizaje y recopilación de información mediante la internet y la tecnología. Permite la creación de “aulas virtuales”, en donde el estudiante y catedráticos tienen interacción entre ellos. Se logra realizar pruebas cortas, evaluaciones, intercambio de información, y mediante chats o foros lograr resolver dudas.

Entre las grandes ventajas que ofrece E-learning están:

- Eliminación de la barrera física y temporal
- Aprendizaje y enseñanza sin restricción de horario
- Obtención de nuevo conocimiento de alta calidad y a bajo costo, muchas veces hasta de forma gratuita.
- No se requiere comprar libros pesados y caros.
- Foros o chats con solución a ejercicios o dudas.

Para que todo lo anterior sea posible, los catedráticos y estudiantes deben poder utilizar las TIC y se debería usar de la mejor forma posible. El material generado con la ayuda de la tecnología puede ser usado para la educación o para el entretenimiento, todo depende del uso que le dé el usuario final. Por lo anterior el catedrático debe generar contenido digital de alta calidad, ellos exigirán y establecerán la calidad del conocimiento que se pretende enseñar. La audiencia, en este caso los estudiantes, deben poder recibir de manera creativa el conocimiento de esta manera logrará captar la concentración total del estudiante para que se logre un aprendizaje de calidad.

En el tipo de educación E-learning la comunicación es esencial, este tipo de comunicación puede ser de dos tipos: síncrona y asíncrona. Se brindan las herramientas y características de los dos tipos de comunicación.

1.8.1.1. Comunicación Síncrona

La comunicación síncrona se logra cuando se tiene una conversación entre dos o varias personas en tiempo real. Durante este tipo de comunicación se pretende establecer una comunicación clara entre el emisor y el receptor del mensaje o información.

Cuando la tecnología se suma a este tipo de comunicación se logra obtener nuevas tecnologías como CMC (del inglés *Computer Mediated Communication*). CMC redefine el término “Comunicación Síncrona”, ahora se define como el proceso de intercambio de información en tiempo real la cual se logra con ayuda de Internet y plataformas digitales.

1.9. Herramientas usadas con la comunicación síncrona

Se brinda una explicación clara de las herramientas más utilizadas en la comunicación tipo síncrona.

1.9.1. Chat

Permite la comunicación en tiempo real entre varios usuarios, los usuarios utilizan el teclado de su dispositivo para enviar mensajes. El mensaje de texto es enviado y recibido inmediatamente por otros usuarios. Aparte de mandar texto es posible enviar imágenes, videos y archivos de diversos formatos. La mayoría de este tipo de herramientas son totalmente gratuitas.

1.9.2. Webinars

Este término proviene de la combinación de la palabra web y seminario, se obtiene un seminario impartido en línea. Los webinars se dan en tiempo real, previamente se establece una hora y fecha de la sesión. Se tiene una interacción en tiempo real entre el conferencista y los participantes, se puede hacer preguntas en tiempo real. La característica más importante es que se cuenta con una transmisión de audio y video simultáneo, por lo anterior se debe considerar tener una conexión a internet estable y rápida.

Una de las desventajas de este tipo de herramientas es el precio para obtener una versión completa de la aplicación o software. Las versiones completas o de pago eliminan la restricción del número máximo de personas que pueden acceder al Webinar además de eliminar el tiempo máximo de transmisión.

1.9.3. Videollamadas

Es un tipo de videoconferencia entre dos personas las cuales pueden verse y escucharse instantáneamente con la ayuda de internet. Se necesita de un software o aplicación en los depósitos de los usuarios para lograr la comunicación. Este tipo de herramienta es más personal, esto es muy útil cuando se requiere de asesoría o tutorías de forma individual.

1.10. Comunicación Asíncrona

Es otra categoría dentro del CMC (del inglés *Computer Mediated Communication*) en donde la comunicación en tiempo real no está presente, es

aquella que permite la comunicación entre varias personas mediante internet de forma no simultánea, el ejemplo más básico es el correo electrónico.

En este tipo de comunicación es el estudiante el único responsable de leer y analizar los temas propuestos por el catedrático. Muchas veces se tiene un foro en donde se publican preguntas o dudas sobre los temas estudiados las cuales el catedrático responderá de forma no inmediata así mismo la comunicación puede ser mediante correo electrónico.

La comunicación entre el estudiante y catedrático es mediante una plataforma en la cual todo el material de estudio se encuentra lista para ser leída y estudiada. El estudiante se debe organizar durante la semana para poder cubrir los temas propuestos por el catedrático. Durante la duración del curso se tiene pruebas cortas, exámenes y ejercicios resueltos para evaluar la calidad de lo aprendido por el estudiante y al finalizar se podrá determinar si el estudiante ha cumplido con la meta establecida por el catedrático.

2. KEIL® MDK UTILIZANDO LENGUAJE ENSAMBLADOR PARA PROCESADORES ARM® CORTEX®-M4

Arm Holdings PLC fue una compañía de computadoras británicas establecida en Cambridge, la compañía se dividió en varios operadores independientes. De esta segmentación surgió Acorn Computers la cual producía computadoras de escritorio. Con la aparición de la popular BBC Micro (una de las primeras computadoras domésticas) y en su intento de mejorar el producto, se pretendió aumentar su rendimiento utilizando un procesador secundario. El procesador propuesto fue el 6502 pero lejos de usar ese procesador Acorn decidió diseñar su propio procesador para aumentar el rendimiento de las computadoras domésticas.

Steve Furber y Sophie Wilson en octubre de 1983 empezaron a trabajar dentro de Acorn en conjunto con otra compañía llamada VLSI Technology, en abril de 1985 usando menos de 25 000 transistores llegó el procesador ARM1. Cabe destacar que los primeros procesadores embebidos típicamente trabajaban a una frecuencia de 50 Mhz a 1 Ghz, la limitación de la frecuencia de operación es debido al bajo consumo de energía.

El procesador ARM1 originalmente fue diseñado para trabajar a una frecuencia de 4 Mhz. Una revisión y mejora en el diseño de ARM1 dio como resultado el diseño de ARM2. ARM2 seguía sin tener memoria caché o memoria MMU (del inglés *Memory management unit*) pero se le agregó instrucciones de acumulación múltiple y uso de una aceleración en cálculo de datos con punto flotante lo que aumentó su rendimiento alcanzando hasta 12 Mhz.

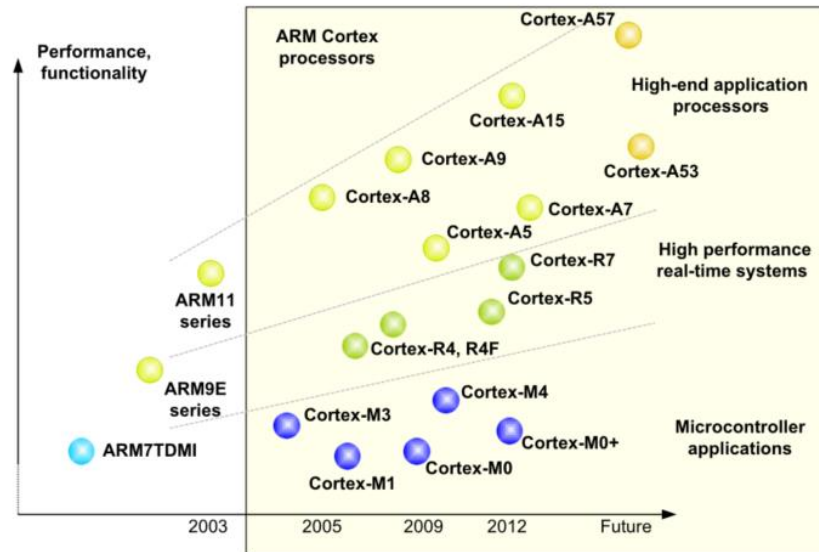
Para competir contra los procesadores que se estaban produciendo en el año 1989 como el MC68000 de Motorola, la estrategia de Acorn fue desarrollar el procesador ARM3 el cual tenía integrado un caché unificado 4k el cual trabajaba a una frecuencia de 25 Mhz. Durante estos años, a la compañía Acorn se le dificultaba permanecer en la cabeza del mercado ya que IBM PC era un fuerte competidor, por otro lado, VLSI Technology fue capaz de encontrar un mercado el cual usaría sus procesadores embebidos, una de ella fue Apple.

Apple estaba interesado en un procesador para sus nuevos dispositivos, *PDA*s (del inglés personal *digital assistants*), durante el desarrollo de éste surgió una nueva empresa (con capital de Apple, 12 ingenieros de Acron Group y herramientas proporcionadas por VLSI Technology) llamada Advanced RISC Machines Limited. En dicha empresa se le cambió el nombre a la arquitectura de los procesadores que se estaban diseñando, el nuevo nombre sería *Advanced RISC Machine* (ARM).

Advanced RISC Machines Limited desarrolló un nuevo modelo de negocios en el cual lejos de vender procesadores ellos se han dedicado a vender los derechos de los diseños de sus procesadores a otras empresas las cuales se encargan de la fabricación de sus procesadores. El primer procesador que surgió de esta nueva empresa fue el procesador con arquitectura ARM6.

Oficialmente la familia de procesadores ARM7 fue introducido al mercado en el año 1993, el diseño fue usado por Acorn para diseñar computadoras y por Psion para diseñar *PDA*s. En la figura 1 se presenta una gráfica con las diversas familias de procesadores ARM y su fecha aproximada de lanzamiento.

Figura 1. **Versiones de procesadores ARM y sus respectivas familias**



Fuente: MEDICINA TIPS. *Picobit - VM Schem.*

http://medicina.tips/picobit/images/arm_family.png. Consulta: agosto de 2020.

A inicio del año 2000 ARM introdujo al mercado nuevas líneas de procesadores, siendo estos: la familia ARM11, la familia Cortex y procesadores con múltiples núcleos y aplicaciones seguras.

Los asociados más importantes a ARM para inicios del 2002 fueron Philips, Analog Devices, LSI logic, PrairieComm y Qualcomm, estas empresas emplearon principalmente la arquitectura de la versión ARM7 como procesador primario. En 2003 ante la carencia de periféricos lógicos como puertos USB, motores gráficos, DSPs y buses especiales ARM compro Adelante Technologies, de esa forma logró solventar esos inconvenientes. En 2004 con el fin de obtener nuevas herramientas de hardware y por librerías de estándares de celdas y compiladores de memorias compro Axys Design Automation y también a Artisan Components.

En 2005 ARM compra el software Keil para usarlo como herramienta para los microcontroladores.

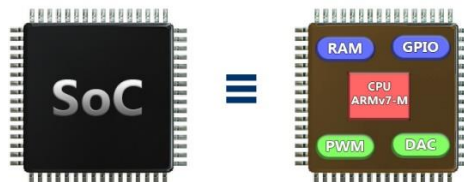
2.1. SoC (System on a Chip)

La tecnología cada vez está más presente, con el gran avance en el campo de los dispositivos móviles haciéndolos cada vez más potentes y baratos; la mayoría de las personas cuentan con un teléfono inteligente. Detrás de las pantallas o botones de dispositivos portables y de alto desempeño se encuentran placas de circuitos con dispositivos electrónicos soldados, el dispositivo más sobresaliente es el SoC (del inglés *system on a chip*).

Un SoC básicamente es la combinación de procesadores, memoria y chips gráficos interconectados que se han fabricado e implementado dentro de un paquete por lo que se obtiene un gran ahorro de energía y espacio.

Dentro de un SoC hay mucho más que un CPU, cada uno está diseñado con base en las necesidades del dispositivo final. Dentro de los SoCs se pueden encontrar procesadores con arquitectura ARM Cortex. Los procesadores con este tipo de arquitectura se usan cuando se requiere bajo consumo ya que estos procesan sets de instrucciones cortas y simples.

Figura 2. Encapsulado de un SoC y su vista interna



Fuente: elaboración propia, empleando Adobe Photoshop.

Los SoCs con el paso del tiempo han tenido grandes mejoras en cuanto procesamiento de datos y consumo de energía. Muchas empresas con el fin de ahorrarse tiempo y dinero para desarrollar desde cero uno nuevo optan por comprar las licencias a ARM Ltd., adaptarlos a sus propias necesidades y luego producirlos masivamente.

2.2. Procesador y Microprocesador

A inicios de la década de los sesenta los procesadores eran construidos empleando elementos discretos (resistores, condensadores, diodos, transistores, entre otros.). Por tal razón los procesadores eran de gran tamaño.

A finales de esa misma década, el microchip o circuito integrado (los cuales pueden contener desde unos pocos componentes discretos hasta millones de éstos) fue inventado. Todos los componentes electrónicos que formaban un procesador ahora pueden ser colocados fácilmente en una sola pieza del tamaño de una moneda. El tamaño de un procesador resultó ser miles de veces más pequeño, ahora bien y para aclarar, a un procesador con un tamaño mucho menor se le conoce como microprocesador.

2.2.1. Microprocesador

El microprocesador es un circuito electrónico que actúa como la Unidad Central de Proceso (CPU). Es un semiconductor programable el cual es usado para ejecutar instrucciones lógicas y aritméticas, los cuales procesan datos digitales (específicamente números binarios, 0 y 1) o controlar otros dispositivos.

El microprocesador reconoce y procesa un grupo de bits, este grupo de bits tiene un tamaño máximo predefinido por el fabricante. A la cantidad máxima de

bits con la que un procesador puede trabajar se le conoce como *Word* (8,16,32 o 64 bits dependiendo del modelo). Sabiendo cual es el tamaño en bits para una palabra (en inglés *word*) se puede saber cuál es el valor de una media palabra (en inglés *half-word*). El valor en bits para *half-word* es el valor de *word* dividido entre 2 como se muestra en la tabla III.

Tabla III. **Unidades de medidas en los procesadores**

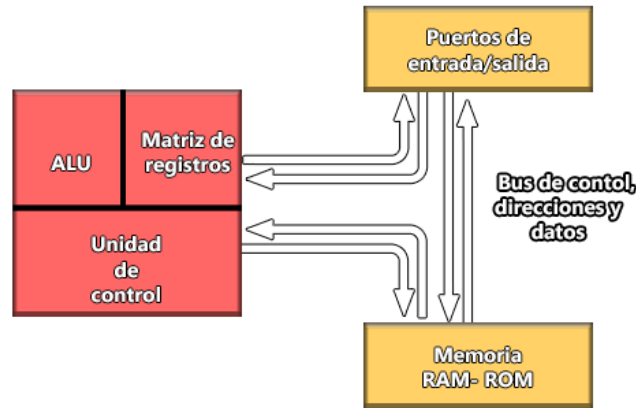
Unidad de media	Equivalente
<i>Bit</i>	0 ó 1 lógico
<i>Byte</i>	8 bits
Octeto	1 byte
<i>Nibble</i>	4 bits
<i>Word</i> para Intel 8088	8 bits
<i>Half-word</i> para Intel 8088	4 bits
<i>Word</i> para Intel 8086	16 bits
<i>Half-word</i> para Intel 8086	8 bits
<i>Word</i> para Cortex ARMv7-M	32 bits
<i>Half-word</i> para Cortex ARMv7-M	16 bits

Fuente: elaboración propia, empleando Adobe Photoshop.

Las partes básicas de un microprocesador son:

- Unidad Aritmética y lógica (ALU)
- Unidad de Control (CU)
- Registros
- Bus de Datos
- Bus de Direcciones
- Memoria

Figura 3. **Configuración interna básica de un procesador**



Fuente: elaboración propia, empleando Adobe Photoshop.

2.3. Endianness

Es la forma designada en la que se almacenan los datos de más de un bite en una memoria.

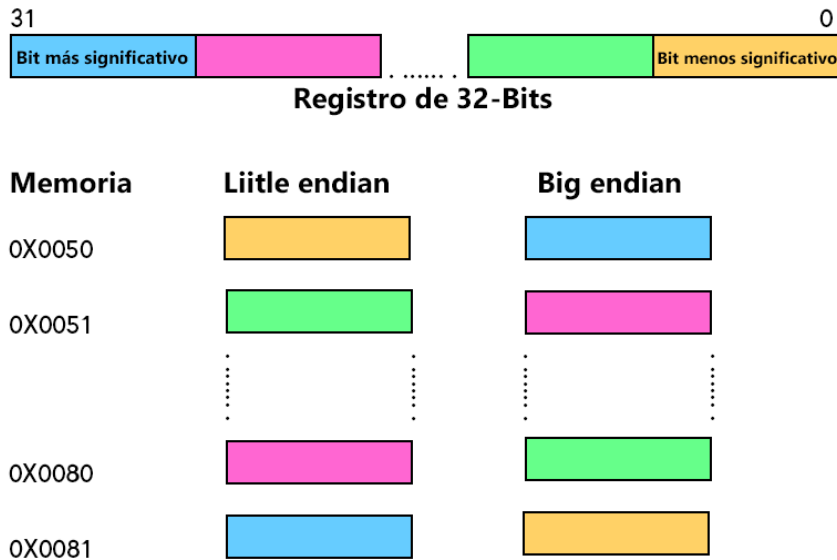
2.3.1. Big-Endian

Primero coloca el valor más significativo (o el más grande), seguido de los valores menos significativos.

2.3.2. Little-Endian

Almacena primero el valor menos significativo, seguido de valores cada vez más significativos.

Figura 4. Almacenamiento de los bits en memoria



Fuente: elaboración propia, empleando Adobe Photoshop.

2.4. Rango de direcciones para Cortex-M4

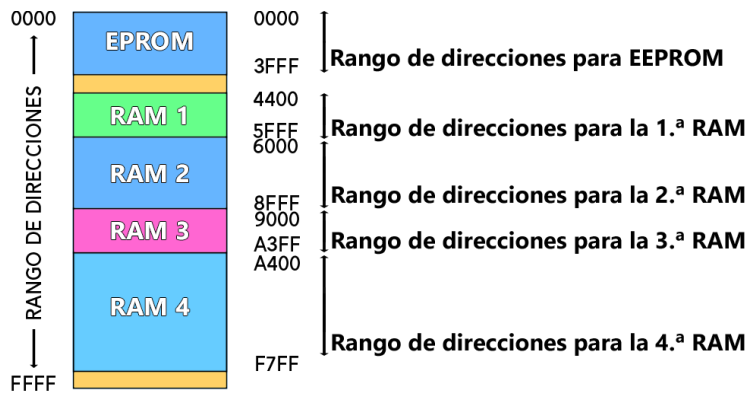
El rango de direcciones en los chips de memoria varía según el fabricante. En la figura 5 se muestra un mapa de memoria básico y genérico. Cada bloque de memoria, registro e incluso periféricos de entrada/salida están ligados a una dirección en el rango de memorias.

El microcontrolador TM4C123GH6PM tiene los siguientes tipos de memoria:

- 32 KB SRAM
- 256 KB Memoria Flash
- 2KB EEPROM
- Memoria Rom interna con el software Tivaware

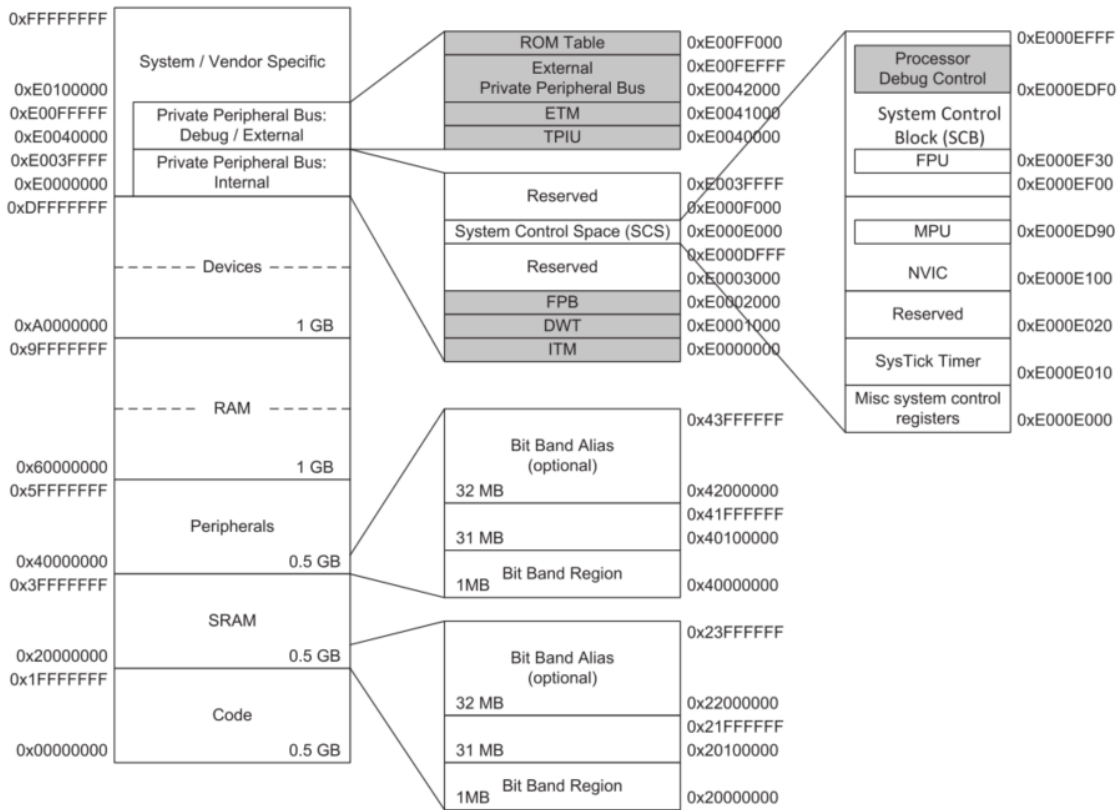
Se presenta a detalle los distintos rangos de memoria junto a su descripción, ver figura 6.

Figura 5. **Mapeado de la memoria y direcciones de un procesador**



Fuente: elaboración propia, empleando Adobe Photoshop.

Figura 6. Mapa de memoria para procesador Cortex-M4



Fuente: YIU, Joseph. *The Definitive Guide to ARM® Cortex®-M3 and Cortex®-M4 Processors*.

p. 195.

Tabla IV. **Rango de memorias para procesadores Cortex M3 y M4**

Región	Rango de dirección	Descripción
Código	0x00000000 - 0x1FFFFFFF	Memoria de 512 MB principalmente para el código del programa.
SRAM	0x20000000 - 0x3FFFFFFF	Memoria de 512 MB para conectar SRAM la cual se encuentran en chips.
Periféricos	0x40000000 - 0x5FFFFFFF	Memoria de 512 MB se utiliza para los periféricos integrados en chips.
RAM	0x60000000 - 0x9FFFFFFF	Contiene dos ranuras de 512 MB se puede utilizar para código de programa y datos.
Dispositivos	0xA0000000 - 0xDFFFFFFF	Contiene dos ranuras de 512 MB para otros periféricos internos como externos.
Sistema	0xE0000000 - 0xFFFFFFFF	Regiones varias del sistema.

Fuente: elaboración propia, empleando Microsoft Word.

2.5. Arquitectura de los procesadores

Las dos arquitecturas de los procesadores más importantes son: Risc y Cisc. Las cuales se diferencian básicamente en la forma de procesar las instrucciones.

Entre los años 1981 y 1982, David Patterson y Carlos Séquina en la Universidad de California en Berkeley, iniciaron la investigación llamada "*Design and Implementation of Risc I*"² en donde se desarrolló la idea de armar un procesador con instrucciones cortas, instrucciones más simples. En Standford

² SÉQUINA, Carlos; PATTERSON, David. *Design and Implementation of RISC I*. <https://www2.eecs.berkeley.edu/Pubs/TechRpts/1982/CSD-82-106.pdf>. Consulta: 11 de enero de 2020.

John Hennessy también inició con una investigación similar, procesadores con instrucciones simples.

Figura 7. **David Patterson and Carlos Séquina**



Fuente: BERKELEY NEWS. *Pioneer of modern computer architecture.*

<https://news.berkeley.edu/wp-content/uploads/2018/03/PattersonSequinRISC750.jpg>. Consulta: agosto de 2020.

La finalidad de la investigación fue crear procesadores con arquitectura simple. Se quería romper con las técnicas de diseño tradicional la cual era CISC (del inglés *Complex Instruction Set Computer*).

Los microprocesadores con arquitectura CISC tienen un conjunto de instrucciones amplias por lo que se logran operaciones complejas, las cuales toman lugar en memorias o en los registros internos. Una de las desventajas en este tipo de arquitectura es la limitación debido a la falta del paralelismo entre instrucciones.

Se logró obtener procesadores con nueva arquitectura, arquitectura RISC, los cuales tienen las siguientes características:

- Todas las instrucciones son del mismo tamaño, de un tamaño fijo.
- Las instrucciones son fáciles para decodificar.
- Todas las instrucciones están libres de microcódigo.
- Orientado al software.
- Instrucciones simples.
- Ejecución de una instrucción por ciclo de reloj.
- Requiere muchas líneas de código.

2.5.1. RISC

Los procesadores con arquitectura RISC (del inglés *Reduced Instruction Set Computer*) se usan en dispositivos portables ya que consumen poca energía, un ejemplo de esta arquitectura se puede encontrar en el procesador ARM Cortex M4 el cual es licenciado por ARM Holdings Limited.

Cada instrucción se completa en un ciclo de reloj (cada línea de código corresponde a un ciclo de reloj). En la arquitectura RISC se posibilita la segmentación y el paralelismo en la ejecución de instrucciones.

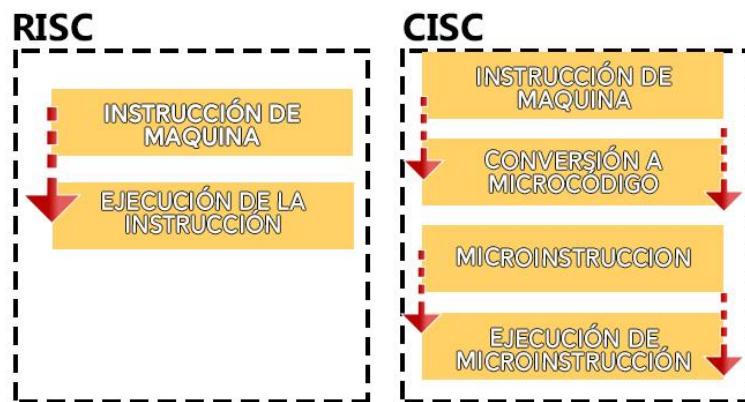
2.5.2. CISC

Los procesadores con arquitectura CISC (del inglés *Complex instruction set computer*) poseen instrucciones muy complejas en pocas líneas de código y, por consiguiente, se consigue completar tareas complejas entre datos almacenados en memoria y/o en los registros en menos líneas de código. Este tipo de

arquitectura dificulta el paralelismo entre instrucciones. Características más importantes:

- Orientado en hardware
- Contiene múltiples relojes
- Las instrucciones pueden llevar varios ciclos de reloj para ser ejecutados
- Requiere pocas líneas de código y la decodificación de la instrucción es compleja

Figura 8. **Arquitectura RISC vs CISC**



Fuente: elaboración propia, empleando Adobe Photoshop.

2.6. Von Neumann & Harvard

La distribución interna de los equipos informáticos, computadoras o microcontroladores, suele ser distinta con base en las conexiones físicas que tiene el procesador hacia los distintos tipos de memorias.

Los procesadores deben poder acceder a las memorias para poder realizar las tareas propuestas por el usuario. La distribución de memorias y procesador

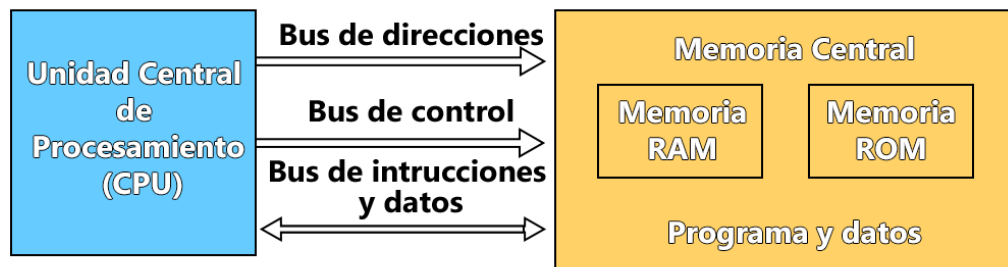
impactan directamente en el rendimiento del equipo. En la arquitectura Von Neumann y Harvard se puede apreciar de gran manera como las memorias, buses y procesadores pueden ser distribuidos, las ventajas y desventajas que se logran.

2.6.1. Arquitectura Von Neumann

En este tipo de arquitectura la memoria almacena instrucciones y datos. El procesador no puede acceder a las instrucciones almacenadas en la ROM y los datos almacenados en la RAM simultáneamente, desventaja importante ante la arquitectura Harvard.

Al solicitar simultáneamente instrucciones y los datos en memoria ocurre un evento llamado “Cuello de botella”, el cual afecta el rendimiento del sistema. Este problema se atenuó introduciendo una memoria de tipo caché entre el procesador y la memoria principal.

Figura 9. Diagrama de la arquitectura Von Neumann

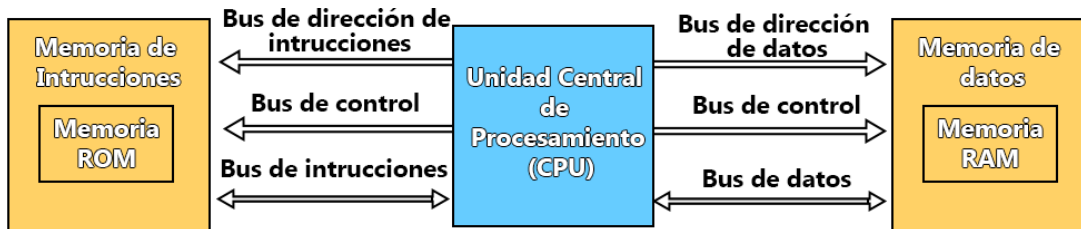


Fuente: elaboración propia, empleando Adobe Photoshop.

2.6.2. Arquitectura Harvard

En este tipo de arquitectura las instrucciones y los datos son almacenados en distintas memorias usando buses diferentes. Con lo que se obtiene la posibilidad de ejecutar instrucciones y acceder a datos simultáneamente, ver figura 10.

Figura 10. Diagrama de la arquitectura Harvard



Fuente: elaboración propia, empleando Adobe Photoshop.

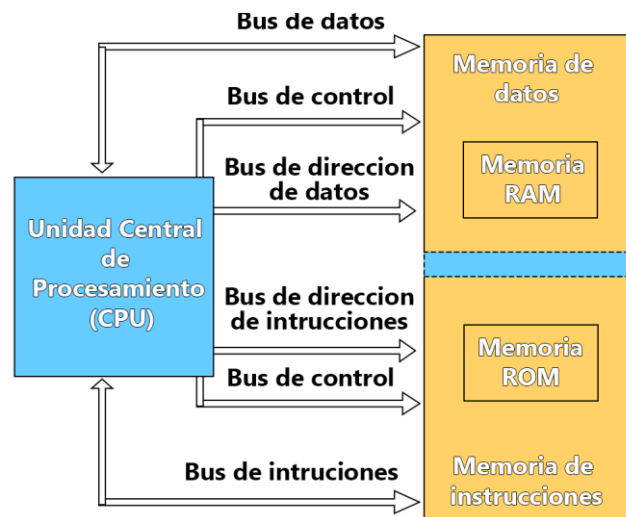
2.6.3. Arquitectura Harvard modificada

Es un tipo de arquitectura en donde las memorias (RAM Y ROM) se encuentran en un mismo encapsulado, pero perfectamente divididas, como se observa en la figura 11. Por lo que en este aspecto se considera ser similar a la arquitectura Von Neumann.

En este caso las memorias se encuentran en espacios físicos diferentes. Las cuales poseen diferentes buses e instrucciones especiales para que los datos no sean confundidos con las instrucciones, a ese tipo de arquitectura se llama Harvard modificada.

Simultáneamente puede leer o escribir datos en una dirección y puede obtener instrucciones desde otra memoria. Se agregó una memoria caché de instrucciones para mejorar el rendimiento comparado con la arquitectura Harvard.

Figura 11. Diagrama de la arquitectura Harvard modificada



Fuente: elaboración propia, empleando Adobe Photoshop.

2.7. La Familia Cortex

Los mercados han requerido distintas soluciones para poder construir dispositivos electrónicos de alta calidad, por esta razón ARM Holdings Limited decidió diversificar sus procesadores en tres diversas familias a las cuales se les llamó Cortex estas son:

- Cortex-A
- Cortex-R
- Cortex-M

2.7.1. Cortex A

La familia Cortex-A lleva su nombre por “Aplicación”, diseñado específicamente para electrónica de consumo como smartphone, tablets, servidores, Smart TV y otros productos que requieren alto poder en rendimiento.

Estos procesadores tienen las características necesarias para poder brindar soporte a sistemas operativos como Linux, Windows y otros. Los procesadores de la familia Cortex-A tiene dos subcategorías las cuales son:

- Procesadores de 32-bits en donde se incluyen los Cortex A5, A7, A8, A9, A12 y A15.
- Procesadores de 64-bits en donde se incluyen los Cortex A57 y A53.

2.7.2. Cortex R

La familia Cortex-R se caracteriza por ejecutar aplicaciones de control en “Real-Time” (tiempo real), de este término su nombre. En esta familia se pueden mencionar los Cortex R4, R5 y R7.

Son diseñados para aplicaciones en donde se tiene prioridad la seguridad de los datos y el procesamiento de éstos en tiempo real. Se encuentran en aplicaciones que requieren procedimientos críticos como en un sistema automotriz o sistema médicos, se ejecutan acciones sin errores y de forma inmediata, no puede existir un retraso en la manipulación de datos o datos corruptos. La solución a estos posibles problemas es usar sistemas con redundancia los cuales poseen más de un procesador.

2.7.3. Cortex M

La familia Cortex-M por “Microcontrolador”, diseñado especialmente para ser usado dentro de un microcontrolador, comúnmente está súper embebido en el sistema que muchas veces no se nota y comúnmente son de 32-bits. Dentro de esta familia se pueden encontrar los Cortex M0, M0+, M1, M2, M3 y M4.

Cortex-M0 y Cortex-M0+ son los más pequeños en esta familia. El modelo más básico es el Cortex-0 el cual contiene un solo núcleo, un NVIC (del inglés *nested vectored interrupt controller*), un bus y un set de 56 instrucciones.

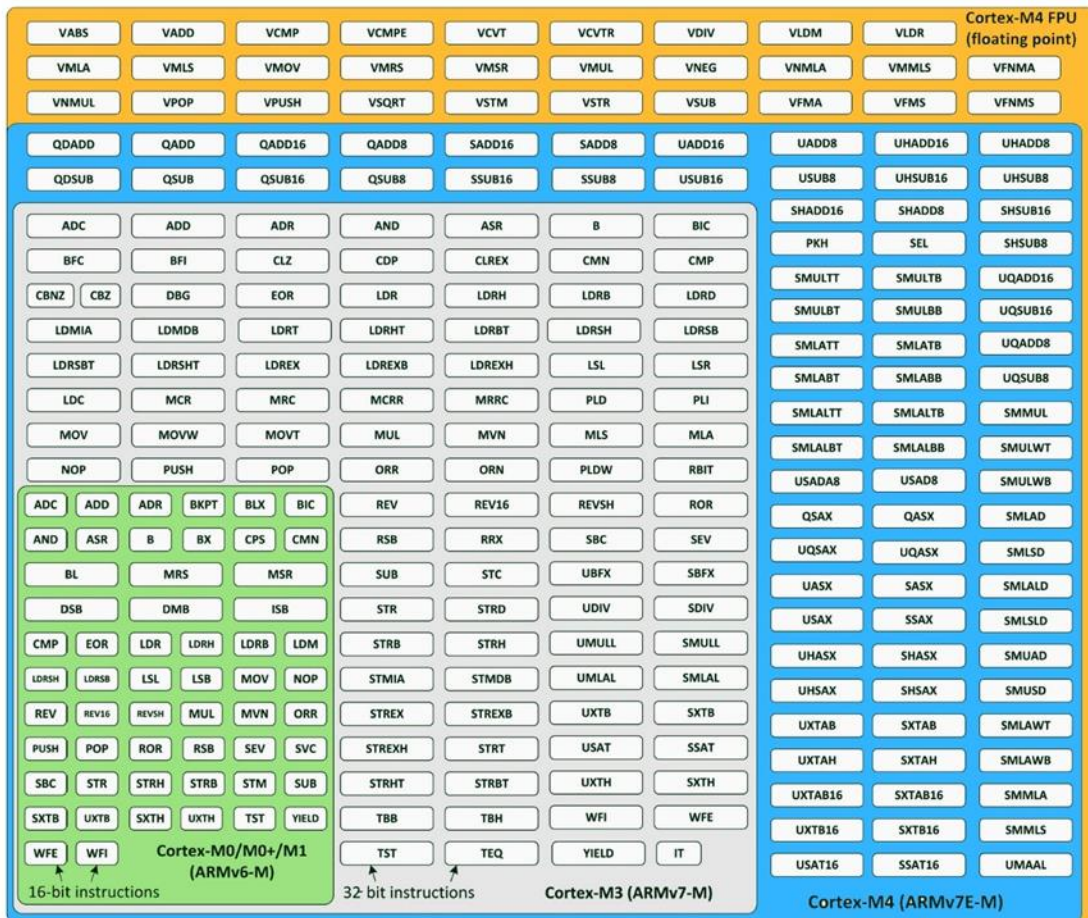
El Cortex-M0+ similar a Cortex-M0 pero con la diferencia que M0+ implementa MPU (del inglés *memory protection unit*), una tabla de vectores localizables, interfaces de entrada/salida controlada en un solo ciclo de trabajo y una debug (depuración) mejorada.

Ahora bien, el Cortex-M1 fue diseñado para implementaciones con FPGA (del inglés *Field-programmable gate array*). Para aplicaciones que requieren alto desempeño y bajo consumo de energía se diseñó el Cortex-M3, en el año 2003, el cual fue el primer procesador de esta familia en salir al mercado, está presente en diversos sistemas automotrices, sistemas de control industriales, redes inalámbricas y en sensores.

Finalmente, el microprocesador Cortex-M4, diseñado para mercados que demandan alta eficiencia, fácil de usar y alto procesamiento de datos, en si el Cortex-M4 es una mejora del Cortex-M3. Los Cortex-M4 son utilizados en controladores de motores, equipos de administración de energía, equipos de audio y sistemas automotrices industriales.

En la figura 12 se muestran las distintas instrucciones presentes en los procesadores de la familia Cortex-M.

Figura 12. Instrucciones en la familia Cortex-M



Fuente: YIU, Joseph. *The Definitive Guide to ARM® Cortex®-M3 and Cortex®-M4 Processors*. p. 158.

Como se observa en el área verde de la figura 12 las instrucciones son designadas para el procesamiento general de datos y para el control de tareas de entrada/salida. El área gris contiene instrucciones para procesamiento avanzado, para manipulación de campos de bit y MAC (del inglés *Multiply-*

Accumulate). El área celeste contiene las instrucciones dedicadas para DSP (del inglés *digital signal processor*) en las que se incluyen lo siguiente: SIMD, MAC rápida y saturación aritmética. Finalmente, el área naranja muestra las instrucciones dedicadas para operaciones con punto flotante.

2.8. Cortex M4

Cabe resaltar que los Cortex-M4 son una unidad completa de microcontrolador (en inglés MCU, Complete Microcontroller Unit) ya que no solo cuenta con el núcleo de la CPU.

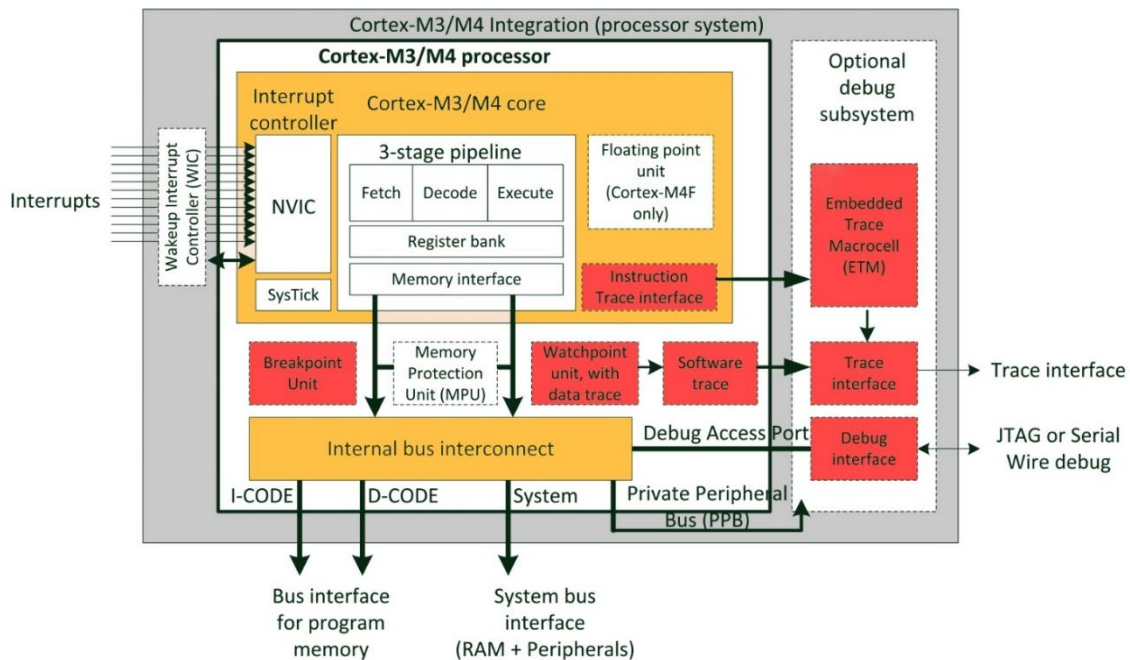
Los Cortex-M4 cuentan con una interfaz estándar de bus, una arquitectura de depuración (en inglés llamado *debug*), CPU, estructura de interrupciones, control de energía y protección de memoria. La programación en los procesadores de la familia Cortex-M4 es igual en todos los fabricantes, por lo que al aprender a programar un procesador basado en Cortex-M4 se podrá usar el mismo aprendizaje y aplicarlo en microcontrolador basado en Cortex-M4 sin importar quien lo fabricó. En lo único en que varían los Cortex-M4 es en la cantidad disponible de memoria, cada fabricante designa la memoria en función de sus propias necesidades.

La razón de la variación en la cantidad de memoria es debido a que los Cortex-M4 vienen con una interfaz de bus genérica en chip, entonces cada fabricante puede añadir su propio sistema de memoria a los Cortex-M4. El protocolo de interfaz de bus principal usado en los procesadores Cortex M3 y M4 es *AHB Lite* (del inglés *Advanced High-performance Bus Lite*). El *AHB Lite* es un protocolo tipo *Pipeline* lo cual le permite alta frecuencias de operación y bajos costos en cuanto a hardware. Otro protocolo usado en los microcontroladores

basados en ARM es el protocolo APB (del inglés *Advanced Peripheral Bus*) el cual brinda soporte para la depuración (*debug*).

Como se observa en el diagrama de bloques de la figura 13, el procesador cuenta con varios tipos de interfaz de bus. En la tabla V se explica la función de cada una de sus interfaces de bus.

Figura 13. Diagrama de bloques para Cortex®-M4 y M3



Fuente: YIU, Joseph. *The Definitive Guide to ARM® Cortex®-M3 and Cortex®-M4 Processors*.

p. 62.

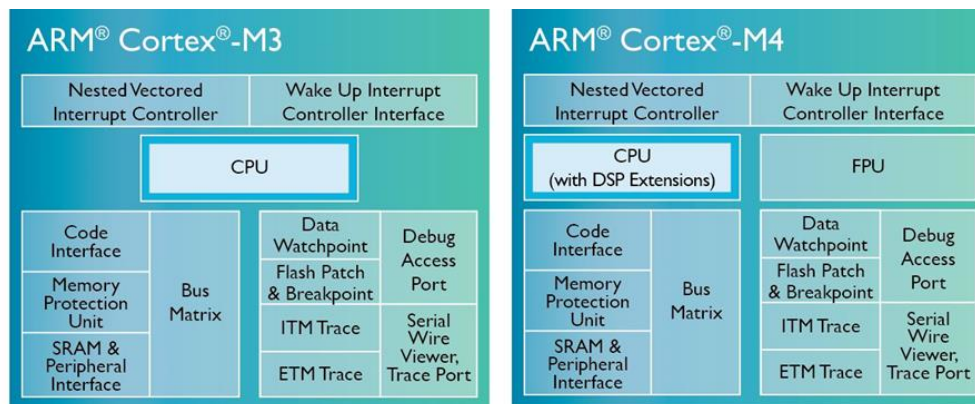
Tabla V. **Interfaces de bus en procesadores Cortex® M3 y M4**

Interfaz de Bus	Descripción
I-CODE	Se usa principalmente para programar la memoria: búsqueda de instrucciones y búsqueda de vectores para las direcciones 0x0 a 0x1FFFFFFF.
D-CODE	Se usa principalmente para programar la memoria: acceso a los datos y depurador para las direcciones 0x0 a 0x1FFFFFFF.
Sistema	Se usa principalmente para la RAM y periféricos: cualquier acceso en el rango de direcciones de 0x20000000 a 0xFFFFFFFF.
PPB	Bus periférico privado externo: Para componentes privados de depuración del nivel del sistema desde la dirección 0xE0004000 a 0xE00FFFFF.
DAP	Puerto de acceso al depurador.

Fuente: elaboración propia, empleando Microsoft Word.

Cortex-M4 es una mejora del Cortex-M3 por tal razón comparten muchas características, se presenta el diagrama de estos procesadores en la figura 14.

Figura 14. **Diagrama de un Cortex-M3 y M4**



Fuente: ARM DEVELOPER. *ARM Developer*. <https://developer.arm.com/ip-products/processors/cortex-m/cortex-m4>. Consulta: agosto de 2020.

Entre Las características extras en el Cortex-M4 comparado con el Cortex-M3 están el soporte de algoritmo DSP (del inglés *digital signal processing*) y de la unidad de punto flotante (FPU del inglés *Floating Point Unit*). El Cortex-M4 posee instrucciones basadas en DSP en forma de instrucciones SIMD (del inglés *Single Instruction, Multiple Data*). El hardware de acumulación múltiple (MAC) también se ha mejorado para que muchas instrucciones aritméticas de $32 * 32$ tengan un solo ciclo.

2.8.1. Instrucciones DSP

El Cortex-M4 tiene un conjunto de instrucciones SIMD destinadas a admitir algoritmos DSP. Estas instrucciones permiten realizar varias operaciones aritméticas paralelas en un solo ciclo de procesador. Las instrucciones SIMD funcionan con 16 u 8 bits que se empaquetan en instrucciones con tamaño de 32 bits. Las instrucciones SIMD se pueden usar para mejorar enormemente el rendimiento de los algoritmos DSP, como los filtros digitales, que básicamente realizan muchos cálculos de multiplicación y suma en un pipeline de datos.

Los algoritmos que se implementan en el módulo DSP son transformadas matemáticas como, por ejemplo:

- La transformada rápida de Fourier (FFT)
- Filtros digitales como los filtros de respuesta finita al impulso (FIR)
- Algoritmos de lazo de control como un controlador proporcional, integral y derivativo (PID).

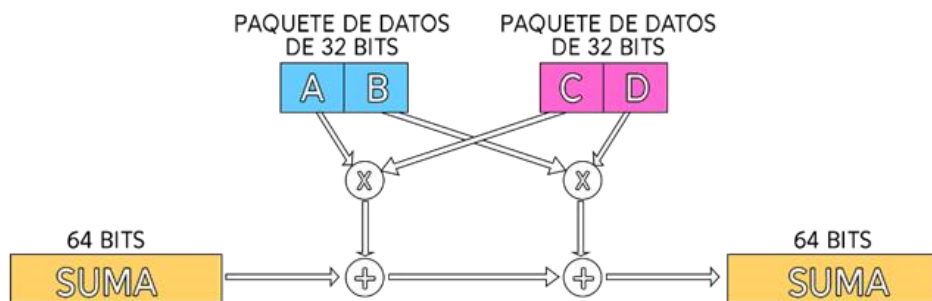
2.8.2. Unidad de punto flotante

El procesador Cortex-M también puede estar equipado con una FPU de hardware. La elección se realiza en la etapa de diseño por parte del proveedor del microcontrolador. La FPU admite cálculos aritméticos de punto flotante de precisión simple utilizando el estándar IEEE 754.

El hardware dedicado a punto flotante logra realizar una operación en un solo ciclo de reloj logrando un aumento en el rendimiento del microprocesador.

La FPU puede considerarse como un coprocesador que se encuentra junto a la CPU del Cortex-M4. Cuando se realiza un cálculo con números con decimales, los valores de punto flotante se transfieren directamente desde los registros de FPU hacia y desde el almacén de memoria SRAM, sin la necesidad de utilizar los registros de la CPU central.

Figura 15. Instrucción SIMD



Fuente: elaboración propia, empleando Adobe Photoshop.

2.9. Lenguaje de bajo nivel

La tarjeta perforada fue implementada por primera vez por Joseph Marie Jacquard en los telares en 1804. Esas tarjetas fueron modificadas y usadas para introducir información e instrucciones a computadoras eléctricas desde los años 1960. Estas tarjetas eran perforadas en función del código binario requerido y colocadas en un orden preciso, si se perdía el orden o se extraviaba una tarjeta se necesitaba perforar un conjunto nuevo en el orden correcto. Estas tarjetas eran poco eficientes debido a que se necesitaban cientos de tarjetas para poder ejecutar programas cortos.

Figura 16. Tarjeta perforada



Fuente: TWO-BIT HISTORY. *The IBM 029 Card Punch*.
<https://twobithistory.org/images/card.png>. Consulta: agosto de 2020.

2.9.1. Bits a Comandos

Todos los procesadores son programados con sets de instrucciones. Las instrucciones son patrones de unos y ceros, el patrón es único para cada instrucción. Cada set de instrucciones es único para cada tipo de procesador.

Leer y escribir los unos y ceros para añadir una instrucción es muy complicado como se vio en las tarjetas perforadas, como ayuda para la programación se crearon patrón de unos y ceros los cuales son mapeados como un nombre llamado mnemónico.

El término *opcode* es un mnemónico usado para referirse a una instrucción en lenguaje ensamblador. Ejemplos de *opcodes* o mnemónicos son: add, mov, sum, entre otros.

Tabla VI. **Representación de mnemónicos para Intel i386**

Código binario	Código hexadecimal	Código ASCII	Nemónico	Acción
101000	50	P	ADD	Suma al acumulador
10110001	B1	±	SUB	Resta al acumulador

Fuente: elaboración propia, empleando Microsoft Word.

Con el paso del tiempo la programación se ha vuelto más sencilla. Programas como C/C++, Python, Perl y otros, los cuales necesitan poder traducir las instrucciones a un set de instrucciones nativo para el microprocesador. El programa encargado de hacer esa tarea se llama compilador.

El compilador toma instrucciones como *if*, *while* y *else*, y posteriormente los convierte en lenguaje ensamblador. Los programas en lenguaje ensamblador producen archivos que el hardware directamente puede comprender, usan el lenguaje de máquina unos y ceros.

2.10. Lenguaje ensamblador

Es un lenguaje de programación de bajo nivel, (*assembler* en inglés). Este lenguaje facilita la programación, ya que dispone de código simbólico para reemplazar los largos sets de instrucciones escritos puramente con 0 y 1 (lenguaje de máquina) y además le da formato al código.

2.10.1. Jerarquía de una computadora

Para tener una mejor idea del lugar donde se encuentra lo que se programa en *assembler* se debe tomar en cuenta la jerarquía de la computación. Como se observa en la figura 17 el nivel más bajo está compuesto puramente de transistores y resistencias los cuales se encargan de transportar las señales eléctricas, funcionan como interruptores plenamente controlados.

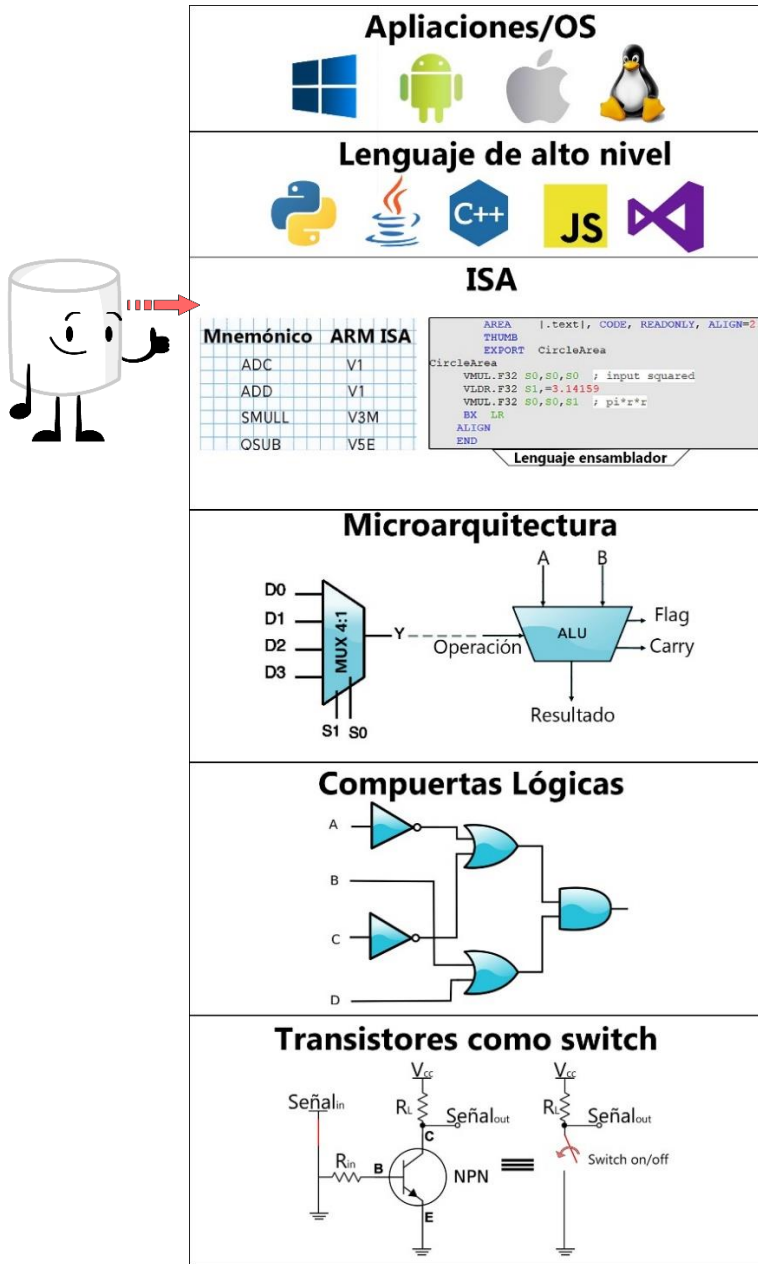
Los interruptores hechos con los transistores se usan para construir compuertas lógicas en el nivel superior. Las compuertas lógicas a su vez se usan para implementar sumadores completos, multiplicadores, multiplexores, memorias, registros y demás, los elementos básicos para crear la arquitectura de un procesador. Las arquitecturas de los procesadores varían en función del fabricante. La arquitectura especifica cómo se deben procesar los datos, cómo se controla la memoria y cómo las interrupciones van a ser controladas. Cada procesador tiene un lenguaje propio en función de la configuración de sus componentes internos. Con todo el conjunto de componentes que conforman el nivel llamado Microarquitectura ya se posee todo lo necesario para poder programar los circuitos. Ahora bien, en el siguiente nivel llamado ISA se encuentra el lenguaje y especificaciones necesarias para poder controlar todos los circuitos dentro de una computadora.

ISA (del inglés Instruction Set Architecture) se puede definir como un modelo abstracto el cual define a una máquina de computación ante un programador o compilador basado en el lenguaje de máquina, este nivel es el más importante para el presente trabajo de investigación. El término ISA engloba las instrucciones, tipo de datos aceptados de forma nativa, *endianess*, tamaño y cantidad de registros que un CPU puede entender y ejecutar.

Tome en cuenta que existen varios tipos de ISAs. Una ISA contendrá un determinado número de instrucciones (*mov, add, sub, not, and, or*, entre otros.) que el programador tiene disponible para crear códigos en ensamblador.

Un nivel superior al conjunto de instrucciones ISA puede ser el lenguaje C o C++, luego en el nivel superior a todos se encuentra el sistema operativo el cual ejecuta tareas o aplicaciones cuando son requeridas.

Figura 17. Jerarquía de las computadoras



Fuente: elaboración propia, empleando Adobe Photoshop.

2.11. Arquitectura y lenguaje ensamblador

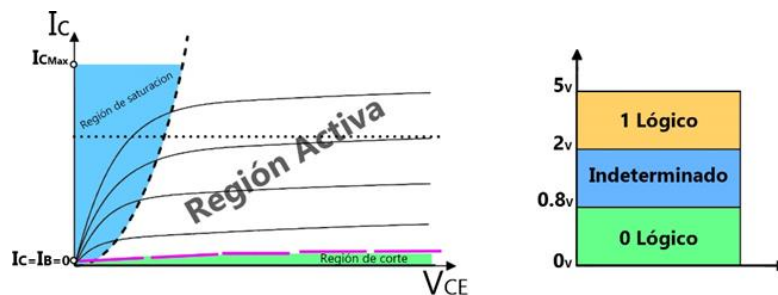
El lenguaje ensamblador depende plenamente de la arquitectura del procesador con el que se esté trabajando. Por esa razón el estudio del lenguaje ensamblador y del microprocesador van de la mano.

La ventaja más grande al programar en lenguaje ensamblador es la posibilidad de escribir números en decimal, hexadecimal y por supuesto en binario. El programa escrito en ensamblador posteriormente es convertido al lenguaje de máquina.

2.11.1. Sistema numérico

Como se muestra en la figura 17 los procesadores operan internamente con transistores utilizados como interruptores. Estos interruptores pueden tener dos estados: encendido o apagado, esos estados pueden representarse mediante 0 lógico o 1 lógico. El estado depende plenamente de la región de operación del transistor, las cuales pueden ser corte o saturación, como se muestra en la figura 18.

Figura 18. Regiones de operación del transistor



Fuente: elaboración propia, empleando Adobe Photoshop.

El sistema de numeración binario o de base dos es el ideal para poder controlar la electrónica dentro de los procesadores. El sistema binario al igual que los procesadores solo operan con dos estado o números los cuales son 0 y 1.

2.12. Sistema de numeración

En la numeración binaria cada dígito es representado por una potencia de dos, a diferencia del sistema decimal el cual cada dígito es representado por una potencia de 10. Ahora bien, el último sistema numérico para tener en consideración es el sistema hexadecimal en el cual cada dígito representa una potencia de 16.

2.12.1. Sistema de numeración decimal

El sistema decimal o de base 10, utiliza diez dígitos: 0, 1, 2, 3, 4, 5, 6, 7, 8 y 9. El sistema decimal es posicional porque el valor de los números se obtiene multiplicando cada coeficiente o dígito por la potencia de 10 que le corresponda según la posición del dígito del número.

El número 2 022 es equivalente a decir 2 millares más 2 decenas, más 2 unidades. Para ser más exactos, podemos escribirlo como se observa en la figura 19.

Figura 19. **Representación del sistema decimal**

Posición	3	2	1	0
Base	10^3	10^2	10^1	10^0
Número Decimal	2	0	2	2

$$2x10^3 + 0x10^2 + 2x10^1 + 2x10^0 = 2,022_{10}$$

Fuente: elaboración propia, empleando Adobe Photoshop.

2.12.2. Sistema de numeración binario

El sistema binario o de base 2 cuenta con solo dos dígitos 1 y 0. Cada coeficiente se debe multiplicar por la potencia de 2 correspondiente a su posición para poder representarlo en el sistema decimal.

Figura 20. **Representación del sistema binario**

Posición	3	2	1	0
Base	2^3	2^2	2^1	2^0
Número Binario	1	1	0	1

$$1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 = 13_{10}$$

Fuente: elaboración propia, empleando Adobe Photoshop

2.12.3. Sistema de numeración hexadecimal

El sistema hexadecimal o de base 16, cuenta con 16 caracteres (números y letras): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E y F. Similar a los dos casos anteriores cada carácter se multiplica por la potencia 16 correspondiente a la posición.

Figura 21. Representación del sistema hexadecimal

Posición	3	2	1	0
Base	16^3	16^2	16^1	16^0
Número hexadecimal	B	E	0	2

$$11x16^3 + 14x16^2 + 0x16^1 + 2x16^0 = 48,642_{10}$$

Fuente: elaboración propia, empleando Adobe Photoshop.

En la figura 22 se presenta una tabla en donde se muestran los primeros 16 caracteres para el sistema binario, decimal y hexadecimal, y se ordena de tal forma para hacer evidente sus equivalencias en cada sistema numérico.

Figura 22. Sistema decimal, binario y hexadecimal

Decimal (base 10)	Binario (base 2)	Hexadecimal (base 16)
00	0000	0
01	0001	1
02	0010	2
03	0011	3
04	0100	4
05	0101	5
06	0110	6
07	0111	7
08	1000	8
09	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Fuente: MORRIS, Mano. *Sistemas binarios Diseño digital*. p. 8.

2.13. Software de desarrollo para microcontroladores basados en ARM

En el mercado digital existen varios entornos de desarrollo para poder programar los procesadores con arquitectura ARM, usando lenguaje *assembler*, uno de los más populares es Keil® MDK-ARM (del inglés *Microcontroller Development Kit for ARM*) y OpenSTM32.

El presente trabajo se enfocará en Keil® MDK-ARM, el cual contiene un conjunto de componentes:

- Entorno de desarrollo integrado uVision® (IDE)
- Herramientas para compilar ARM:
 - Compilador de C/C++

- Compilador de ensamblador
- *Debugger*
- Simulador
- Ejemplos de programas

Para iniciar en el mundo de la programación de procesadores con arquitectura ARM el entorno de desarrollo ideal es Keil® MDK-ARM. Con Keil® MDK-ARM no es necesario contar con un microcontrolador físico conectado a nuestra computadora. El entorno de desarrollo uVision® cuenta con un simulador de sets de instrucciones, el simulador permite probar los programas sin la necesidad de cargarlos en la tarjeta física.

El debugger dentro de Keil® MDK-ARM puede ser usado con distintos adaptadores de *debug* como, por ejemplo: Keil ULINK2, Signum Systems JTAGjet, Stellaris® ICDI y otros.

Figura 23. Equipos Ulink

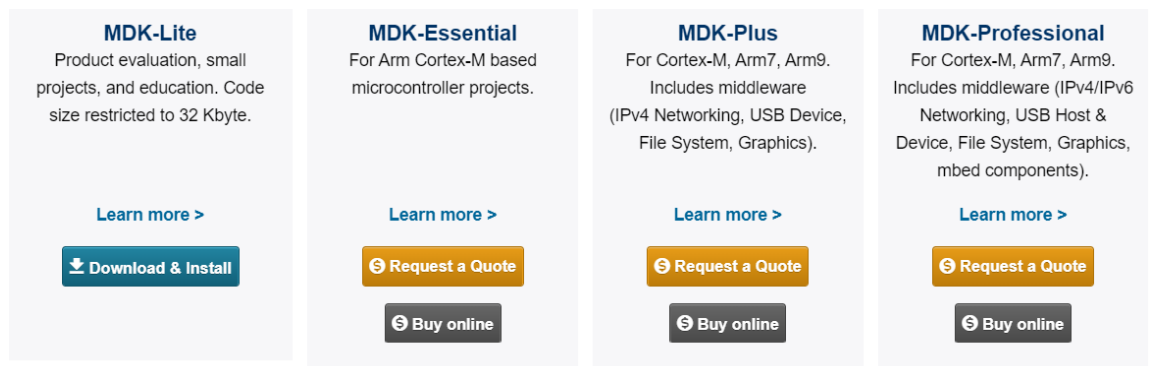


Fuente: ARM KEIL. *ULINK debug and trace adapters*. https://www2.keil.com/images/default-source/mdk5/ulinkpro_10perc.png?sfvrsn=2. Consulta: agosto de 2020.

Cabe destacar que los entornos de desarrollo para procesadores Cortex®-M son de pago. Keil® brinda una versión *Lite* la cual es totalmente gratuita. Esta

versión brinda un entorno de desarrollo bastante completo sin límite de tiempo de uso, la única limitante es el tamaño total del programa el cual no puede ser mayor de 32 Kb.

Figura 24. Versiones de **MDK** disponibles para descargar



Fuente: ARM KEIL. *MDK Microcontroller Development Kit.*

https://www2.keil.com/images/default-source/mdk5/mdk_v526_armds_673x323.png. Consulta: agosto de 2020.

2.13.1. Keil uVision®

Es el entorno de desarrollo brindado por Keil® MDK-ARM, esta herramienta brinda un soporte completo para la tarjeta de desarrollo TM4C123x, desarrollada por Texas Instruments. Actualmente la versión más actualizada es Keil uVision® 5. Esta última versión comparada con Keil uVision® 4 en cuanto a la programación son totalmente idénticas.

En la versión 5 se han agregado paquetes nuevos que, en este caso, al usar la tarjeta TM4C123x son irrelevantes. Para poder mantener la compatibilidad dentro de Keil uVision® 5 con proyectos previamente creados en Keil uVision 4 se requiere la instalación del paquete llamado *Legacy Support*. Otra de la razón

de esta instalación es debido a que algunos microcontroladores solo son compatibles con Keil uVision 4.

Keil uVision® 4 y 5 presenta dos modos básicos de trabajo:

- Modo principal o de proyecto
- Modo de depuración

2.13.2. Componentes básicos de Keil uVision®

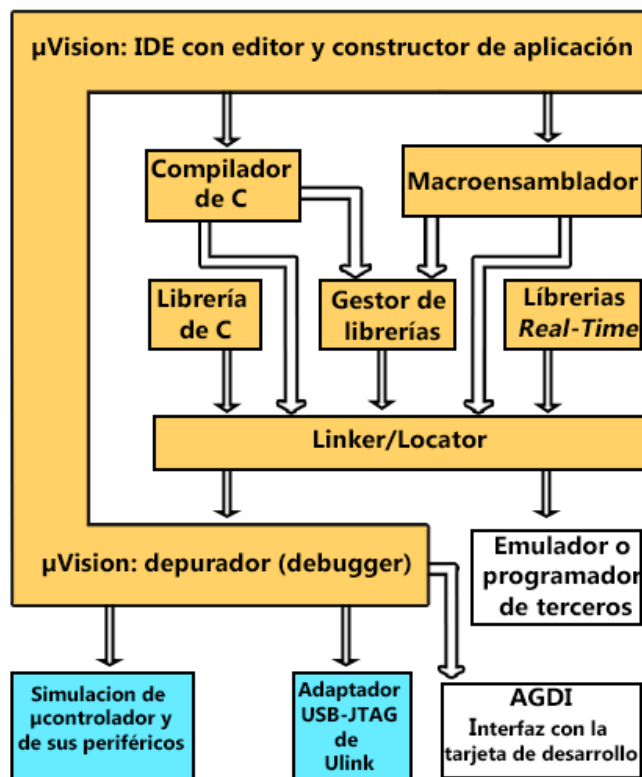
- Integrated Development Environment (IDE) o también llamado entorno de desarrollo integrado (EDI) de uVision® tiene las siguientes características:
 - Gestiona los proyectos
 - Editor dedicado y capaz de detectar errores
 - Opciones de ajustes
 - Herramienta de construcción de la aplicación
 - Ayuda en línea
- Compilador
 - MDK incluye un compilador ARM C/C++ en conjunto con ensamblador, *linker* y biblioteca optimizados con bajo tiempo de respuesta, se adaptan para el mejor tamaño y rendimiento de códigos.
- Depurador (*debugger*)
 - Se trata de un depurador simbólico a nivel de código, como su nombre lo indica sirve para depurar rápidamente un programa. El depurador incluye un simulador de alta velocidad capaz de simular hardware externo y muchos periféricos dentro de los SoCs. Lo anterior se puede realizar de las siguientes formas:
 - Conectando un adaptador Keil ULINK USB-JTAG o Stellaris® ICDI, el cual permite descargar el programa a la

memoria flash del microcontrolador y depurar el programa sobre dispositivos ARM.

- Conectando una interfaz AGDI, la cual permite conectar y usar el depurador de uVision® sobre el sistema objetivo.

En la figura 25, se presenta un diagrama de bloques básico en donde se puede apreciar los componentes básicos de Keil® MDK ARM.

Figura 25. Diagrama del funcionamiento de Keil® MDK-ARM



Fuente: elaboración propia, empleando Adobe Photoshop.

2.13.3. Instalación del software de desarrollo Keil® MDK

El software de desarrollo es totalmente gratuito, para descargarlo es necesario dirigirse a la página oficial de Keil: <https://www.keil.com/>.

Primeramente, se procederá con la instalación de Keil uVision®4, la versión oficial del año 2014. Keil® MDK ha sido actualizado con el paso de los años y seguirá siendo actualizado, esto debido al desarrollo en la tecnología. Actualmente en la página oficial en el apartado de descargas aparece la versión de Keil® MDK más reciente. Con el fin de ayudar al estudiante a mantener el software de Keil actualizado se procederá posteriormente a la instalación Keil uVision®5, versión oficial del año 2021.

Entre la versión 4 y 5 de Keil uVision® no existe gran diferencia con base en la programación en procesadores con arquitectura ARM® Cortex-M. En ambas versiones se logra programar el procesador sin problemas, en dado caso se haya empezado a programar el procesador con la versión 4 y posteriormente se quiere continuar en la versión 5 se requiere la instalación del paquete *MDK v4 Legacy Support*.

Al instalar el paquete anterior, se tendrán las siguientes ventajas:

- Mantener la compatibilidad con proyectos creados con MDK versión 4, sin tener que realizar el proyecto desde cero.
- Utilizar dispositivos más antiguos que no son compatibles con un paquete familiar de dispositivos.

Queda a discreción del estudiante la versión a instalar. Las diferencias de la versión 4 ante la versión 5 son (mejoras agregadas al software):

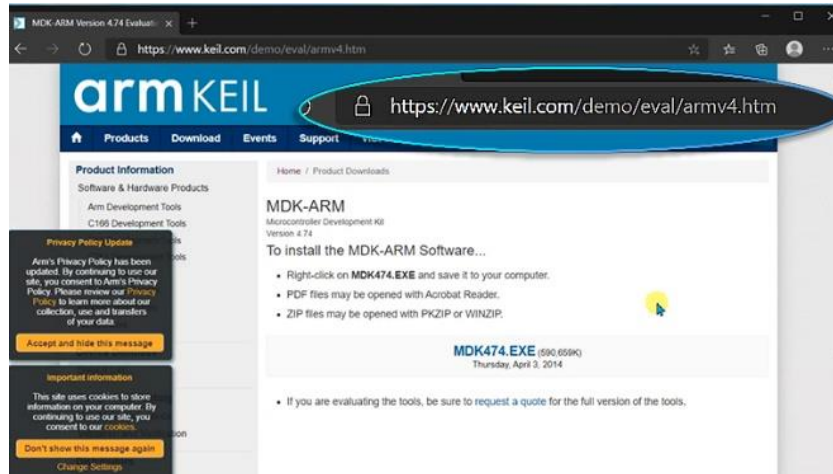
- Optimización en la densidad del código.
- Soporte para Atmel, Atmel WM y Philips con las directivas MODA2 y MODP2.
- Las variables automáticas como son: *data*, *pdata* y *xdata*; se superponen en todos los modelos de memoria.
- El tipo de datos se ajusta automáticamente a 8 o 16 bits.
- Funciones nuevas en biblioteca *modf*, *strtod*, *strtol* y *strtoul*.
- Nuevas directivas agregadas: BROWSE, INCDIR, ONEREBANK, RET_XSTK y RET_PSTK.

2.13.4. Instalación de Keil Uvision®4

Se procede a ingresar a la página web oficial de Keil, en la sección de descargas aparece la versión más actualizada hasta el momento, esa versión no es requerida por el momento, para poder descargar la versión 4 se debe ingresar al siguiente enlace: <https://www.keil.com/demo/eval/armv4.htm>.

Proceda a descargar el instalador de Keil Uvision®4 ingresando a la página web correspondiente, como se muestra en la figura 26.

Figura 26. **Página web de Keil versión 4**



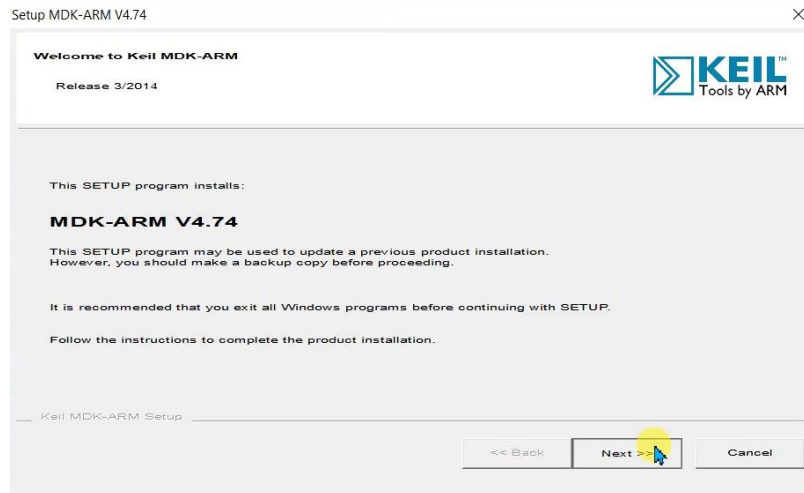
Fuente: elaboración propia, empleando Snipping Tool.

Por defecto la versión de Keil uVision®4 trae preinstalados todos los paquetes necesarios para poder trabajar con los distintos procesadores con arquitectura ARM Cortex®-M, por lo tanto, no es necesaria una instalación de paquetes extras.

Se inicia la instalación con la ayuda del *wizard*, como se muestra en la figura 27 se inicia al darle al botón llamado *next*. Se llenan los campos requeridos como se muestra en la figura 28, al finalizar de llenar los campos dar clic en *next* para que el *wizard* siga con la instalación, ver figura 29.

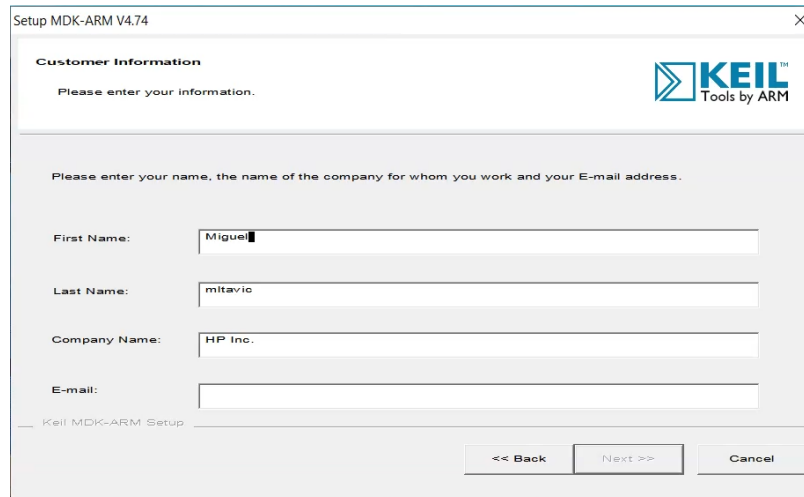
Luego aparecerán unas opciones como se muestra en la figura 30 y figura 31 dejar marcada las casillas tal cual como aparecen y dar click en *next*.

Figura 27. **Wizard de instalación**



Fuente: elaboración propia, empleando Snipping Tool.

Figura 28. **Formulario de datos de usuario**



Fuente: elaboración propia, empleando Snipping Tool.

Figura 29. **Formulario completo de usuario**

Setup MDK-ARM V4.74

Customer Information
Please enter your information.

KEIL™
Tools by ARM

Please enter your name, the name of the company for whom you work and your E-mail address.

First Name: Miguel

Last Name: Tavico

Company Name: HP Inc.

E-mail: Keil4

Keil MDK-ARM Setup

<< Back Next >> Cancel

Fuente: elaboración propia, empleando Snipping Tool.

Figura 30. **Agregar ejemplos básicos en Keil**

Setup MDK-ARM V4.74

File installation completed

KEIL™
Tools by ARM

µVision Setup has installed all files successfully.

Add example projects to the recently used project list.

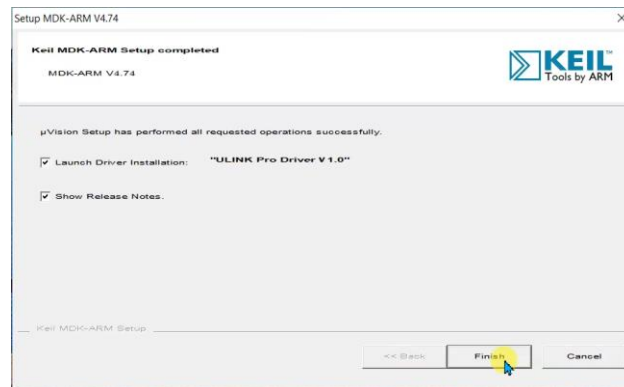
Preselect Example Projects for:
Simulated Hardware

Keil MDK-ARM Setup

<< Back Next >> Cancel

Fuente: elaboración propia, empleando Snipping Tool.

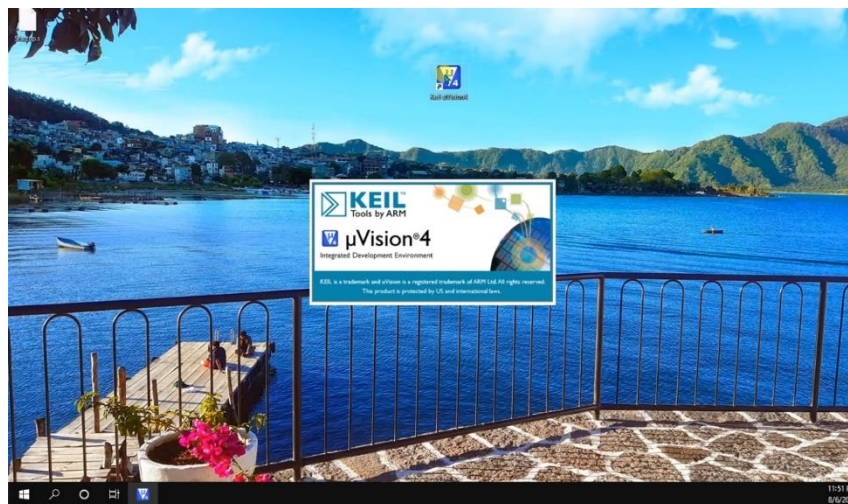
Figura 31. **Finalizar instalación de Keil uVision® 4**



Fuente: elaboración propia, empleando Snipping Tool.

Finalmente, aparecerá el ícono de Keil uVision®4 en el escritorio de Windows listo para ser ejecutado, como se muestra en la figura 32.

Figura 32. **Keil uVision® 4 instalado correctamente**

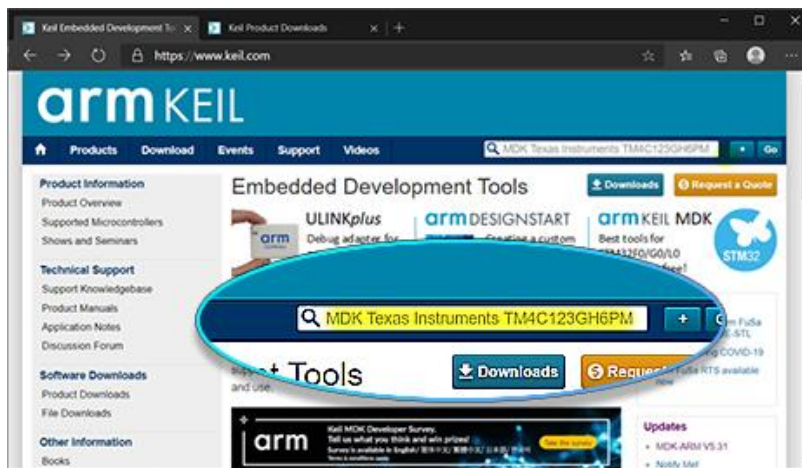


Fuente: elaboración propia, empleando Snipping Tool.

2.13.5. Instalación de Keil Uvision®5

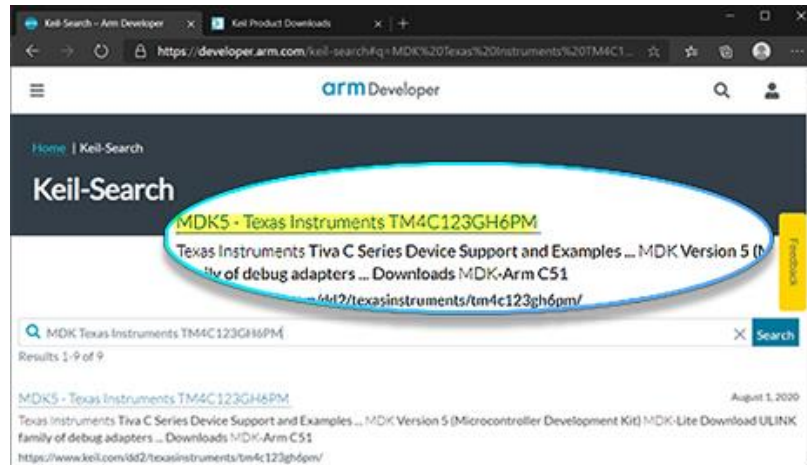
Proceda a buscar el software en la página web oficial de Keil. En el apartado de búsqueda colocar “MDK Texas Instruments TM4C123GH6PM”, ver figura 33. Como se muestra en la figura 34, seleccione la primera opción que aparece en la búsqueda. En la figura 35, en el apartado *Quick Links* seleccionar *MDK version 5*. Finalmente dar clic en *download MDK* como se muestra en la figura 36.

Figura 33. Búsqueda de Keil uVision® 5



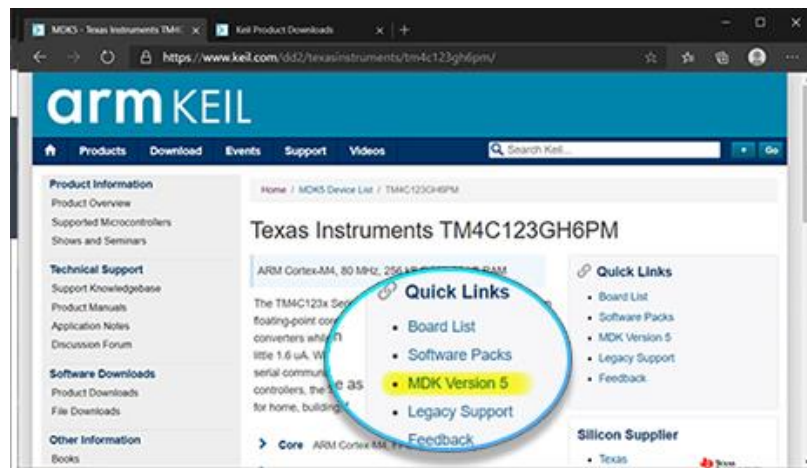
Fuente: elaboración propia, empleando Snipping Tool.

Figura 34. Selección de MDK5 -Texas Instruments para descargar



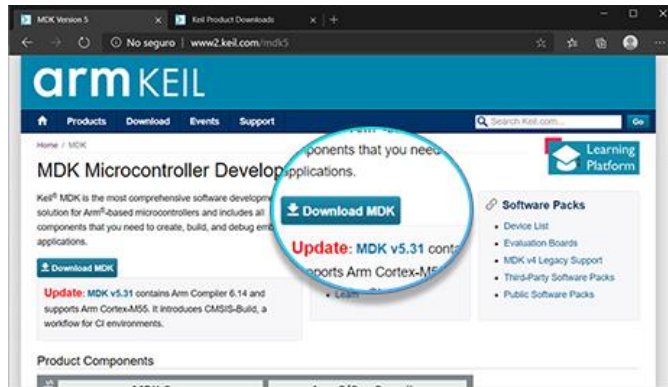
Fuente: elaboración propia, empleando Snipping Tool.

Figura 35. Selección de MDK versión 5 para descargar



Fuente: elaboración propia, empleando Snipping Tool.

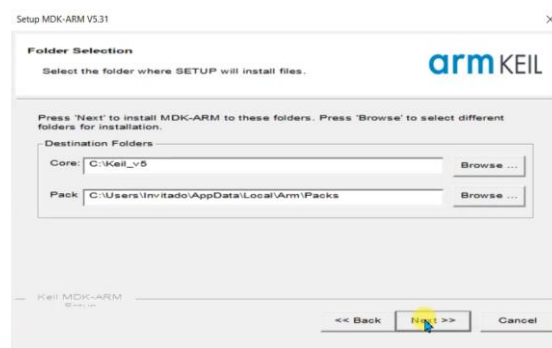
Figura 36. Descarga de MDK v5



Fuente: elaboración propia, empleando Snipping Tool.

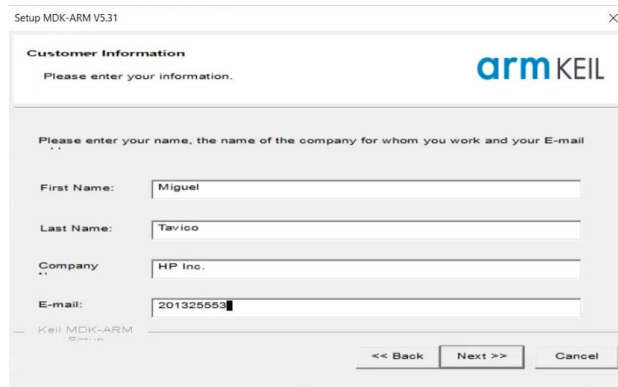
Finalmente, al descargar el instalador de Keil uVision®5 proceda a instalarlo. Se ejecuta el instalador y con ayuda del *wizard* se procede a instalar. En la figura 37 se muestra la dirección en donde se instalaron los archivos, se deja tal y como los muestra por defecto, para continuar con la instalación dar clic en el botón *next*. Llenar los campos que indica como se muestra en la figura 38 y dar click en el botón *next*.

Figura 37. Selección de ruta para instalar Keil uVision® 5



Fuente: elaboración propia, empleando Snipping Tool.

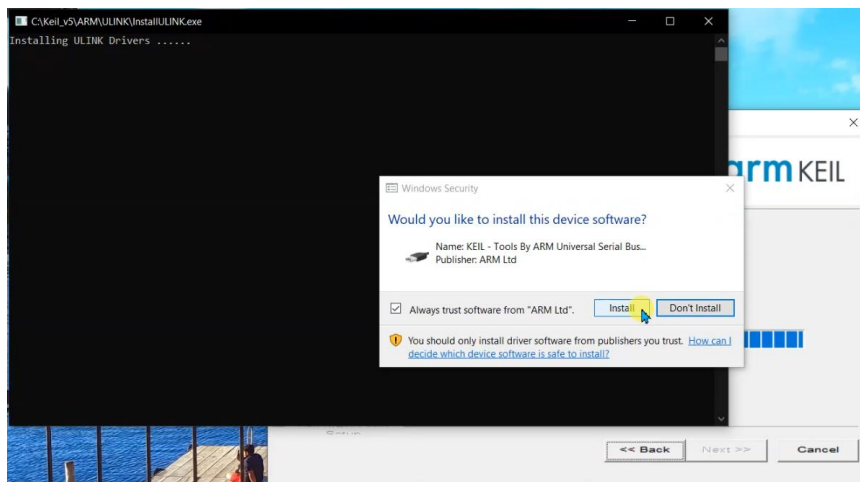
Figura 38. **Formulario de datos de usuario**



Fuente: elaboración propia, empleando Snipping Tool.

Finalmente, se abrirá una ventana en donde se autorizará la instalación de *Ulink Drivers*, dar clic en *Install* como se muestra en la figura 39.

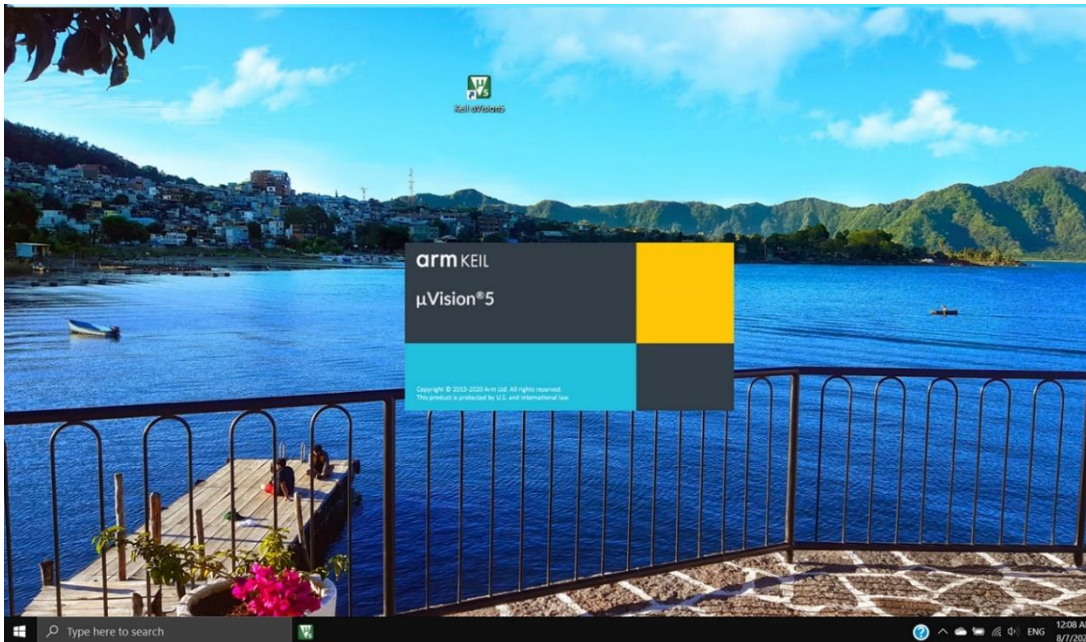
Figura 39. **Instalación de *Ulink Drivers***



Fuente: elaboración propia, empleando Snipping Tool.

Posteriormente aparecerá el ícono de Keil uVision®5 en el escritorio de Windows listo para ser ejecutado, como se muestra en la figura 40.

Figura 40. **Keil uVision® 5 instalado correctamente**



Fuente: elaboración propia, empleando Snipping Tool.

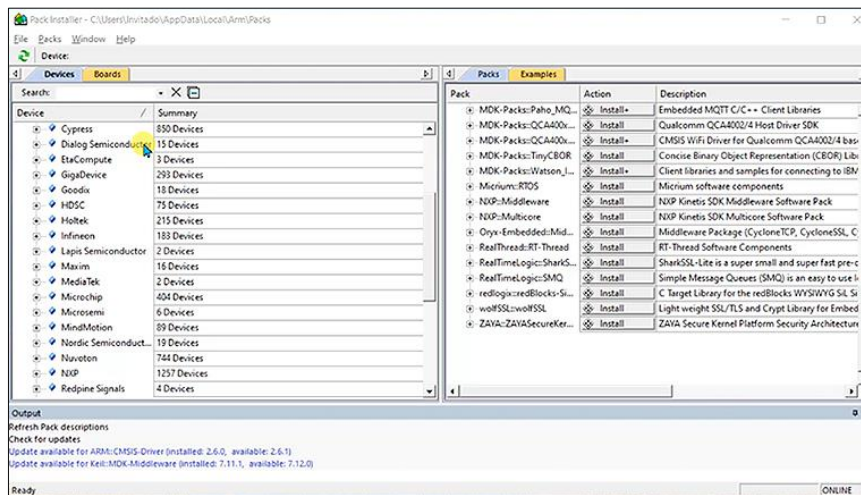
2.13.6. Instalación de los paquetes para el procesador Cortex®M TM4C123GH6PM

Keil uVision®5 comparado con uVision®4 no trae preinstalado todos los paquetes para programar los procesadores, se deben instalar de forma manual los paquetes que se requieren.

Al momento de finalizar la instalación del software keil uVision®5 y posteriormente ser ejecutado, automáticamente se abrirá una ventana que ayudará a instalar los paquetes necesarios para poder trabajar con procesadores

de la serie TM4C123G por parte de Texas Instruments, como se muestra en la figura 41.

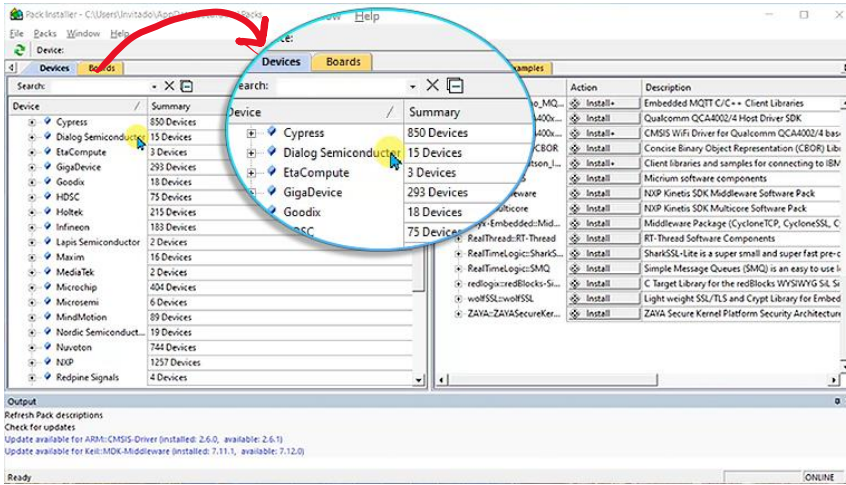
Figura 41. Instalación de paquetes en Keil uVision®5



Fuente: elaboración propia, empleando Snipping Tool.

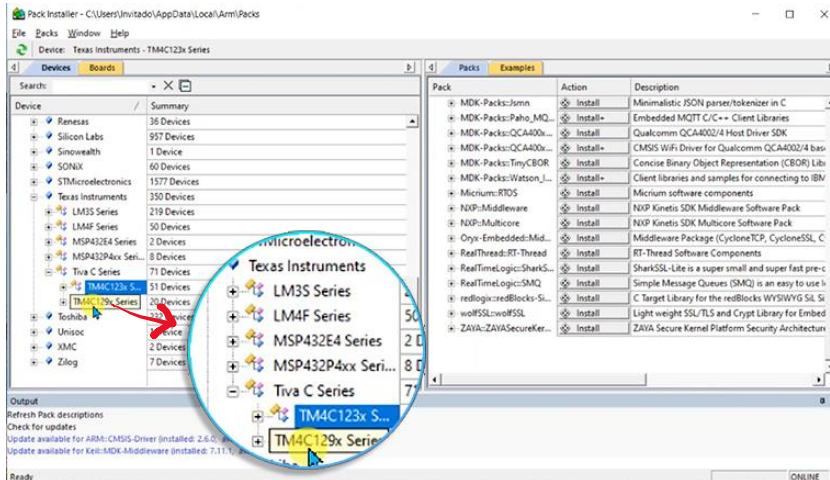
En la sección *Device*, ver figura 42, seleccionar la subcategoría Texas Instruments, dentro de esa subcategoría seleccionar Tiva C Series luego seleccionar TM4C123x Series, como se muestra en la figura 43.

Figura 42. Búsqueda de paquetes para *Tiva C Series*



Fuente: elaboración propia, empleando Snipping Tool.

Figura 43. Selección del paquete *TM4C123x Series*

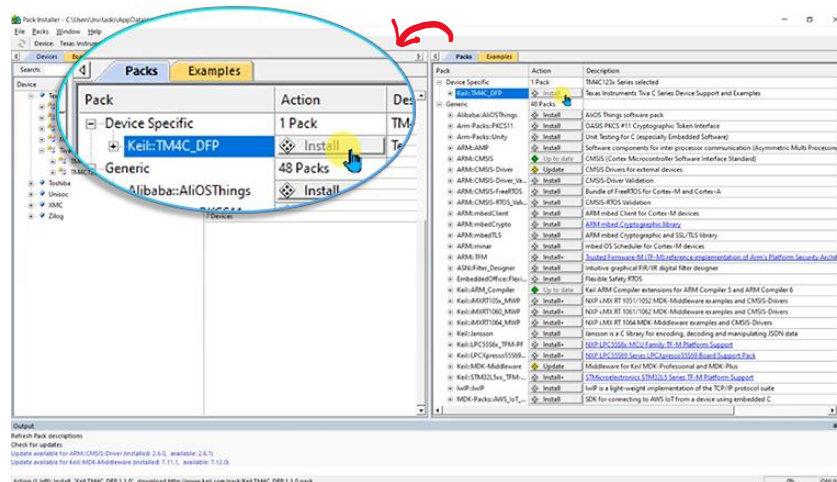


Fuente: elaboración propia, empleando Snipping Tool.

Al momento de seleccionar *TM4C123x Series* en la pestaña *Packs* seleccionar la subsección *Device Specific*. Seguidamente se localiza una

subsección llamada *Keil::TM4C_DFP* y a la par un botón con la etiqueta *Install*, dar click en *Install* como se muestra en la figura 44.

Figura 44. Instalación de Keil::TM4C_DFP



Fuente: elaboración propia, empleando Snipping Tool.

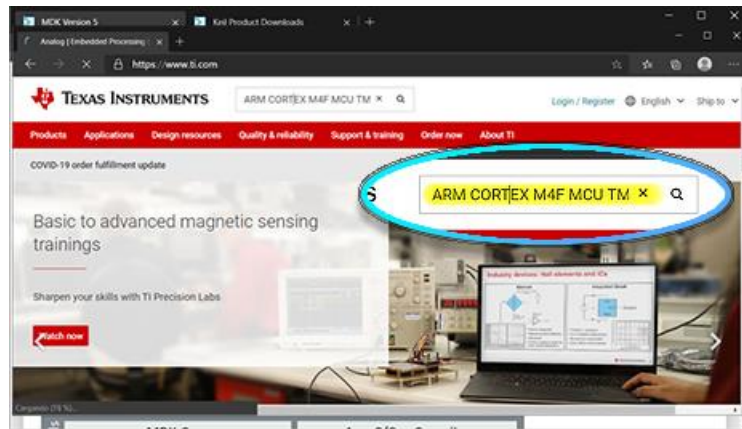
Luego de haber instalado el paquete de Texas Instruments ya tenemos listo el entorno de desarrollo para poder iniciar a programar el procesador Cortex®-M4 TM4C123GH6PM.

2.13.7. Instalación de drivers

Para poder trabajar con el procesador Cortex®-M dentro del microcontrolador TM4C123GH6PM es necesario la instalación de los drivers de Tiva-C Launchpad el cual es el dispositivo electrónico en donde se encuentra embebido el procesador a programar. Ir a la página oficial de Texas Instruments: <https://www.ti.com/>. Se procede a buscar *ARM CORTEX M4F MCU TM4123G*

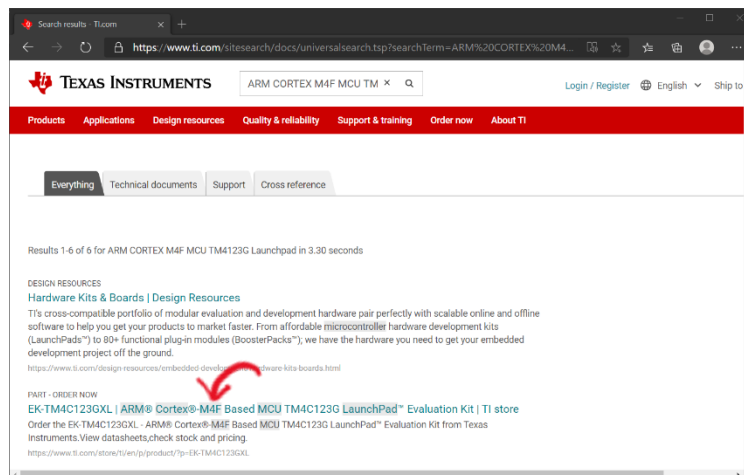
Launchpad como se muestra en la figura 45 seguidamente seleccione la opción llamada EK-TM4C123GXL como muestra en la búsqueda de la figura 46.

Figura 45. **Página oficial de Texas Instruments**



Fuente: elaboración propia, empleando Snipping Tool.

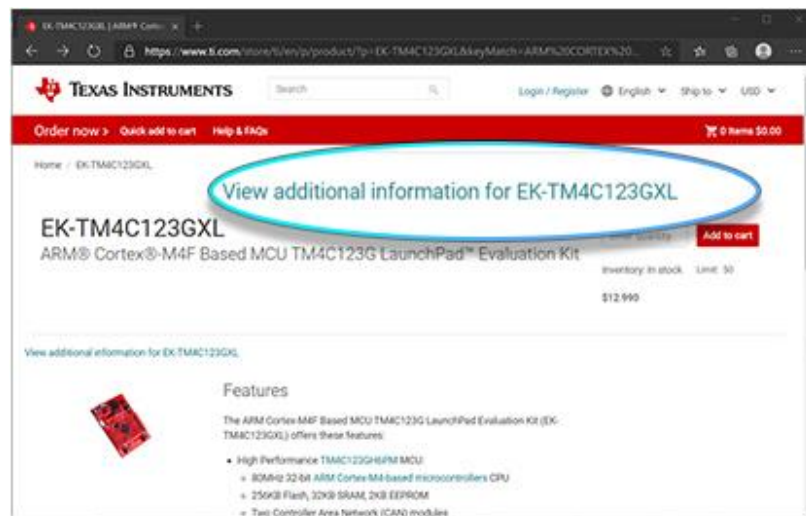
Figura 46. **Búsqueda de ARM CORTEX**



Fuente: elaboración propia, empleando Snipping Tool.

Como muestra la figura 47, dar clic en *View additional information for EK-TM4C123GXL* lo cual llevará a la página web en donde se puede descargar los softwares y paquetes necesarios para poder usar la tarjeta de desarrollo en los distintos sistemas operativos.

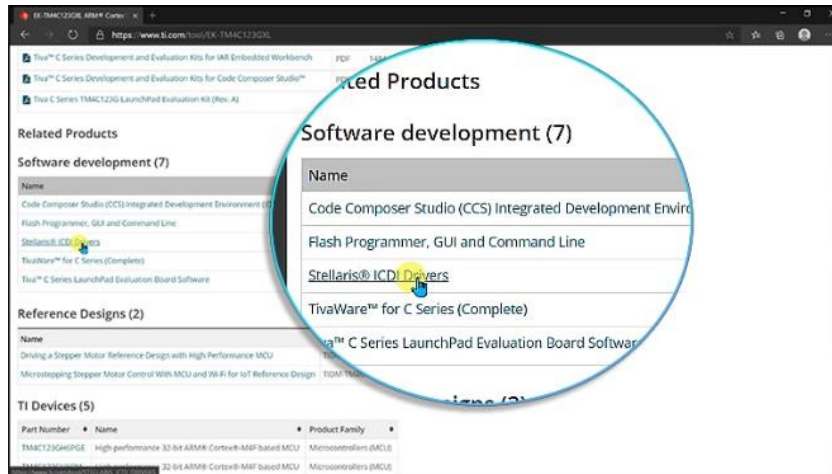
Figura 47. **Búsqueda del software para ARM CORTEX**



Fuente: elaboración propia, empleando Snipping Tool.

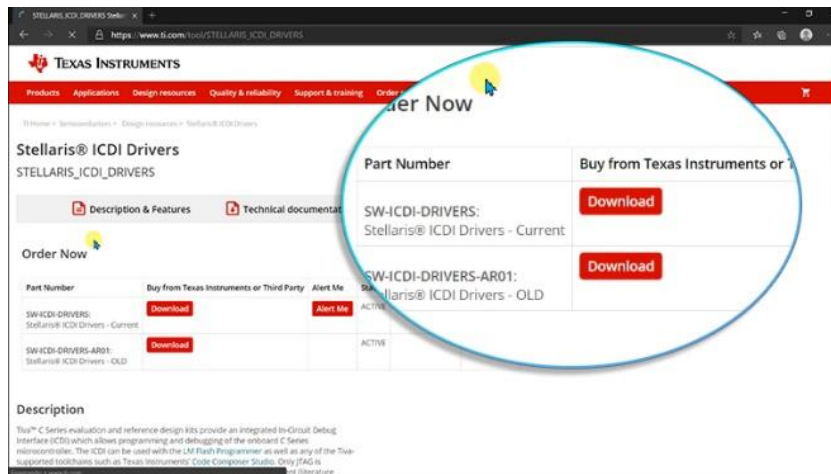
En la siguiente página web, buscar el apartado de *Software Development* en la subsección dar clic en el link con etiqueta llamada *Stellaris® ICD1 Drivers* como se muestra en la figura 48. Luego dar click en el botón *Download* como se muestra en la figura 49 y esperar hasta que se descargue el archivo.

Figura 48. **Búsqueda de Stellaris® ICDI Drivers**



Fuente: elaboración propia, empleando Snipping Tool.

Figura 49. **Descarga de Stellaris® ICDI Drivers**

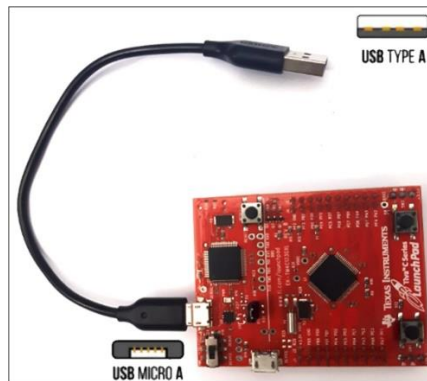


Fuente: elaboración propia, empleando Snipping Tool.

Proceda a conectar la Tiva-C Launchpad a la computadora con la ayuda de un cable *micro-usb* tipo A como se muestra en la figura 50, se usa el puerto en el

área de *debug* como se muestra en la figura 51. Finalmente, conecte la tarjeta mediante el cable usb a la computadora, como se muestra en la figura 52.

Figura 50. **Conexión del cable USB A**



Fuente: elaboración propia, empleando Adobe Photoshop.

Figura 51. **Conexión del cable USB en la Tiva C**



Fuente: elaboración propia, empleando Adobe Photoshop.

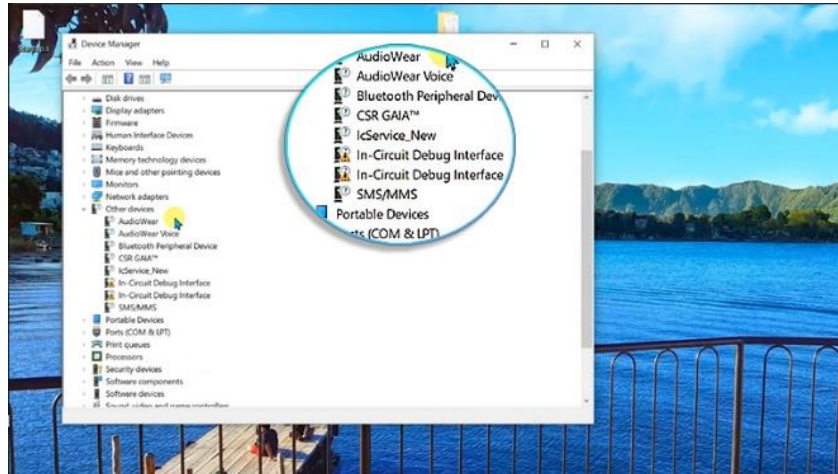
Figura 52. **Conexión Tiva C y computadora**



Fuente: elaboración propia, empleando Adobe Photoshop.

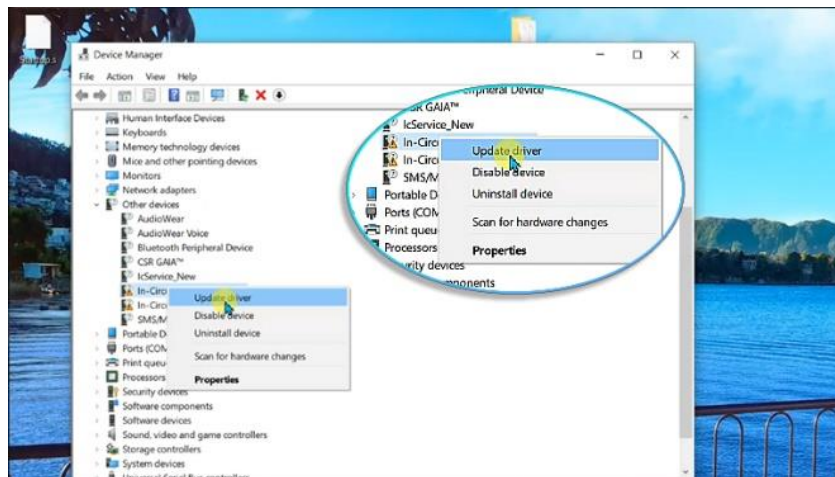
Luego ir a las opciones de *Device Manager* de Windows y buscar el dispositivo que Windows no ha reconocido como se muestra en la figura 53, posicionarse sobre el icono con signo de exclamación y dar clic derecho seguidamente clic en *Update Driver*, como se muestra en la figura 54, seguidamente seleccione la opción *Browse my computer for driver software*. En la ventana que aparecerá, seleccione la ruta de la carpeta en la cual se descargó Stellaris® ICDI Drivers como se muestra en la figura 55.

Figura 53. **Device Manager de Windows**



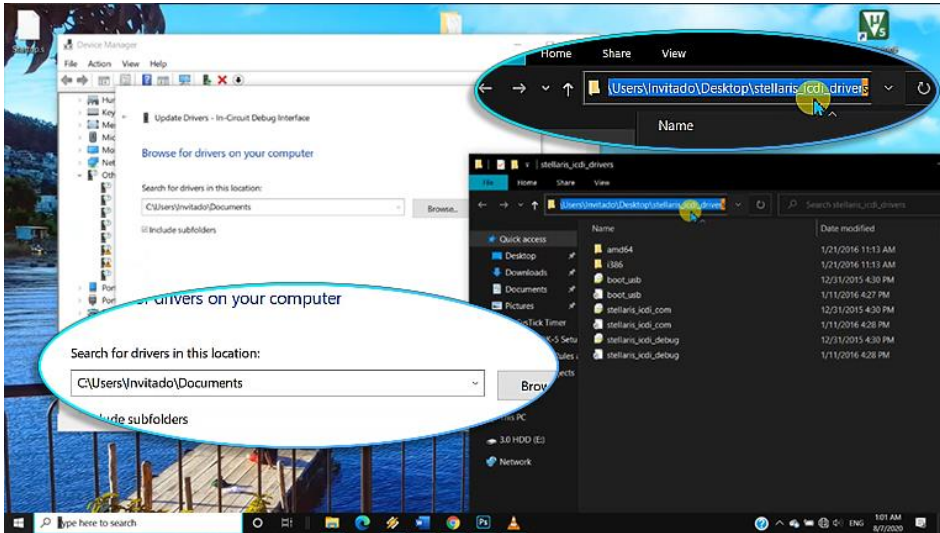
Fuente: elaboración propia, empleando Snipping Tool.

Figura 54. **Update Drivers en Device Manager**



Fuente: elaboración propia, empleando Snipping Tool.

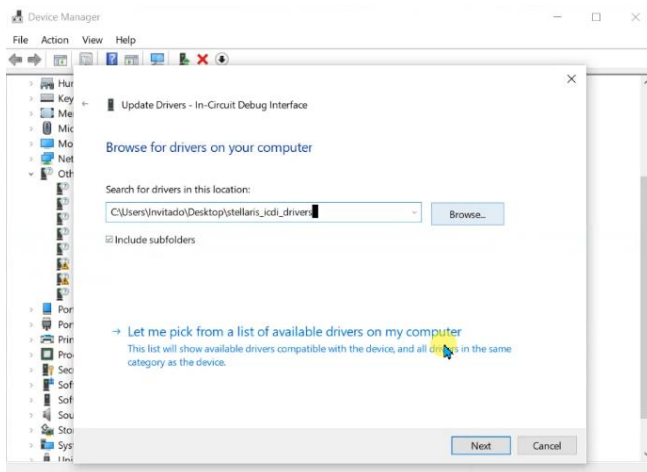
Figura 55. **Ubicación de los drivers dentro de Windows**



Fuente: elaboración propia, empleando Snipping Tool.

Finalmente dar clic en *Next* y esperar a que los drivers se terminen de instalar, como se muestra en la figura 56.

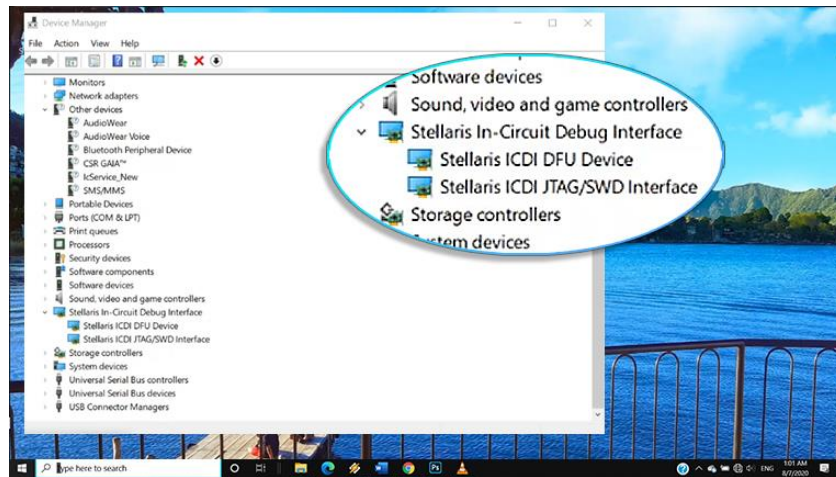
Figura 56. **Selección de la ubicación de los drivers**



Fuente: elaboración propia, empleando Snipping Tool.

Al tener una instalación correcta, en la ventana llamada *Device Manager* aparecerán los íconos distintos juntos como se muestra en la figura 57.

Figura 57. **Actualización exitosa de drivers para *Stellaris® ICDI***

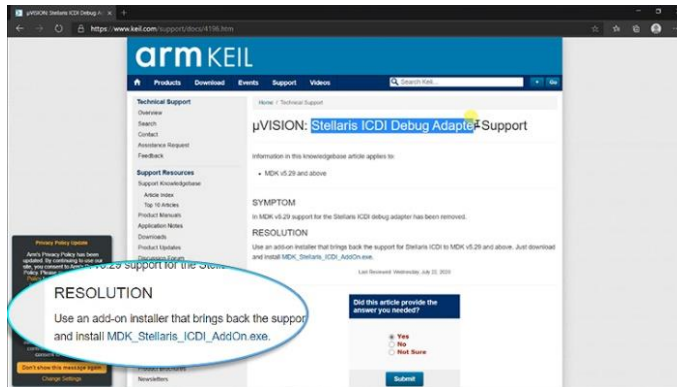


Fuente: elaboración propia, empleando Snipping Tool.

2.13.8. Instalación de Debug Stellaris® ICDI

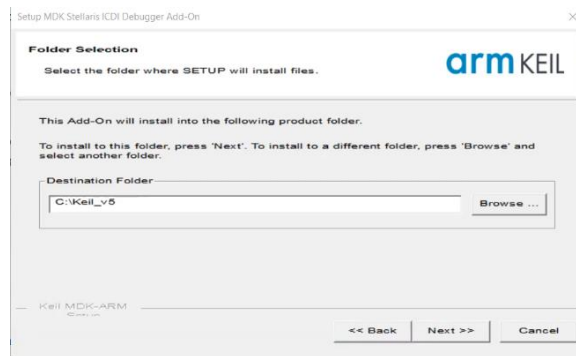
En la página web oficial de Keil buscamos Stellaris® ICDI *Debug Adapter Support* como se muestra en la figura 58. Proceda a instalarlo en la ruta de la carpeta en donde se ha instalado Keil, como se muestra en la figura 59.

Figura 58. **Búsqueda de Stellaris® ICDI Debus Adapter Support**



Fuente: elaboración propia, empleando Snipping Tool.

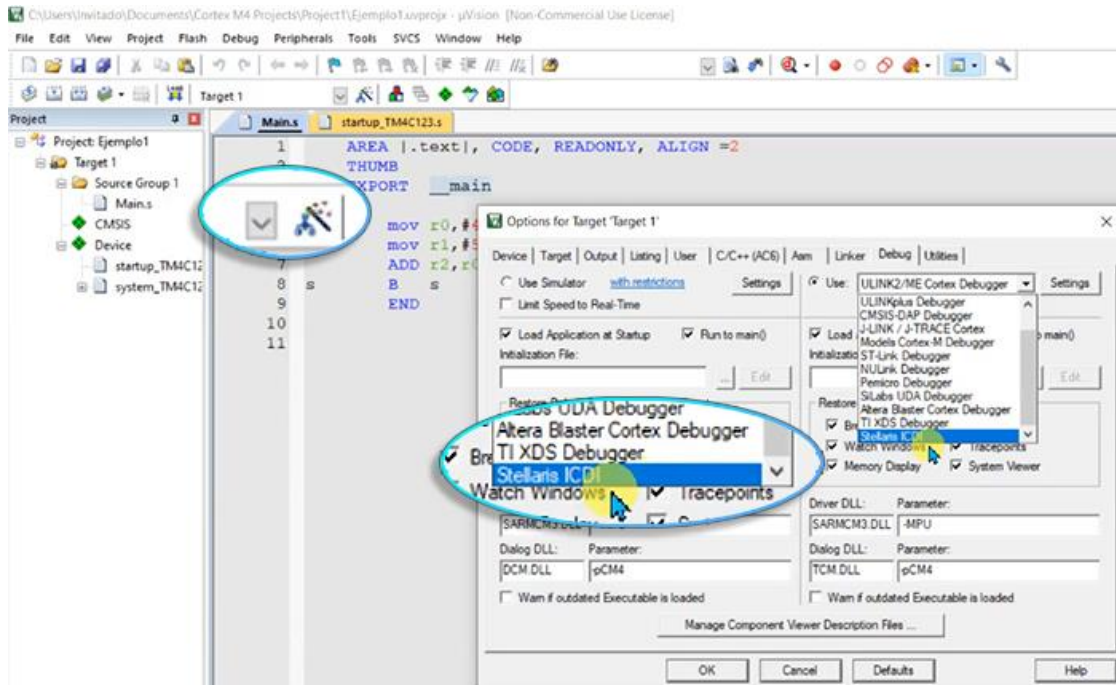
Figura 59. **Instalación de Stellaris® ICDI Debus Adapter Support**



Fuente: elaboración propia, empleando Snipping Tool.

Al finalizar la instalación, proceda a reiniciar el software Keil, en el menú de *Options for Target* y en la pestaña *Debug* se visualiza Stellaris® ICDI, recién instalado, como se muestra en la imagen 60.

Figura 60. Verificación de la correcta instalación de Stellaris® ICDI



Fuente: elaboración propia, empleando Snipping Tool.

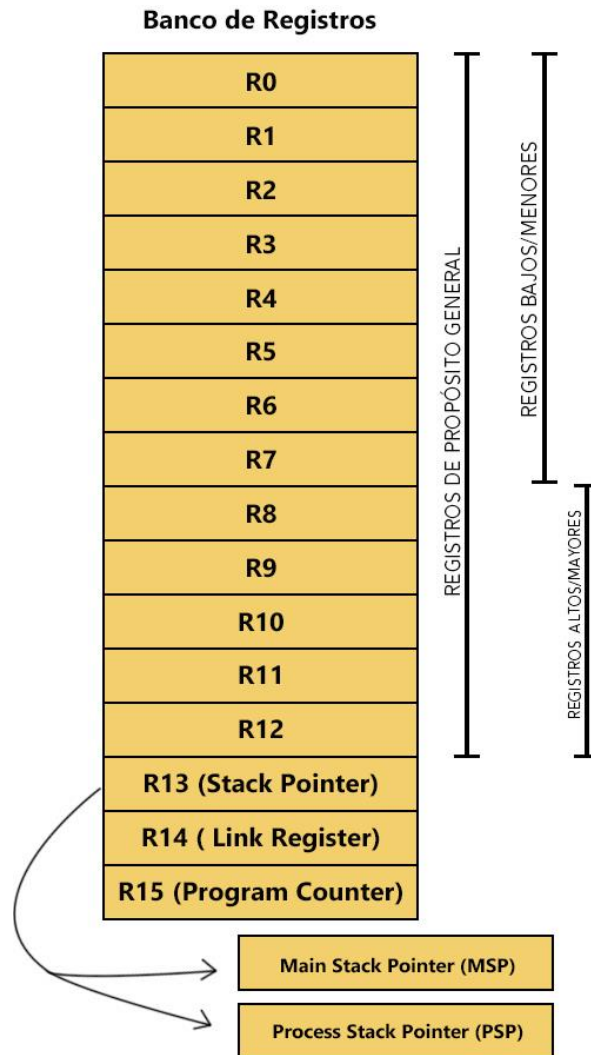
3. PROGRAMACIÓN EN LOS PROCESADORES CORTEX-M UTILIZANDO ENSAMBLADOR

3.1. Registros

Los registros de los procesadores Cortex®-M son utilizados para realizar el procesamiento y control de datos. Los registros se agrupan en una unidad llamada banco de registro (en inglés *Register Bank*). Cada instrucción especifica la operación a realizar y los registros a ser utilizados. En la arquitectura ARM, si los datos almacenados en memoria se utilizaran, primeramente, deben ser leídos de la memoria y ser cargados hacia los registros dentro del banco de registros. Luego se procesan los datos dentro del procesador y finalmente se deben volver a escribir en memoria.

El procesador Cortex®-M4 posee un banco de registro central y un banco de registro para la unidad de punto flotante. El banco de registro central está compuesto por 16 registros de los cuales 13 son registros de propósito general de 32 bits y los otros 3 registros son registros especiales igualmente de 32 bits. En la figura 61 se presenta el banco de registros centrales.

Figura 61. **Banco de registros del núcleo**



Fuente: elaboración propia, empleando Adobe Photoshop.

3.1.1. **Stack Memory**

Stack Memory o Pila de memoria es un área contigua de memoria en bloque de tipo LIFO (del inglés *Last-In-First-Out*), que se utiliza para:

- Almacenar variables locales
- Almacenar argumentos adicionales que serán utilizados en subrutinas cuando no hay suficientes registros de argumentos disponibles
- Trabajar con funciones o rutinas anidadas
- Manejar interrupciones del procesador

Los procesadores ARM usan la memoria de Pila para almacenar datos mediante las instrucciones PUSH y POP. PUSH se utiliza para almacenar datos en la Pila y POP para recuperar los datos almacenados. Los procesadores Cortex®-M usan el modelo de memoria llamada *full-descending stack*, esto debido a que cada vez que se guarda un nuevo dato (con la instrucción PUSH) en la Pila el Stack Pointer desciende en posición y luego escribe el dato en la Pila. Por lo anterior el Stack Pointer siempre apunta al último elemento apilado en la memoria.

3.1.2. R0-R12

Los registros en el rango de R0 a R12 son llamados registros de propósito general, de los cuales los primeros 8 registros (R0 a R7) son llamados registros bajos o menores. Por la limitación de espacio disponible muchas instrucciones de 16 bits solo pueden acceder a los registros bajos. Los registros altos o mayores (R8 a R12) pueden ser utilizados con instrucciones de 32 bits y también por algunas instrucciones de 16 bits.

3.1.3. R13, Stack Pointer (SP)

SP es un registro que apunta hacia los datos en la parte superior de la Pila. El Cortex®-M3 y Cortex®-M4 contienen dos punteros de Pila, los cuales son:

- Main Stack Pointer (MSP): se usa cuando se manejan interrupciones y se usa opcionalmente durante la ejecución regular del programa.
- Process Stack Pointer (PSP): utilizado por el código de la aplicación del usuario.

SP es usado para acceder a la Pila de memoria mediante las operaciones PUSH y POP. En los procesadores ARM Cortex®-M, las operaciones PUSH y POP son siempre de 32 bits, y las direcciones de las transferencias en las operaciones de Pila deben estar alineadas con los límites de una palabra (*word* en inglés) de 32 bits. La instrucción PUSH es utilizada para almacenar datos en la Pila y POP para recuperar datos de esa misma Pila. Los dos bits más bajos de los punteros de la pila (SP) siempre son cero, lo que significa que siempre están alineados a una palabra (Word).

3.1.4. R14, Link Register (LR)

LR se utiliza para almacenar la dirección de retorno al momento de llamar o invocar (como puede ocurrir con una instrucción *branch*) una función o subrutina. Al finalizar las operaciones de la función o subrutina que ha sido llamada, el control del programa (PC) puede regresar a la ubicación de donde se realizó la llamada y carga el valor de LR en el contador de programa (PC).

Si una función Función_1 requiere llamar a una segunda función Función_2 y posteriormente se requiere regresar a Función_1 se necesita almacenar el valor de LR justo antes de realizar la llamada dentro de esta función, el valor de LR se almacena en la Pila.

3.1.5. R15, Program Counter (PC)

Este registro contiene la dirección del programa actual, el registro puede ser escrito y leído. Al momento de leer el registro este devuelve la dirección de la instrucción actual más 4 (se tiene un *offset* debido a la naturaleza del Pipeline del procesador ARMv7). Escribir en el registro PC provoca un salto (en inglés *branch*). Las instrucciones deben estar alineadas a un límite de *half-word* o *word*, el bit menos significativo (LSB del inglés *least significant byte*) de PC es cero. Al momento de utilizar instrucciones de lectura de memoria o Branch se debe establecer el LSB del registro PC en 1 para indicar que se está trabajando en estado Thumb.

3.1.6. Registros especiales

Estos registros contienen el estado del procesador, definen los estados de operación y la máscara de las interrupciones o excepciones. Los registros especiales no están mapeados en memoria y se puede acceder a ellos mediante instrucciones especiales de acceso a registros como MSR y MRS.

- MRS <registro>, <registro_especial>; Leer un registro especial en un registro de propósito general.
- MSR <registro_especial>, <registro>; Escritura dentro de un registro especial

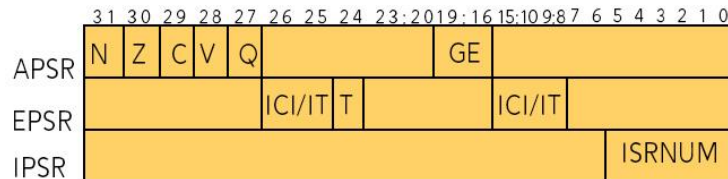
3.1.7. Registro de estado del programa

El registro de estado del programa o Program Status Registers (PSR) en inglés, está compuesto de 3 estados de registros diferentes:

- *Application PSR (APSR)*
- *Execution PSR (EPSR)*
- *Interrupt PSR (IPSR)*

xPSR es una combinación de los 3 registros diferentes: *APSR*, *IPSR* y *EPSR* (del inglés Application Programm Status Register, Interrupt Program Status y Execution Program Status Register, respectivamente). En la figura 62 se muestra la configuración de los registros APSR, EPSR e IPSR.

Figura 62. **Registros APSR, EPSR e IPSR**



Fuente: elaboración propia, empleando Adobe Photoshop.

En la figura 63 y 64 se muestra la combinación de los registros APSR, EPSR e IPSR. Al combinar los tres registros se obtiene el registro único xPSR.

Figura 63. **Bits dentro del registro xPSR**

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
xPSR	N	Z	C	V	Q	ICI/IT	T		GE	ICI/IT		NÚMERO DE EXCEPCIÓN				

Fuente: elaboración propia, empleando Adobe Photoshop.

Figura 64. **Registro xPSR**



Fuente: elaboración propia, empleando Adobe Photoshop.

Se puede acceder a los tres registros como un solo registro combinado usando PSR como se muestra en el ejemplo de la figura 65.

Figura 65. **Acceso al registro PSR**

```
93 MRS r0, PSR ; Lee la palabra de estado del programa (xPSR)
94 MSR PSR, r0 ; Escribe la palabra de estado de programa (xPSR)
```

Fuente: elaboración propia, empleando Notepad++.

Se puede acceder al registro PSR de forma individual, como se muestra en el ejemplo de la figura 66.

Figura 66. **Acceso individual al registro PSR**

```
96 MRS r0, APSR ; Leer el estado de la bandera en R0
97 MRS r0, IPSR ; Lee el estado de excepción/interrupción
98 MSR APSR, r0 ; Escribe el estado de la bandera
```

Fuente: elaboración propia, empleando Notepad++.

3.1.8. Principios básicos en ensamblador

Los procesadores de la familia Cortex-M están basados en la tecnología Thumb-2, por lo que pueden trabajar con instrucciones de longitud de 16 y 32 bits. La tecnología Thumb-2 combina las instrucciones de 16 y 32 bits en un único estado de operación.

3.1.8.1. Sintaxis usada en lenguaje ensamblado

Las instrucciones escritas en lenguaje ensamblador tienen 4 campos los cuales se encuentran separados por espacio o un tabulador (tecla *Tab*). El orden correcto para escribir una línea de código se presenta en la figura 67.

Figura 67. Campos para utilizar en *assembler*

Etiqueta ⇄ **Opcode** ⇄ **Operandos** ⇄ **;Comentario**

Fuente: elaboración propia, empleando Adobe Photoshop.

Label o etiqueta, este campo es opcional y es usado para encontrar la posición de la instrucción actual dentro de la memoria ya que se usa como referencia. La dirección de la instrucción puede ser obtenida buscando la etiqueta.

Después de *Label* se encuentra un *opcode* o nemónico el cual indica el nombre de la instrucción a ejecutar, seguidamente se encuentran los operandos. La cantidad de operandos dependen del tipo de *opcode*, puede ser dos o tres operandos. Finalmente se tiene el comentario al cual le precede un punto y coma (“;”) en donde por lo general se coloca texto el cual explica la línea de código, el

comentario es opcional. En la figura 68 se presentan los campos y su correcto uso.

Figura 68. **Sintaxis en lenguaje ensamblador**



Fuente: elaboración propia, empleando Adobe Photoshop.

Aspectos para tomar en consideración:

- Para instrucciones de procesamiento, el primer operando presente en línea de código representa el registro de destino.
- Para instrucciones de solo lectura, el primer operando presente en la línea de código representa a el registro destino, en el cual se almacenarán los datos.
- Para una instrucción de escritura de memoria, el primer operando presente en la línea de código será el registro que contenga los datos que se escribirán en memoria.

3.1.8.2. **Constantes**

Como en todos los lenguajes de programación es muy útil el uso de valores constantes. Una constante es un valor que no se puede cambiar cuando se ejecuta el programa, una constante puede ser usada en todo el código del programa y es definido al inicio por el mismo programador o usuario. Al momento de usar constantes en el código, éste se vuelve más sencillo de entender y de ser editado. En la figura 69 se presenta la sintaxis correcta para definir un valor

constante, al momento de definir un valor constante se usa el *opcode* llamado *EQU*.

Figura 69. **Sintaxis de código**

Nombre_de_la_constante opcode valor ;comentario

Fuente: elaboración propia, empleando Adobe Photoshop.

En la figura 70 se presenta un ejemplo de la correcta forma de asignar de un valor constante en lenguaje ensamblador.

Figura 70. **Implementación de sintaxis en *assembler***

Edad EQU 26 ;asigna el valor de 26 al simbolo Edad

Fuente: elaboración propia, empleando Adobe Photoshop.

3.1.8.3. Insertar valores en el programa

Cuando se usa la instrucción LDR para cargar un valor dentro de un registro el valor requiere de un prefijo “=”. Cuando se requiere cargar un valor inmediato dentro de un registro (como con la instrucción MOV) se debe usar el prefijo “#”. Generalmente la instrucción LDR es usada para cargar un valor de la memoria a un registro y la instrucción STR es usada para almacenar el valor dentro de un registro a la memoria. Se presenta un ejemplo del uso correcto de LDR y MOV en la figura 71.

Figura 71. Instrucción LDR Y MOV

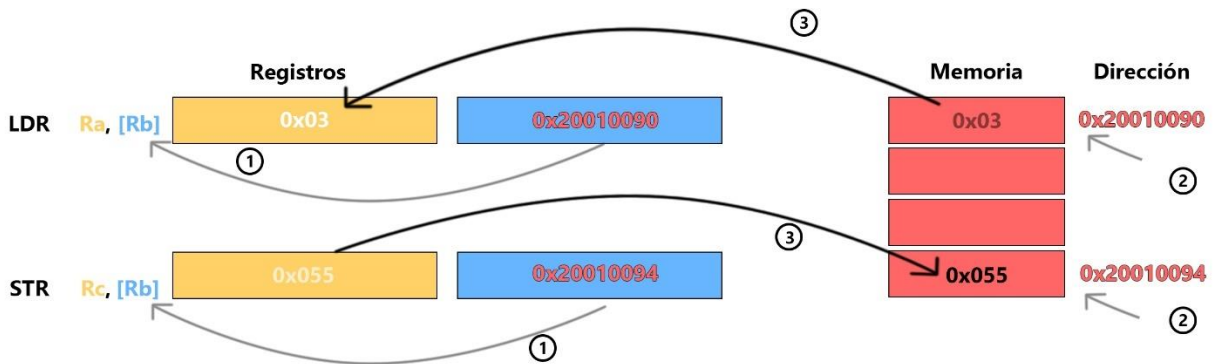
```

26 Constante_1 EQU 0xE000FF00
27 Constante_2 EQU 0X200
28 LDR R0,=Constante_1; Coloca el valor 0xE000FF00 dentro de R0
29 MOVS R1,#Constante_2; Coloca el valor inmediato de 0x200 dentro del registro R1
30 STR R1,[R0] ; Almacena 0x200 en la dirección de memoria igual a 0xE000FF00
    
```

Fuente: elaboración propia, empleando Notepad++.

En la figura 72 se presenta una comparación entre la instrucción LDR y STR.

Figura 72. Uso de las instrucciones LDR y STR



Fuente: elaboración propia, empleando Adobe Photoshop.

Con la instrucción LDR ocurre lo siguiente:

- Se ubica el registro Rb y se lee su contenido, el contenido es una dirección ubicada en memoria. Cuando se especifica una dirección en memoria se utiliza corchetes “[]”.
- Se ubica la dirección de memoria almacenada en Rb, la dirección es 0x20010090.
- Se lee el valor almacenado en memoria con dirección igual a 0x20010090, seguidamente se carga el valor 0x03 en el registro Ra.

Con la instrucción STR ocurre lo siguiente:

- Se ubica el registro Rb y se lee su contenido, el contenido es una dirección ubicada en memoria.
- Se ubica la dirección de memoria almacenada en Rb, la dirección es 0x20010094.
- Se lee el valor almacenado en Rc y seguidamente se almacena en memoria ubicada en la dirección 0x20010094.

3.1.8.4. Datos en programa

Otra característica típica de las herramientas del lenguaje ensamblador es permitir la inserción de datos dentro del programa. Se permite definir datos en una determinada ubicación en memoria del programa y posteriormente acceder a ellos con instrucciones de lectura de memoria. En la figura 73 se presenta un ejemplo en donde se puede ingresar datos dentro del programa.

Figura 73. Datos en programa

```
33 Align 4
34 My_Number DCD 0x12345678
35 HOLA_MUNDO DCB "Hello\n",0 ; Cadena terminada con Null
36 ...
37 LDR R3,=My_Number; R3 almacena el valor 0x12345678 el cual es una dirección de memoria
38 LDR R4, [R3]; En R4 se almacenara un valor, dicho valor se encuentra situado en la dirección de memoria guardada en R3
39 LDR R0, HOLA_MUNDO; obtiene la dirección inicial de HOLA_MUNDO
40 BL PrintText; llama la función con el nombre PrintText
41
```

Fuente: elaboración propia, empleando Notepad++.

En el ejemplo de la figura 73 se utilizó la instrucción DCD para insertar datos del tamaño de una palabra (en inglés *word*) y DCB se utilizó para insertar datos con un tamaño de un byte. Al insertar datos del tamaño de una palabra en el programa, debemos usar la directiva “ALIGN” para alinear los datos y acceder a

ellos con menos ciclos de reloj. En el ejemplo anterior se debe usar *ALIGN 4*, el número 4 determina el tamaño de la alineación. Al utilizar *ALIGN 4* se fuerza a los datos a ser alineados con un límite de una palabra (en inglés *word*), se utilizará 4 bytes para almacenar cada uno de los datos.

3.1.8.5. Directivas

En la tabla VII se presenta un listado de directivas de uso común para insertar datos en un programa.

Tabla VII. **Directivas para insertar datos en programa**

Tipo de dato a insertar	Keil MDK-ARM, Opcode	Ejemplo Hexadecimal
<i>Byte</i>	DCB	DCB 0xFF
<i>Half-word</i>	DCW	DCW 0xFFFF
<i>Word</i>	DCD	DCD 0xFFFFFFFF
<i>Double-word</i>	DCQ	DCQ 0xFFFFFFFFFFFFFFFF
<i>Floating point (Precisión simple)</i>	DCFS	DCFS 1E2
<i>Floating Point (Precisión Doble)</i>	DCFD	DCFD 2.71828
<i>String</i>	DCB	DCB "Miguel\n" 0,
<i>Instruction</i>	DCI	DCI 0BE00; Breakpoint (BKPT 0)

Fuente: elaboración propia, empleando Microsoft Word.

Otro listado de directivas se presenta en la tabla VIII, las cuales son muy útiles.

Tabla VIII. Directivas importantes

Directivas	Lenguaje ensamblador
THUMB	Especifica el código en lenguaje ensamblador como instrucciones <i>Thumb</i> en el formato UAL.
CODE16	Especifica el código en lenguaje ensamblador como instrucciones <i>Thumb</i> Pre-UAL.
AREA <nombre_de_sección> {,<atributo>} {,<atributo>} ...	Instruye al ensamblador para ensamblar un nuevo código o sección de datos. Las secciones son fragmentos de código o datos independientes, con nombre e indivisibles que son manipulados por el "Linker".
SPACE <cantidad de Bytes>	Reserva un bloque de memoria y lo llena con ceros.
FILL <cantidad de Bytes>{,<valor>{,<tamaño_del_valor>}	Reserva un bloque de memoria y lo llena con el valor especificado. El tamaño del valor puede ser <i>byte</i> , <i>word</i> o <i>half-word</i> , especificado por <i>tamaño_del_valor</i> (1/2/4).

Continuación de la tabla VIII.

ALIGN {<expresión>{,<offset>{.<pad>{,<padsiz>}}}}	Alinea la ubicación actual a un límite especificado rellenando con ceros o instrucciones <i>NOP</i> . Por ejemplo, <i>ALIGN 4</i> , se asegura que la siguiente instrucción o los siguientes datos tengan una alineación con límite de una palabra (<i>word</i>).
EXPORT <símbolo>	Declara un símbolo que puede ser usado por el <i>Linker</i> para resolver referencias simbólicas en objetos o librerías de archivos separadas.
IMPORT	Declarar una referencia de símbolo en archivos de biblioteca u objetos separados que debe resuelto por el <i>Linker</i> .
CODE	Específica que se tendrá solo código ingresado por el usuario
DATA	Específica que se tendrá solo datos ingresados por el usuario

Continuación de la tabla VIII.

LTORG	Indica al ensamblador que ensamble el grupo literal actual de inmediato. El grupo literal contiene datos tales como valores constantes para la pseudo instrucción LDR.
END	Se coloca para indicar el final del archivo
EQU	Asigna un valor numérico constante a un símbolo.

Fuente: elaboración propia, empleando Microsoft Word.

En los procesadores Cortex®-M cuando a una instrucción se le agrega un sufijo se obtiene que esa instrucción tenga una función extra. Una instrucción como puede ser *ADD* al agregarle el sufijo *S* la instrucción pasaría a ser *ADDS* la cual aparte de sumar dos operandos actualiza las banderas del registro APSR, eso al estar usando la sintaxis *UAL*. Los sufijos para las instrucciones se muestran en la tabla IX.

Tabla IX. **Sufijos para lenguaje ensamblador para Cortex®-M**

Sufijo	Descripción
S	Actualiza APSR (del inglés <i>Application Program Status Register</i> como por ejemplo <i>Carry</i> , <i>Overflow</i> , <i>Zero</i> y <i>Banderas Negativas</i>).
EQ, NE, CS, CC, MI, OL, VS, VC, HI, LS, GE, LT, GT y LE	Ejecuciones condicionales, EQ = igual, NE = no es igual, LT= Menor que, GT = mayor que, entre otros..
.N, .W	Especifica el uso de instrucciones de 16 bits (<i>narrow</i>) o 32 bits (<i>wide</i>).
.32, .F32	Especifica el uso de datos de precisión simple con un tamaño de 32 bits.
.64, F64	Especifica el uso de datos de precisión doble con un tamaño de 64 bits.

Fuente: elaboración propia, empleando Microsoft Word.

3.1.9. Lenguaje ensamblador unificado o en inglés UAL

Cuando se desarrolló la tecnología Thumb-2, casi todas las instrucciones Thumb estaban disponibles dentro de esa nueva tecnología. Cabe resaltar que la sintaxis para código en Thumb-2 y Thumb es distinta.

Una de las diferencias entre estas dos versiones es que Thumb actualiza automáticamente el registro APSR y en Thumb-2 la actualización de APSR es opcional. Por lo anterior, se estableció una sintaxis única en el lenguaje ensamblador llamada UAL. Al utilizar UAL se evita complicaciones y confusiones al momento de programar.

La versión de sintaxis pre-UAL (del inglés *Unified Assembler Language*) sigue estando disponible en la mayoría de las herramientas de desarrollo, en nuestro caso con *Keil® Microcontroller Development Kit for ARM* (abreviado como MDK-ARM) y el compilador ARM se puede seleccionar la sintaxis a utilizar.

Para seleccionar la sintaxis UAL se usa la directiva "Thumb" y para trabajar con la versión sintaxis *pre-UAL* se usa la directiva "CODE16". Cabe destacar que en el presente trabajo se estará trabajando plenamente con UAL.

Figura 74. **Ejemplo de código con UAL y pre-UAL**

```
101 Pre-UAL
102 ADD R0, R1 ; R0 = R0 + R1 y actualiza automáticamente APSR
103 UAL
104 ADDS R0, R0, R1 ; R0 = R0 + R1, suma y actualiza APSR
```

Fuente: elaboración propia, empleando Notepad++.

3.1.10. Set de instrucciones

Todas las instrucciones dentro del procesador Cortex-M puede ser divididas en varios grupos dependiendo de su funcionalidad:

- Mover datos dentro del procesador
- Instrucciones de acceso a la memoria
- Operaciones Aritméticas
- Operaciones lógicas
- Prueba y compara
- Control del flujo del programa
- Otro tipo de instrucciones
- Instrucciones basadas en DSP mejorado
- Aspectos para tomar en consideración

3.1.10.1. Mover datos dentro del procesador

La función más básica dentro de los procesadores es el manejo de datos, mover datos de los registros hacia las memorias y viceversa. Muchas veces se tendrá la necesidad de:

- Mover datos de un registro hacia otro.
- Mover datos entre los registros centrales y registros especiales.
- Mover valores inmediatos hacia un registro.
- Mover datos entre el banco de registro principal y el banco de registros de punto flotantes.
- Mover datos entre los registros de sistema para punto flotante (*FPSCR* del inglés *Floating point Status and Control Register*).

En la tabla X se presentan instrucciones para el manejo de datos dentro del procesador.

Tabla X. **Instrucciones para transferencia de datos**

Instrucción	Registro destino	Registro fuente	Operación
MOV	R5,	R1	Copia el valor de R1 a R5
MOVS	R4,	R0	Copia el valor de R0 a R4 y actualiza las banderas De APSR
MRS	R7,	PRIMASK	Copia el valor de PRIMASK (registro especial) hacia R7
MSR	CONTROL	R2	Copia el valor de R2 hacia CONTROL (registro especial)
MOV	R3,	#0x34	Establece el valor inmediato de 0x34 a R3
MOVS	R3,	#0x34	Establece el valor inmediato de 0X34 a R3 y actualiza APSR
MOVW	R6,	#0x1234	Establece un valor constante de 0X1234 a R6

Continuación de la tabla X.

MOVT	R6,	#0X8765	Establece los 16 bits superiores de R5 a un valor de 0x8765
MVN	R3,	R7	Mueve un valor negativo de R7 hacia R3

Fuente: elaboración propia, empleando Microsoft Word.

Cuando se requiere mover datos con punto flotante se requiere el uso de distintas instrucciones a las mostradas en la tabla X. En la tabla XI se muestran las instrucciones para mover datos con punto flotante, en todas las instrucciones tiene el prefijo "V" está presente.

Tabla XI. **Transferir datos entre la FPU y los registros centrales**

Instrucción	Destino	Fuente	Operación
VMOV	R0,	S0	Copia S0 (registro de punto flotante) a R0 (registro de propósito general)
VMOV	S0,	R0	Copia R0 (registro de propósito general) a S0 (registro de punto flotante)
VMOV	S0,	S1	Copia S1 a S0
VMRS.F32	R0,	FPSCR	Copia el valor de FPSCR (del inglés Floating Point Status and Control Register) hacia R0
VMRS	APSR_nzcv,	6	Copia el estado de las banderas FPSCR hacia APSR.
VMSR	FPSCR,	R3	Copia R3 hacia FPSCR
VMOV.F32	S0,	#1.0	Mueve un valor (<i>immediate value</i>) de precisión simple hacia un registro de punto flotante S0

Fuente: elaboración propia, empleando Microsoft Word.

Para configurar un registro dentro del banco de registros (en inglés *Register Bank*) de propósito general un valor inmediato (en inglés *immediate value*) de 8 bits, la instrucción MOVS es suficiente y puede llevarse a cabo con una instrucción Thumb de 16 bits si el registro de destino es un registro bajo (R0 a R7).

Para establecer un valor inmediato (en inglés *immediate value, immed*) con un tamaño entre 9 y 16 bits la instrucción MOVW debe ser utilizada. Cuando se requiere mover un valor inmediato de 32 bits a un registro se puede utilizar la pseudo instrucción "LDR", en la figura 75 se presenta un ejemplo.

Figura 75. Valor de 32 bit en R0

```
46 LDR R0, =0x12345678 ; Establece el valor de 0x12345678 (valor de 32 bits) en R0
```

Fuente: elaboración propia, empleando Notepad++.

3.1.10.2. Instrucciones de acceso a la memoria

Hay una gran cantidad de instrucciones de acceso a la memoria para los procesadores Cortex®-M3 y Cortex®-M4. Esto se debe a la combinación de varios modos de direccionamiento, así como al tamaño y la dirección de transferencia de datos. Para transferencias de datos normales, las instrucciones disponibles se dan en la tabla XII.

Tabla XII. Instrucciones de acceso a memoria

Tipo de dato	Load (lee desde la memoria)	Store (escribe en la memoria)
8 bits sin signo	LDRB	STRB
8 bits con signo	LDRSB	STRB
16 bits sin signo	LDRH	STRH
16 bits con signo	LDRSH	STRH
32 bits	LDR	STR
32 bits múltiples	LDM	STM
Doble-word (64 bits)	LDRD	STRD
Stack operations (32-bit)	POP	PUSH

Fuente: elaboración propia, empleando Microsoft Word.

3.1.10.2.1. PUSH y POP

Para cargar y restablecer múltiples datos en la memoria se usan las instrucciones PUSH y POP. Estas instrucciones usan el puntero de Pila (en inglés *Stack Pointer*) seleccionado recientemente para la generación de direcciones. El puntero de la Pila puede ser MSP (del inglés *Main Stack pointer*) o PSP (del inglés *Process Stack Pointer*), esto se basa en el modo actual del procesador y el valor que se haya configurado en el registro llamado CONTROL. El uso de las instrucciones PUSH y POP se presentan en la tabla XIII.

Tabla XIII. **PUSH y POP**

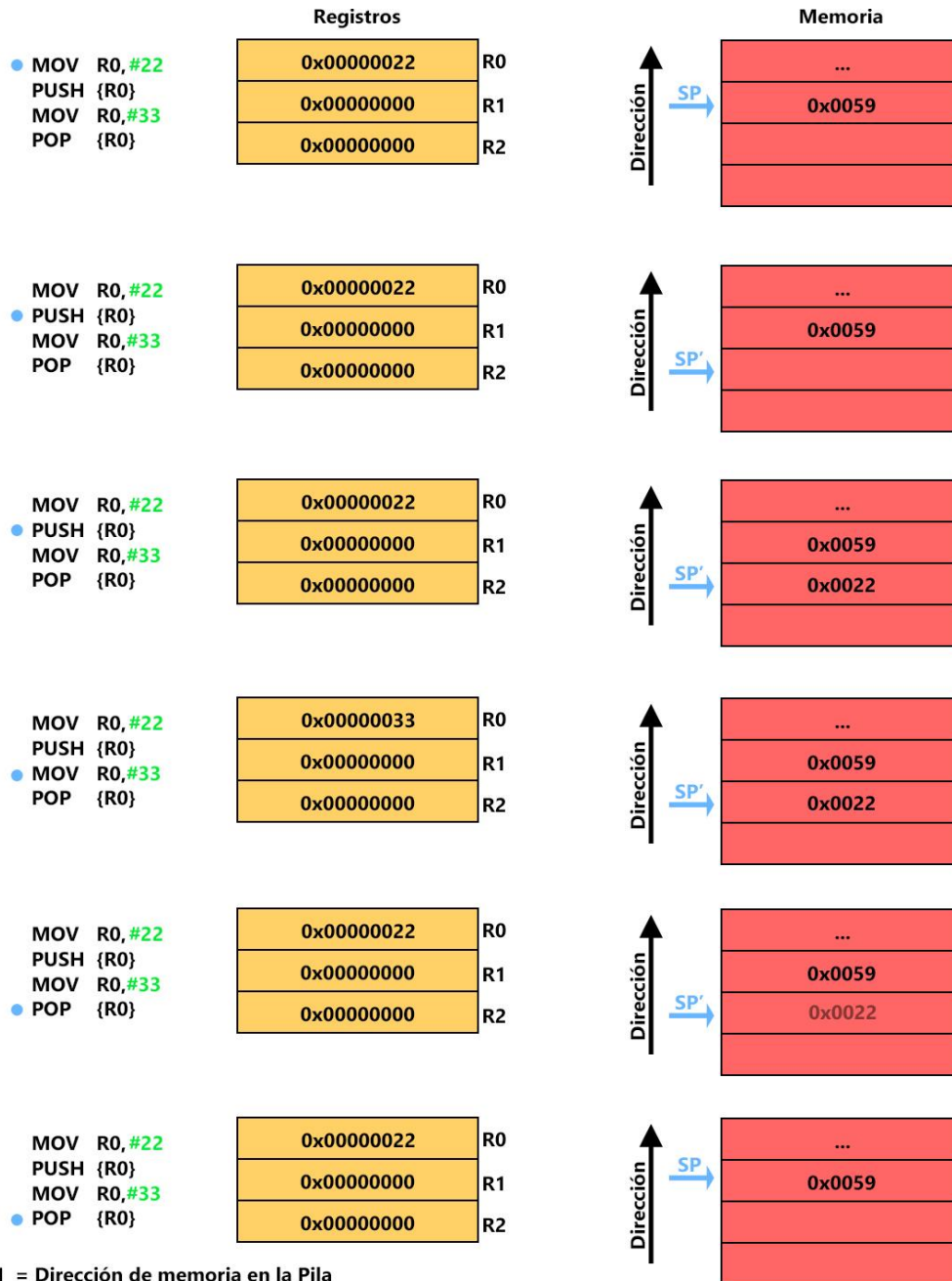
Operaciones en la Pila	Descripción
PUSH <lista de registro>	Almacena registros o registro en la Pila
POP <lista de registro>	Restablece registros o registro de la Pila

Fuente: elaboración propia, empleando Microsoft Word.

En la figura 76 se demuestra el uso correcto de PUSH y POP. El orden del uso de la instrucción PUSH y POP es:

- La primera instrucción señalada con el punto azul carga el valor 0x22 en el registro R0 y SP señala al último valor almacenado en memoria.
- La instrucción PUSH provoca un decremento en la posición señalada por SP, ahora $SP' = SP - 1$.
- La posición de memoria señalada por SP' se utiliza para almacenar el valor 0x22.
- Se mueve un nuevo valor al registro R0, R0 puede ser utilizado en una rutina o subrutina y cambiar de nuevo el valor dentro de R0.
- La posición de memoria señalada por SP' se lee y en este caso se carga al registro R0, R0 vuelve a tener su valor inicial.
- La instrucción POP provoca un incremento en la posición señalada por SP' , ahora $SP = SP' + 1$.
- Ahora SP señala al último valor almacenado en la Pila de memoria.

Figura 76. Carga y descarga de la Pila mediante PUSH y POP



N = Dirección de memoria en la Pila
 SP = Stack Pointer
 SP' = Stack Pointer desfasado una posición en la Pila

Fuente: elaboración propia, empleando Adobe Photoshop.

Cuando el procesador es iniciado, el *Stack Pointer* (SP) se posiciona al final del espacio de memoria reservado para la memoria de Pila. Al momento de cargar un valor a la memoria utilizando PUSH, SP primeramente decrementa su valor y luego almacena el valor en la dirección de memoria señalada por SP. SP siempre señalará o apuntará a la dirección de memoria en donde se cargó el último dato.

Para recuperar un valor cargado a la memoria se usa POP, el valor señalado por SP es leído y recuperado seguidamente el valor de SP aumenta automáticamente. Como se observa en la figura 77 la instrucción POP y PUSH pueden ser utilizadas para almacenar y restablecer más de un valor a la vez.

Figura 77. **Instrucción POP y PUSH**

```
48 PUSH {R0, R4-R7, R9} ; Almacena R0, R4, R5, R6, R7 y R9 dentro de la Pila
49 POP {R2, R3} ; Restablece R2 y R3 de la Pila
```

Fuente: elaboración propia, empleando Notepad++.

Por cada instrucción PUSH se tendrá un POP correspondiente con la misma lista de registros. Las versiones de 16 bits de PUSH y POP son LR (para PUSH) y PC (para POP). Las instrucciones LR y PC están limitadas a registros bajos (R0 a R7). Por lo tanto, si se modifica un registro alto en una función y es necesario guardar el contenido del registro, debe utilizar las instrucciones PUSH y POP las cuales tienen un tamaño de 32 bits.

Existe la variante de PUSH y POP para datos de punto flotante las cuales son VPUSH y VPOP respectivamente, ver tabla XIV. VPUSH y VPOP funcionan de la misma manera con el detalle de que son exclusivas para datos con punto flotante. Cabe destacar que VPUSH y VPOP requieren que:

- Los registros de la lista de registros sean consecutivos.
- El número máximo de registro con punto flotante para VPUSH y VPOP son de 16 en total.

Tabla XIV. **VPUSH y VPOP**

Operación para la Pila de Memoria	Descripción
VPUSH.32 <Lista de registros S>	Almacena datos con punto flotante de precisión simple por ej.: S0-S31.
VPUSH.64 < Lista de registros D>	Almacena datos con punto flotante de precisión doble por ej.: D0-D15.
VPOP.32 < Lista de registros S >	Restablece datos con punto flotante de precisión simple.
VPOP.64 < Lista de registros D >	Restablece datos con punto flotante de precisión doble.

Fuente: elaboración propia, empleando Microsoft Word.

3.1.10.3. Operaciones Aritméticas

La familia de procesadores Cortex®-M4 contiene una gran cantidad de instrucciones para realizar operaciones aritméticas. Se presentan las instrucciones más utilizadas, se debe tomar en cuenta que una instrucción puede tener varias formas de ser escritas. Cuando se agrega prefijo o sufijo a una instrucción esta logra realizar más de una función, como por ejemplo sumar y a su vez actualizar el registro APSR. En el caso de la instrucción ADD, como se muestra en la figura 78.

Figura 78. **Variantes de la instrucción ADD**

```
51 ADD R0, R0, R1 ; R0 = R0 + R1
52 ADDS R0, R0, #0x12 ; R0 = R0 + 0x12 & Actualiza las banderas de APSR
53 ADC R0, R1, R2 ; R0 = R1 + R2 + carry/acarreo
```

Fuente: elaboración propia, empleando Notepad++.

Todas las instrucciones que se muestran en la figura 78 son sumas (ADD) pero escritas de distinta forma, en código binario son completamente distintas. Se debe tener en cuenta que al utilizar la tecnología Thumb-2 se debe especificar cuándo se necesita actualizar las banderas del registro APSR.

Por defecto en las operaciones aritméticas, cuando se tiene una división (UDIV y SDIV) entre cero el resultado automáticamente será cero. Se puede configurar una excepción para la división entre cero en el registro NVIC de control de configuración en el bit llamado DIVBYZERO. En la tabla XV se presentan las instrucciones aritméticas más utilizadas en los procesadores Cortex®-M.

Tabla XV. Instrucciones para operaciones aritméticas

Instrucciones (Vi = Valor inmediato)	Operaciones
ADD Rd, Rn, Rm; Rd = Rn+Rm	Suma
ADD Rd, Rn, #Vi ; Rd = Rn+ #Vi	Suma
ADC Rd, Rn, Rm; Rd = Rn+Rm+acarreo	Suma con acarreo
ADC Rd, #Vi; Rd = Rd+#Vi+acarreo	Suma con acarreo
ADDW Rd, Rn, #Vi ;Rd = Rn+#Vi	Suma un registro con un valor inmediato de tamaño de 12 bits
SUB Rd, Rn, Rm; Rd = Rn - Rm	Resta
SUB Rd, #Vi ;Rd=Rd-#Vi	Resta
SUB Rd, Rn,#Vi ; Rd = Rn - #Vi	Resta
SBC Rd, Rn, #Vi; Rd=Rn-#Vi-(valor prestado)	Resta con valor prestado (no acarreo)
SBC Rd, Rn, Rm ; Rd = Rn-Rm-(valor prestado)	Resta con valor prestado (no acarreo)
SUBW Rd, Rn,#Vi ; Rd = Rn - #Vi	Resta un registro con un valor inmediato con un tamaño de 12 bits
RSB Rd, Rn, #Vi ; Rd = #Vi - Rn	Resta inversa
RSB Rd, Rn, Rm; Rd = Rm - Rn	Resta inversa
MUL Rd, Rn, Rm; Rd = Rn * Rm	Multiplicación con resultado de 32 bits
UDIV Rd, Rn, Rm; Rd = Rn/Rm	División sin signo
SDIV Rd, Rn, Rm ; Rd = Rn/Rm	División con signo

Fuente: elaboración propia, empleando Microsoft Word.

La familia de procesadores Cortex®-M3 y Cortex®-M4 admiten instrucciones de multiplicación de 32 bits e instrucciones de acumulación de multiplicación (en inglés MAC) que dan resultados de 32 y 64 bits. En la tabla XVI se presentan las instrucciones para multiplicar y MAC las cuales admiten valores sin signo y con signo.

Tabla XVI. **Instrucciones para multiplicar y MAC**

Instrucción (Hi = mayor, Lo = menor)	Operación
MLA Rd, Rn, Rm, Ra; Rd = Ra + Rn * Rm	Instrucción MAC de 32 bits con resultado de 32 bits
MLS Rd, Rn, Rm, Ra; Rd = Ra - Rn * Rm	Multiplicación de 32 bits con instrucción resta, se obtiene un resultado de 32 bit
SMULL RdLo, RdHi, Rn, Rm; {RdHi,RdLo} = Rn* Rm	Multiplicación de 32 bits e instrucción MAC para valores con signo, resultado de 64 bits
SMLAL RdLo, RdHi, Rn, Rm; {RdHi,RdLo} += Rn * Rm	Multiplicación de 32 bits e instrucción MAC para valores con signo, resultado de 64 bits
UMULL RdLo, RdHi, Rn, Rm; {RdHi,RdLo} = Rn * Rm	Multiplicación de 32 bits e instrucción MAC para valores sin signo, resultado de 64 bits
UMLAL RdLo, RdHi, Rn, Rm; {RdHi,RdLo} += Rn * Rm	Multiplicación de 32 bits e instrucción MAC para valores sin signo, resultado de 64 bits

Fuente: elaboración propia, empleando Microsoft Word.

3.1.10.4. Operaciones lógicas

Los procesadores de la familia Cortex®-M3 y Cortex®-M4 admiten instrucciones para realizar álgebra booleana mediante operaciones lógicas como: AND, OR, XOR y muchas más.

Como en las instrucciones aritméticas la versión de 16 bit de las operaciones lógicas actualiza automáticamente las banderas del registro APSR. Si no se usa el prefijo “S” el ensamblador convertirá las instrucciones a un tamaño de 32 bits.

Para utilizar la versión de 16 bits de las instrucciones lógicas se requiere que las operaciones sean entre dos registros, en donde uno de ellos es el destino y a su vez el registro fuente. Se debe recordar que los únicos registros disponibles para instrucciones de 16 bits son los registros en el rango de R0 a R7 y se debe especificar el prefijo “S” al momento de usar una instrucción. En la tabla XVII se presentan las instrucciones para realizar algebra booleana.

Tabla XVII. **Instrucciones para álgebra booleanas**

Instrucciones (V_i = valor inmediato)	Operación bit a bit
AND Rd, Rn; Rd = Rd & Rn	AND
AND Rd, Rn, #Vi; Rd = Rn & #Vi	AND
AND Rd, Rn, Rm; Rd = Rn & Rm	AND
ORR Rd, Rn; Rd = Rd Rn	OR
ORR Rd, Rn, #Vi; Rd = Rn #Vi	OR
ORR Rd, Rn, Rm; Rd= Rn Rm	OR
BIC Rd, Rn; Rd = Rd & (~Rn)	Limpia el bit
BIC Rd, Rn, #Vi ; Rd = Rn & (~#Vi)	Limpia el bit
BIC Rd, Rn, Rm; Rd = Rn & (~Rm)	Limpia el bit
ORN Rd, Rn, #Vi; Rd = Rn (~#Vi)	OR NOT
ORN Rd, Rn, Rm; Rd= Rn (~Rm)	OR NOT
EOR Rd, Rn; Rd = Rd ^ Rn	OR Exclusiva
EOR Rd, Rn, #Vi; Rd = Rn #Vi	OR Exclusiva
EOR Rd, Rn, Rm; Rd = Rn Rm	OR Exclusiva

Fuente: elaboración propia, empleando Microsoft Word.

3.1.10.5. Prueba y compara

Estas instrucciones se utilizan para actualizar las banderas del registro APSR, las cuales luego pueden ser utilizadas por un salto (en inglés *Branch*) condicional o una ejecución condicional, en la tabla XVII se presentan las instrucciones. Estas instrucciones no usan el prefijo “S” debido que siempre se actualizarán el registro APSR.

Tabla XVIII. **Instrucciones de prueba y compara**

Instrucciones (Vi=Valor inmediato)	Operación
CMP <Rn>, <Rm>	Compara; Calcula Rn menos Rm. Actualiza el registro APSR pero no lo almacena.
CMP <Rn>, #<Vi>	Compara; Calcula Rn menos Vi.
CMN <Rn>, <Rm>	Comparación negativa: Calcula Rn+Rm. Actualiza el registro APSR pero no lo almacena.
CMN <Rn>, #<Vi>	Comparación negativa: Calcula Rn+Vi. Actualiza el registro APSR pero no lo almacena.
TST <Rn>, <Rm>	Prueba (AND); Calcula Rn AND Rm. El bit N y el bit Z en APSR se actualizan, pero el resultado no es guardado.
TST <Rn>, #<Vi>	Prueba (AND); Calcula Rn AND Vi. El bit N y el bit Z en APSR se actualizan, pero el resultado no es guardado.
TEQ <Rn>, <Rm>	Prueba (XOR); Calcula Rn XOR Rm. El bit N y el bit Z en APSR se actualizan, pero el resultado no es guardado.
TEQ <Rn>, #<Vi>	Prueba (XOR); Calcula Rn XOR Vi. El bit N y el bit Z en APSR se actualizan, pero el resultado no es guardado.

Fuente: elaboración propia, empleando Microsoft Word.

3.1.10.6. Control del flujo del programa

Existen varios tipos de instrucciones para controlar el flujo del programa:

- Salto (Branch)
- Salto Condicional
- Llamada de función
- Comparación combinada y salto condicional
- Ejecución condicionada

3.1.10.6.1. Saltos (*Branch*)

En general las instrucciones más utilizadas, en programación en lenguaje ensamblador, para realizar saltos dentro del código son:

- B (Branch)
- BX (del inglés *Branch with Exchange*)
- POP

En la tabla XIX se presentan las instrucciones disponibles en los procesadores de la familia Cortex®-M3 y Cortex®-M4.

Tabla XIX. **Instrucciones de salto sin condición**

Instrucción	Operación
B <etiqueta>	Salto hacia una <etiqueta>
B.W <etiqueta>	La versión de <i>B</i> para 32-bits es <i>B.W</i> , cubre un rango de -2KB a +2KB
BX <Rm>	Salto y cambio. Salta a un valor que define una dirección almacenada en Rm y establece el estado de ejecución del procesador (T-bit) basado en el bit 0 de Rm. El bit 0 de Rm debe ser 1 porque el procesador Cortex®-M solo admite el estado <i>Thumb</i> .

Fuente: elaboración propia, empleando Microsoft Word.

3.1.10.6.2. Saltos (*Branches*) condicionados

Los saltos condicionales se ejecutan en función del valor actual de las banderas dentro del registro APSR (N, Z, C y V). En la tabla XX se presentan las banderas y sus respectivas descripciones.

Tabla XX. **Estado de los bits de las banderas del registro APSR**

Bandera	Bit PSR	Descripción de la condición
N	31	Bandera negativa, la última operación dio como resultado un valor negativo.
Z	30	Cero, la última operación dio un resultado igual a cero. Ej: Cuando se comparan dos registros con el mismo valor se obtiene un resultado igual a cero.
C	29	Acarreo, la última operación da como resultado un acarreo.
V	28	Desbordamiento, la última operación causó un desbordamiento.

Fuente: elaboración propia, empleando Microsoft Word.

La condición requerida para que tenga lugar un salto condicional se indica mediante un sufijo <condición> como se muestra en la tabla XXI.

Tabla XXI. **Instrucciones para salto condicionado**

Instrucción	Operación
B<condición> <etiqueta>	Salta a la <etiqueta> si la condición es verdadera. Ej: CMP R3, #12 BEQ loop; Salta a la etiqueta "loop" si R3=12
B<condición>.W <etiqueta>	La versión de 32 bits para salto condicionado es B.W. la cual es para rango amplio de -254 bytes a +254 bytes

Fuente: elaboración propia, empleando Microsoft Word.

En la tabla XXII se muestran los 14 posibles sufijos de <condición> que pueden ser utilizados en las instrucciones presentes en la tabla XXI.

Tabla XXII. **Sufijos condicionales**

Prefijo	Condición para realizar salto	Banderas (APSR)
EQ	Igual	Z==1
NE	no es igual	Z==0
CS/HS	Establece acarreo	C==1
CC/LO	limpia el acarreo	C==0
MI	menos	N==1
PL	Más	N==0
VS	Desbordamiento	V==1
VC	Sin desbordamiento	V==0
HI	Mayor que, sin signo, >	(C==1) && (Z==0)
LS	Menor que o igual, sin signo, <=	(C==0) (Z==0)
AL	Se ejecuta siempre	-
NV	Nunca se ejecuta	-

Fuente: elaboración propia, empleando Microsoft Word.

El flujo del programa como se muestra en la figura 79 se puede implementar usando instrucciones de salto condicional y salto simple.

Figura 79. **Ejemplo de uso instrucciones de salto**

```

55     CMP     R0, #22 ; Compara R0 con 22
56     BEQ     F2      ; Si R0 = 22 entonces realiza un salto (Branch) hacia F2
57     MOVS    R3, #1  ; El registro R1 sera igual a 1, R3=1
58     B       F3      ; Realiza un salto (Branch) hacia F3
59 F2   ; Etiqueta (label) F2
60     MOVS    R3, #2  ; R3=2
61 F3   ;
62     ...           ;Instrucciones y operaciones dentro de la etiqueta F3

```

Fuente: elaboración propia, empleando Notepad++.

3.1.10.6.3. Llamada de función

Para invocar o llamar a una función en cualquier parte del código las instrucciones BL (del inglés *Branch and Link*) y BLX (del inglés *Branch and Link eXchange*) pueden ser usadas, como se muestra en tabla XXIII. BL y BLX ejecutan un salto (en inglés *Branch*) y al mismo tiempo guardan la dirección desde donde se ha saltado y se almacena en LR (del inglés *Link Register*). Al finalizar las operaciones de la función llamada (hacia donde se ha saltado) se puede regresar a la posición guardada en LR (posición inicial).

Tabla XXIII. **Instrucción para llamar o invocar a una función**

Instrucción	Descripción
BL <etiqueta>	Salta a la dirección de la <etiqueta> y guarda la dirección de retorno (dirección de donde se ha saltado) en LR.
BLX <Rm>	Salta a la dirección almacenada en Rm, guarda la dirección de retorno en LR, y actualiza T-bit dentro de EPSR con el LSB de Rm

Fuente: elaboración propia, empleando Microsoft Word.

3.1.10.6.4. Compara y salto condicional

Dentro de los procesadores Cortex®-M con arquitectura ARMv-7 se encuentran dos instrucciones para proporcionar una operación de salto condicionado con base en una comparación con cero. Estas dos instrucciones son:

- CBZ (compara y realiza un salto si la operación es igual a cero)

- CBNZ (compra y realiza un salto si el resultado no es igual a cero)

Las instrucciones CBZ y CBNZ son útiles cuando se requiere realizar un bucle (en inglés *loop*) condicionado como puede ser el bucle *while*. En la figura 80 se presenta el código con un ciclo *while* compilador en lenguaje de programación C.

Figura 80. **Ejemplo de CBZ en código C**

<pre>int i = 5; while (i != 0) funcion_1() i--; }</pre>	<pre>Inicia el contador en 5 Mientras i no sea igual a 0, i != 0 Llama o invoca una función Decrementa el valor de i hasta llegar a 0</pre>
------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------

Fuente: elaboración propia, empleando Notepad++.

El código del ejemplo mostrado en la figura 80 puede ser escrito en lenguaje ensamblador utilizando la instrucción CBZ. En la imagen 81 se presenta el correcto uso de la instrucción CBZ.

Figura 81. **Ejemplo de la instrucción CBZ**

```
74     MOV R0, #5 ; R0=5, R0 almacena el valor del contador y se inicia con valor de 5
75     Bucle_1 CBZ R0, Salir_Bucle ; Si el contador R0 = 0 entonces se saldrá del bucle (Salir_Bucle)
76     BL Funcion_1 ; Llama o invoca a la función Función_1
77     SUBS R0, #1 ; El valor del contador decrementa, al valor asignado a R0 se le resta 1
78     B Bucle_1 ; Salto (Branch) hacia Bucle_1
79     Salir_Bucle
```

Fuente: elaboración propia, empleando Notepad++.

En la figura 82 se presenta un ejemplo básico utilizando Java, el usuario ingresa un número. Si el número ingresado es igual a 0 se mostrará un mensaje

de "ERROR", si el número ingresado es distinto a 0 se aprobará el número y se procederá a realizar un salto hacia otras funciones.

Figura 82. Ejemplo de CBNZ en Java

```
1 import java.util.Scanner;
2 public class Main {
3     public static void main(String[] args) {
4         Scanner teclado=new Scanner(System.in);
5         float n;
6         System.out.println("Ingrese un número:"); //El usuario ingresara un valor numérico
7         n=teclado.nextInt(); // Almacena el valor ingresado en la variable n
8
9         if (n==0){
10            System.out.println("Número no valido, Error");//si el valor ingresado=0, dara un error
11        }else{
12            System.out.println("Número Valido");//Si el valor es distinto a 0, se seguirá ejecutando
                el programa y se podrá llamar a otras funciones
13        }
14    }
15 }
```

Fuente: elaboración propia, empleando Notepad++.

La lógica implementada en el ejemplo de la figura 82 se puede implementar en lenguaje ensamblador. Se realiza un salto (Branch) si la bandera Z no se establece (el resultado no es cero), como se presenta en la figura 83.

Figura 83. Ejemplo de la instrucción CNBZ

```
82 BL NextInit ;Salta a la funcion NextInit para obtener el valor a comparar
83 CNBZ R0, Valor_ingresado_Valido ; Realiza un salto a la funcion Valor_ingresado_Valido, si el resultado no es cero R0/=0
84 BL Muestra_Error ; Si el resultado es igual a 0 salta a la funcion Muestra_Error
85 BL exit
86 Valor_ingresado_Valido
87 ...
```

Fuente: elaboración propia, empleando Notepad++.

3.1.10.7. Otro tipo de instrucciones

Dentro del procesador Cortex®-M están presentes una gran variedad de instrucciones las cuales son de gran utilidad. Una de las instrucciones a tomar en cuenta es la instrucción NOP.

3.1.10.7.1. NOP

Esta instrucción puede ser usada para producir alineación de instrucciones o introducir demoras.

Figura 84. **Ejemplo del uso de NOP**

```
91 NOP; no se hace nada durante un ciclo de reloj
```

Fuente: elaboración propia, empleando Notepad++.

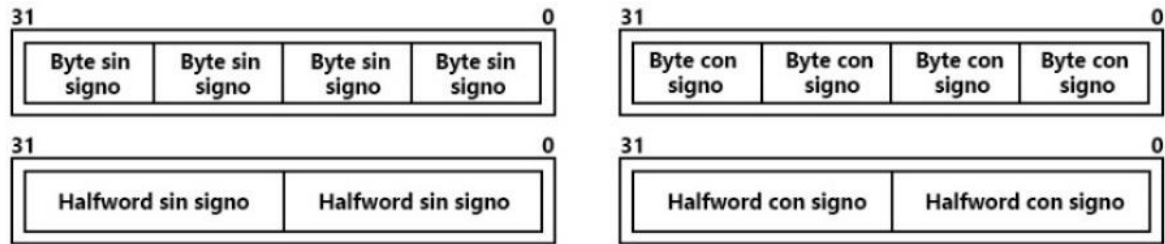
Debe tener en cuenta que el retardo producido por esta instrucción puede variar entre diferentes sistemas, puede no ser un ciclo de reloj preciso. Si el retardo de tiempo debe ser preciso se debe usar un temporizador por hardware.

3.1.11. Instrucciones específicas para Cortex®-M4 basadas en DSP mejorado

Las instrucciones basadas en DSP (del inglés *digital signal processor*) permiten a los procesadores Cortex®-M4 manejar y procesar señales digitales en tiempo real. Los datos que se procesan comúnmente son de tamaño de 16 u 8 bits. Se puede trabajar con audio muestreado con un ADC con resolución de 16 bits, también puede trabajar con datos de una imagen los cuales son representados mediante múltiples canales o matrices de 8 bits.

Los procesadores de la familia Cortex®-M posee rutas (en inglés *paths*) de 32 bits por lo que pueden manejar datos de 2 x 16 bits o de 4 x 8 bits, cabe destacar que los datos pueden ser datos sin signo o con signo. Por lo anterior, en la figura 85 se presentan las posibles configuraciones de datos SIMD.

Figura 85. Datos SIMD posibles en un registro de 32 bits



Fuente: elaboración propia, empleando Adobe Photoshop.

3.1.11.1. Multiplicación e instrucciones MAC

Las instrucciones de multiplicación e instrucciones MAC (del inglés *multiply-accumulate*) disponibles en el procesador Cortex®-M3 y Cortex®-M4 se presentan en la tabla XXIV.

Tabla XXIV. Instrucciones de multiplicación y MAC

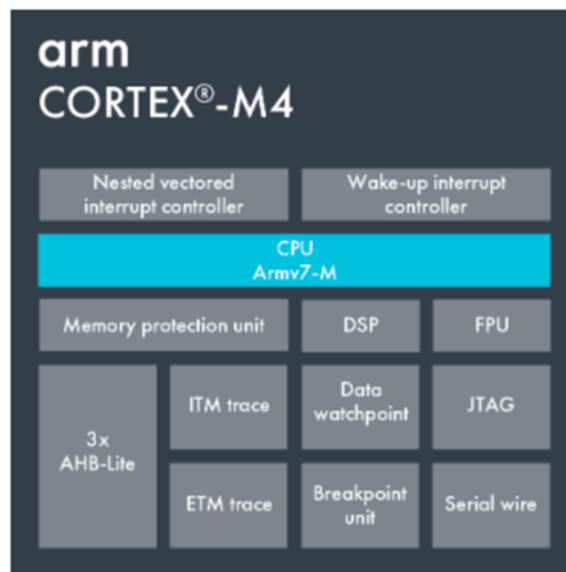
Instrucciones	Descripción & tamaño	Bandera
MUL / MULS	Multiplica sin signo, 32 bits x 32 bits = 32 bits	ninguna, o N y Z
UMULL	Multiplica sin signo, 32 bits x 32 bits= 64 bits	ninguna
UMLAL	MAC sin signo, ((32 bits x 32 bits) + 64 bits = 64 bits)	ninguna
SMULL	Multiplicación con signo, 32 bits x 32 bits = 64 bits	ninguna
SMLAL	MAC con signo, ((32 bits x 32 bits) + 64 bits = 64 bits)	ninguna

Fuente: elaboración propia, empleando Microsoft Word.

3.1.11.2. Instrucciones de punto flotante

Los procesadores Cortex®-M4 pueden traer una unidad para punto flotante, lo que les brinda el soporte para operar datos con punto flotante (número con decimales). Cuando se implementa el CPU Armv7-M dentro de un Cortex®-M4 se tiene la posibilidad de poder manejar datos con puntos flotantes aparte de poder realizar más tareas como se muestra en la imagen 86.

Figura 86. Diagrama del procesador Cortex®-M4



Fuente: DESIGN & REUSE. *Cortex-M4*. <https://www.design-reuse.com/sip/blockdiagram/43824/9-main-Cortex-M4.png>. Consulta: agosto de 2020.

Primeramente, se presentarán los distintos tipos de datos con punto flotante y el formato de cada uno de ellos. Los datos con punto flotante son de:

- Precisión simple

- Precisión media
- Doble precisión

3.1.11.3. Números con punto flotante de precisión simple

El formato que se utiliza en este tipo de datos se presenta en la imagen 87.

Figura 87. Formato de datos de precisión simple



Fuente: elaboración propia, empleando Adobe Photoshop.

Se debe tener en cuenta que para el manejo de este tipo de datos el procesador Cortex®-M4 con Armv7-M basa el cálculo de datos en punto flotante con base en el estándar IEEE 754. Para este tipo de datos se utilizará la ecuación mostrada en la figura 88.

Figura 88. Formato normalizado para datos con precisión simple

$$\text{Value} = (-1)^{\text{Sign}} \times 2^{(\text{exponent} - 127)} \times (1 + (\frac{1}{2} * \text{Fracion}[22]) + (\frac{1}{4} * \text{Fracion}[21]) + (\frac{1}{8} * \text{Fracion}[20]) \dots (1/(2^{23}) * \text{Fracion}[0]))$$

Fuente: elaboración propia, empleando Adobe Photoshop.

En la tabla XXV se muestran valores calculados con base en la ecuación de la figura 88.

Tabla XXV. **Ejemplo para valores con punto flotante**

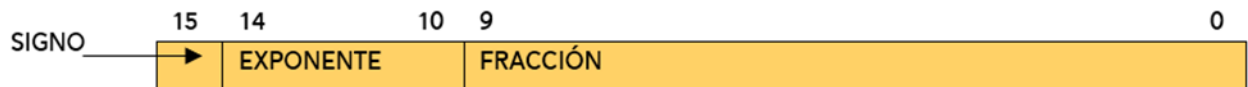
Valor con punto flotante	Signo	Exponente	Fracción	Valor hexadecimal
1	0	127 (0x7F)	000_0000_0000_0000_0000_0000	0x3F800000
1.5	0	127 (0x7F)	100_0000_0000_0000_0000_0000	0x3FC00000
1.75	0	127 (0x7F)	110_0000_0000_0000_0000_0000	0x3FE00000
-4.75 → -1.1875 *2 ²	1	127 + 2 = 129 (0x81)	001_1000_0000_0000_0000_0000	0xC0980000

Fuente: elaboración propia, empleando Microsoft Word.

3.1.11.4. **Números con punto flotante de precisión media**

Este tipo de números utiliza menos bits en el campo del exponente y el campo de fracción comparado con el formato de precisión simple. En la figura 89 se presenta el formato de este tipo de números.

Figura 89. **Formato de datos de precisión media**



Fuente: elaboración propia, empleando Adobe Photoshop.

La ecuación normalizada basándose en el estándar IEEE 754 se presenta en la figura 90.

Figura 90. **Formato normalizado para datos de precisión media**

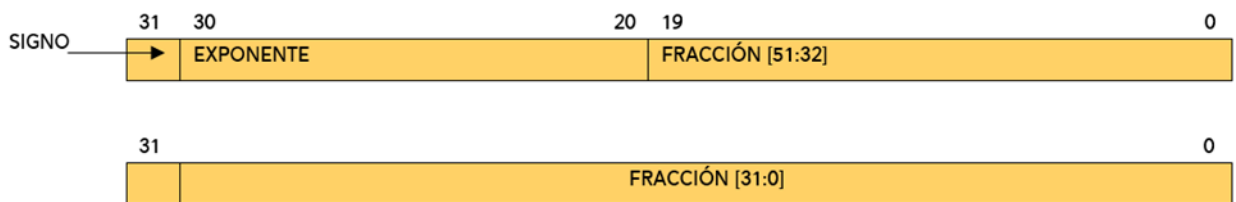
$$\text{Value} = (-1)^{\text{Sign}} \times 2^{(\text{exponent} - 15)} \times (1 + (\frac{1}{2} * \text{Fracion}[9]) + (\frac{1}{4} * \text{Fracion}[8]) + (1/8 * \text{Fracion}[7]) \dots (1/(2^{10}) * \text{Fracion}[0]))$$

Fuente: elaboración propia, empleando Adobe Photoshop.

3.1.11.5. **Números con punto flotante de doble precisión**

Aunque el procesador Cortex®-M4 no admite de forma nativa operaciones con punto flotante de doble precisión, sí se puede tener este tipo de datos en las aplicaciones. El formato de datos con doble precisión se muestra en la figura 91.

Figura 91. **Formato de datos de precisión doble**



Fuente: elaboración propia, empleando Adobe Photoshop.

En un sistema Little Endian la palabra menos significativa se almacena en la dirección inferior (bit 0) de un total de 64 bits disponibles, y la palabra más

significativa se almacena en la dirección superior (bit 64 o 32x2 bits). En un sistema Big Endian es al revés.

Cuando el exponente tiene un valor entre 0 y 0x7FF, el valor es un valor normalizado y el valor del número de doble precisión se podrá representar con la ecuación que se muestra en la figura 92:

Figura 92. **Formato normalizado para datos con precisión doble**

$$\text{Value} = (-1)^{\text{Sign}} \times 2^{(\text{exponent} - 1023)} \times (1 + (\frac{1}{2} * \text{Fracion}[51]) + (\frac{1}{4} * \text{Fracion}[50]) + (\frac{1}{8} * \text{Fracion}[49]) \dots (1/(2^{52}) * \text{Fracion}[0]))$$

Fuente: elaboración propia, empleando Adobe Photoshop.

3.1.12. Unidad de punto flotante en Cortex®-M4

Las operaciones con punto flotante están basadas en el estándar IEEE 754-2008, sin embargo, no se tiene la implementación completa de dicho estándar con respecto a las operaciones o funciones. La unidad que procesa los datos de punto flotante es opcional y solo tiene soporte para cálculos con datos de punto flotante de precisión simple además de poder realizar conversiones y algunas funciones de acceso a memoria.

Debido a que no se cuenta con la implementación completa del estándar IEEE 754, algunas operaciones deben de ser manejadas mediante software y no por hardware. Las instrucciones implementadas mediante software son:

- Operaciones con datos de punto flotante de doble precisión.
- Cálculo del residuo en datos con punto flotante.
- Número de punto flotante redondeado a valores enteros.

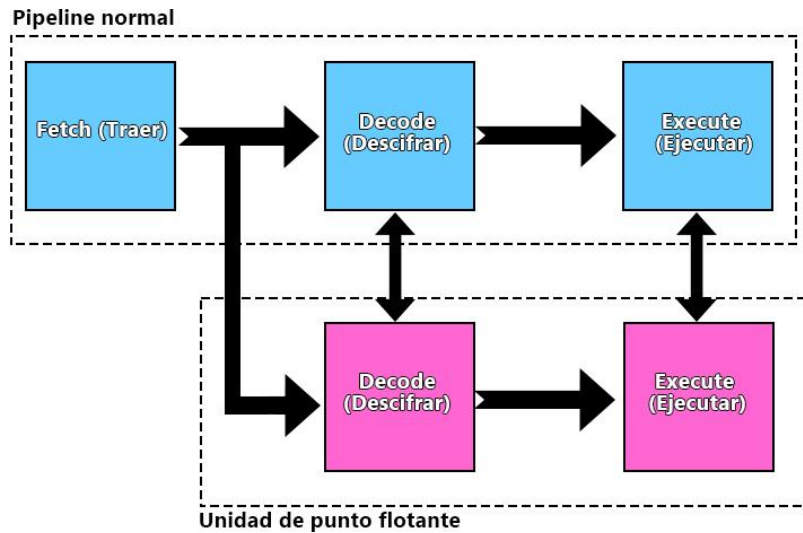
- Conversión de binario a decimal y viceversa.
- Comparación de datos de punto flotante, entre precisión simple y precisión doble.

Con la unidad de punto flotante se tiene:

- Un banco de registros de punto flotante, 32 registros de precisión simple en total.
- Cálculos de datos con punto flotante de precisión simple.
- Conversión de instrucciones.
 - Entero ↔ punto flotante de precisión simple
 - Fixed Point ↔ punto flotante de precisión simple
 - Media precisión ↔ punto flotante de precisión simple
- Transferencia de datos de precisión simple y de precisión doble entre el banco de registros para punto flotante (en inglés FPRB, *Floating point register bank*) y la memoria.
- Transferencia de datos de precisión simple entre FPRB y el banco de registros para enteros (banco de registro central).

Desde el punto de vista de la arquitectura, la unidad de punto flotante es vista como un coprocesador. El *Pipeline* del procesador y la unidad de punto flotante comparten la misma etapa de obtención de instrucciones, pero las etapas de decodificación y ejecución de instrucciones son independientes, como se muestra en la figura 93.

Figura 93. **Concepto de pipeline en coprocesador**



Fuente: elaboración propia, empleando Adobe Photoshop.

Sin embargo, para operaciones de punto flotante y transferencias de datos con punto flotante dentro del procesador Cortex®-M4, se utiliza un conjunto de instrucciones dedicadas para punto flotante en lugar de instrucciones de acceso al coprocesador.

La unidad de punto flotante agrega registros extras al sistema del procesador, se listan los recursos agregados por FPU (del inglés *Floating Point Unit*):

- CPACR (del inglés *Co-Processor Access Control Register*) en SCB (del inglés *System Control Block*).
- Banco de registro para punto flotante.
- Registro de control y estado de punto flotante (FPSCR).

- Registros adicionales en FPU para operaciones y control de datos con punto flotante, como se observa en la tabla XXVI.

Tabla XXVI. **Registros FPU adicionales**

Dirección	Registros	Símbolos CMSIS del núcleo	Función
0xE000EF34	Registro de control de contexto de punto flotante	FPU->FPCCR	Control de datos en FPU.
0xE000EF38	Registro de direcciones de contexto de punto flotante	FPU->FPCAR	Mantiene la dirección de un registro de punto flotante en el <i>Stak Pointer</i> .
0xE000EF3C	Registro de control de estado predeterminado de punto flotante	FPU->FPDSCR	Valores predeterminados para los datos de control de estado para punto flotante (FPCCR).
0xE000EF40	Características de Media y FP en registro 0	FPU->MVFR0	Información de solo lectura sobre las instrucciones VFP implementadas.
0xE000EF44	Características de Media y FP en registro 1	FPU->MVFR1	Información de solo lectura sobre las instrucciones VFP implementadas.

Fuente: elaboración propia, empleando Microsoft Word.

energético. Para habilitar la FPU se debe configurar el registro CPACR para que permita operar datos con punto flotante como se muestra en la figura 95.

Figura 95. **Configuración del registro CPACR**

```
106 SCB->CPACR|= 0x00F00000;Habilita con acceso completo la unidad de punto flotante
```

Fuente: elaboración propia, empleando Adobe Photoshop.

3.1.12.2. Banco de registros para punto flotante

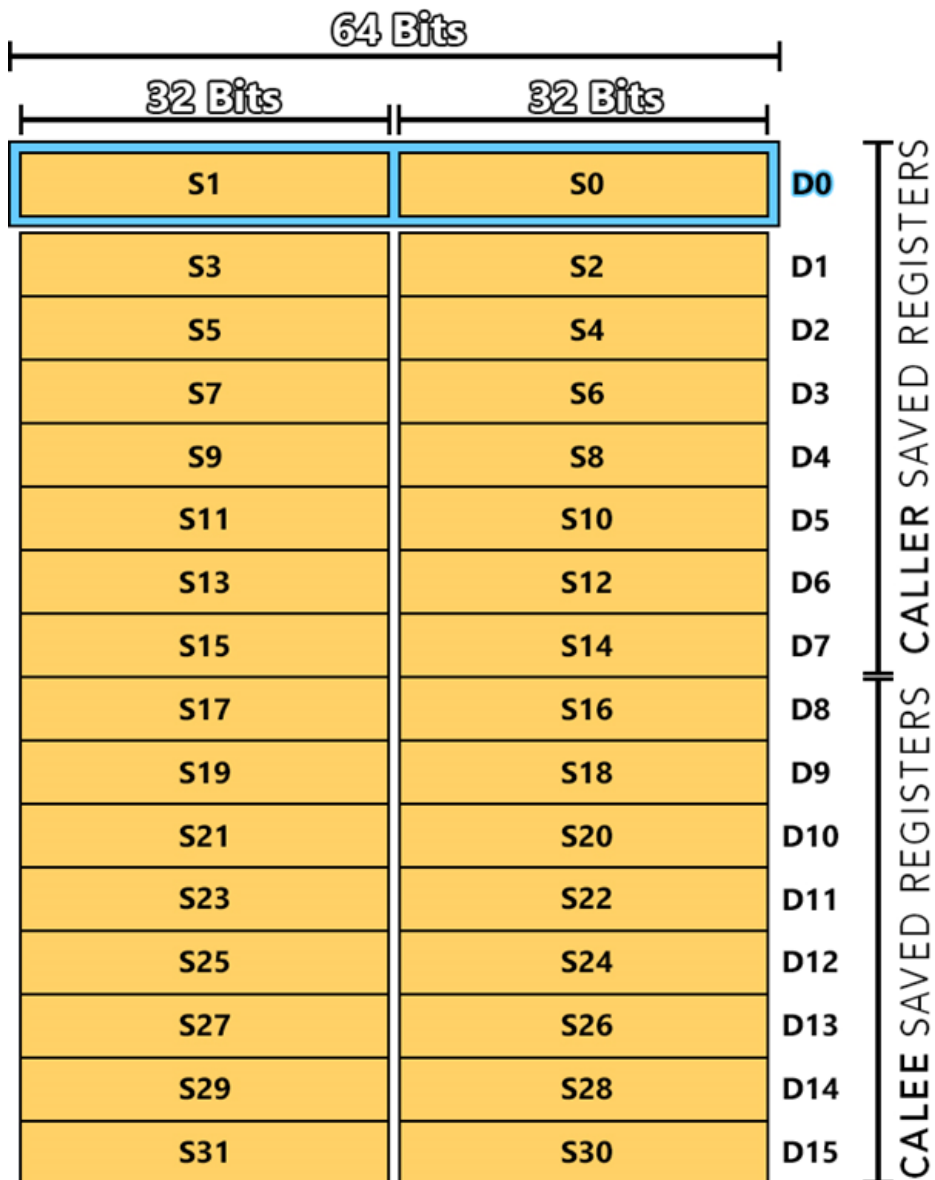
El banco de registros para punto flotante está compuesto por 32 registros cada uno de ellos con un tamaño de 32 bits, también se pueden ver como 16 registros cada uno de ellos con un tamaño de 64 bits. Los registros del rango de S0 a S15 son llamados *caller saved registers* (posee registros volátiles, almacenan datos temporalmente y es la función que invoca o llama a otra función).

Cabe destacar que al momento de llamar funciones dentro del código los *registros* (S0 a S15) deben ser guardados previamente debido a que estos pueden cambiar al momento de saltar de una función a otra. Los registros en el rango de S16 a S31 son *callee saved registers* (posee registros no volátiles, los cuales contienen datos de larga duración que deben conservarse en todas las llamadas y es la función llamada o invocada por el *caller*).

Igualmente, al saltar de una función a otra usando los registros S16 a S31 se debe almacenar los datos de los registros en la Pila, antes de regresar a la función de donde se ha saltado se debe regresar el valor original de los registros (S16 a S31). Los valores iniciales de los registros de punto flotante son

indefinidos, en la figura 96 se presenta gráficamente los registros y como están distribuidos.

Figura 96. **Registros para datos con punto flotante**



Fuente: elaboración propia, empleando Adobe Photoshop.

3.1.12.3. FPSCR

El registro FPSCR contiene el estado de las banderas de las operaciones lógicas, aritméticas y los campos para poder controlar la unidad de punto flotante (FPU). El registro FPSCR está compuesto de varias banderas como se muestra en la figura 97.

Figura 97. Bits en el registro FPSCR

	31	30	29	28	27	26	25	24	23:22	21:08	7	6:05	4	3	2	1	0
FPSCR	N	Z	C	V	RESERVADO	AHP	DN	FZ	MODO R	RESERVADO	IDC	RESERVADO	IXC	UFC	OFC	DZC	IOC

Fuente: elaboración propia, empleando Adobe Photoshop.

En la tabla XXVIII se encuentra la descripción de cada uno de los campos mostrados en la figura 97.

Tabla XXVIII. Descripción de bits en FPSCR

Bit	Descripción
N	Bandera negativa, se actualiza mediante operaciones de comparación de punto flotante.
Z	Bandera cero, se actualiza mediante operaciones de comparación de punto flotante.
C	Bandera de acarreo o pedir prestado, actualización mediante operaciones de comparación de punto flotante.
V	Bandera de desbordamiento, actualización mediante operaciones de comparación de punto flotante.
AHP	Bit de control de precisión media alternativa 0 - Formato de precisión media IEEE (predeterminado) 1 - Formato alternativo de precisión media

Continuación de la tabla XXVIII.

DN	Bit de control de modo <i>NaN</i> (del inglés <i>Not a Number</i>) predeterminado: 0 - Los operandos NaN se propagan hasta la salida de una operación de punto flotante (predeterminado)/ Cualquier operación que involucre un NaN retorna un NaN predeterminado/
FZ	Bit de control del modo <i>Flush-to-Zero</i> : 0- <i>Flush-to-zero</i> deshabilitado 1- <i>Flush-to-zero</i> habilitado
Modo R	Campo de control del modo de redondeo; el modo de redondeo especificado es utilizado por casi todas las instrucciones de punto flotante: 00 - Modo de redondeo al más cercano (RN), predeterminado 01 – Redondeo hacia más infinito (RP) 10 - Redondea hacia menos infinito (RM) 11 - Modo de redondeo hacia cero (RZ)
IDC	Entrada desnormalizada de excepción de bit acumulados; se establece en 1 cuando se produce una excepción de punto flotante, se borra escribiendo 0 en este bit.
IXC	Bit de excepción inexacto y acumulado; se establece en 1 cuando se produce una excepción de punto flotante, se borra escribiendo 0 en este bit.
UFC	Bit de excepción acumulada de subdesbordamiento; se establece en 1 cuando se produce una excepción de punto flotante, se borra escribiendo 0 en este bit.
OFC	Bit de excepción acumulada de desbordamiento; se establece en 1 cuando se produce una excepción de punto flotante, se borra escribiendo 0 en este bit.
DZC	Bit de excepción acumulada de división entre cero; se establece en 1 cuando se produce una excepción de punto flotante, se borra escribiendo 0 en este bit.
IOC	Bit de excepción acumulada de operación no válida; se establece en 1 cuando se produce una excepción de punto flotante, se borra escribiendo 0 en este bit.

Fuente: elaboración propia, empleando Microsoft Word.

Las banderas N, Z, C y V son actualizadas mediante operaciones de comparación para punto flotante, como se muestra en la tabla XXIX.

Tabla XXIX. **Banderas N, Z, C, y V dentro de FPSCR**

Resultado de la comparación	N	Z	C	V
Igual	0	1	1	0
Menor que	1	0	0	0
Mayor que	0	0	1	0
Desordenado	0	0	1	1

Fuente: elaboración propia, empleando Microsoft Word.

Para copiar los estados de las banderas en FPSCR a el estado de las banderas en APSR (ubicadas en el núcleo) se requiere utilizar la siguiente línea de código:

VMRS APSR_nzcv, FPSCR

En la tabla XXX se presenta un listado con la mayoría de las instrucciones para punto flotante disponible en los procesadores Cortex®-M con *FPU*.

Tabla XXX. **Operaciones para datos con punto flotante**

Instrucción	Operandos	Operaciones
VABS.F32	Sd, Sm	Valor absoluto para punto flotante.
VADD.F32	Sd, Sn, Sm	Suma para punto flotante.
VCMP{E}.F32	Sd, Sm	Compara dos registros de punto flotante VCMP: generar una excepción de operación no válida si alguno de los operandos es un NaN de señalización. VCMPE: generar una excepción de operación no válida si alguno de los operandos es de cualquier tipo de NaN .
VCMP{E}.F32	Sd, #0.0	Compara un registro de punto flotante con cero (# 0.0)

Continuación de la tabla XXX.

VCVT.S32.F32	Sd, Sm	Convierte de un valor entero de 32 bits con signo a un valor con punto flotante, redondeo hacia cero.
VCVTR.S32.F32	Sd, Sm	Convierte de un valor entero de 32 bits con signo a un valor con punto flotante, redondeo basado en FPSCR.
VCVT.U32.F32	Sd, Sm	Convierte de un valor entero de 32 bits sin signo a un valor de punto flotante, redondea hacia cero.
VCVTR.U32.F32	Sd, Sm	Convierte de un valor entero de 32 bits sin signo a un valor de punto flotante, redondeo basado en FPSCR.
VCVT.F32.S32	Sd, Sm	Convierte de un punto flotante a un valor entero con signo de 32 bits.
VCVT.F32.U32	Sd, Sm	Convierte de un valor de punto flotante a un valor entero sin signo de 32 bits.
VCVT.S16.F32	Sd, #fbit	Convierte un valor de tipo <i>fixed point</i> con signo de 16 bits a un valor con punto flotante.
VCVT.U16.F32	Sd, #fbit	Convierte un valor de tipo <i>fixed point</i> sin signo de 16 bits a un valor con punto flotante, #fbit (del inglés <i>fraction bits</i>) se encuentra en un rango de 1 a 16 bit.
VCVT.F32.S32	Sd, #fbit	Convierte un valor de punto flotante a un valor de tipo <i>fixed point</i> con signo de 32 bits, #fbit en el rango de 1 a 32.
VCVT.F32.U32	Sd, #fbit	Convierte un valor de punto flotante a un valor de tipo <i>fixed point</i> sin signo de 32 bits, #fbit en el rango de 1 a 32.
VDIV.F32	{Sd,} Sn, Sm	División con punto flotante.
VFMA.F32	Sd, Sm	Punto flotante con multiplicación acumulada, $Sd=Sd+(Sn*Sm)$.
VFMS.F32	Sd, Sm	Punto flotante con multiplicación y reducción, $Sd=Sd-(Sn*Sm)$.
VFNMA.F32	Sd, Sm	Punto flotante con multiplicación y acumulación negativa $Sd=(-Sd)+(Sn*Sm)$.
VFNMS.F32	Sd, Sm	Punto flotante con multiplicación y acumulación negativas $Sd=(-Sd)-(Sn*Sm)$.

Continuación de la tabla XXX.

VLDR.32	Sd,[Rn{, #Vi}]	Carga datos de precisión simple desde la memoria (registro + offset).
VLDR.32	Sd, label	Carga datos de precisión simple desde la memoria, datos literales.
VLDR.32	Sd, [PC, #Vi]	Carga datos de precisión simple desde la memoria, datos literales.
VLDR.64	Dd,[Rn{, #Vi}]	Cargar datos de doble precisión desde la memoria (registro +offset).
VLDR.64	Dd, etiqueta	Cargar datos de doble precisión desde la memoria (datos literales).
VLDR.64	Dd, [PC, #Vi]	Cargar datos de doble precisión desde la memoria (datos literales).
VMLA.F32	Sd, Sn, Sm	Multiplicación acumulada para punto flotante, $Sd=Sd+(Sn*Sm)$.
VMLS.F32	Sd, Sn, Sm	Multiplicación con acumulación negativa para punto flotante $Sd=-Sd+(Sn*Sm)$.
VMOV{.F32}	Rt, Sm	Copia un valor de punto flotante (escalar) a los registros del núcleo ARM.
VMOV{.F32}	Sn, Rt	Copia el valor de un registro del núcleo ARM a un registro de punto flotante (escalar).
VMOV{.F32}	Sd, Sm	Copia registro de punto flotante (Sm) hacia un registro de precisión simple (Sd).
VMOV	Sm, Sm1, Rt, Rt2	Copia dos registros del núcleo ARM hacia dos registros de precisión simple.
VMOV	Rt, Rt2, Sm, Sm1	Copia dos registros de precisión simple a dos registros en el núcleo de ARM.
VMRS.F32	Rt, FPSCR	Copia el valor de FPSCR hacia Rt.
VMRS	APSR_nzcv, FPSCR	Copia el estado de las banderas de FPSCR hacia las banderas de APSR.
VMSR	FPSCR, Rt	Copia el valor de Rt hacia FPSCR.
VMOV.F32	Sd, #Vi	Mueve el valor de precisión simple hacia un registro de punto flotante.
VMUL.F32	{Sd,} Sn, Sm	Multiplicación para punto flotante
VNEG.F32	Sd, Sm	Negación para punto flotante
VNMUL	{Sd,} Sn, Sm	Multiplicación negativa para punto flotante, $Sd = -(Sn*Sm)$.
VMSR	FPSCR, Rt	Copia el valor de Rt hacia FPSCR.

Continuación de la tabla XXX.

VMOV.F32	Sd, #Vi	Mueve el valor de precisión simple hacia un registro de punto flotante.
VMUL.F32	{Sd,} Sn, Sm	Multiplicación para punto flotante
VNEG.F32	Sd, Sm	Nación para punto flotante
VNMUL	{Sd,} Sn, Sm	Multiplicación negativa para punto flotante, Sd = -(Sn*Sm).
VNMLA	Sd, Sn, Sm	Multiplicación con acumulación negada, Sd=-(Sd+(Sn*Sm)).
VNMLS	Sd, Sn, Sm	Multiplicación con acumulación negativa y resultado negado Sd=-(Sd-(Sn*Sm)).
VSQRT.F32	Sd, Sm	Raíz cuadrada para punto flotante.
VSTR.32	Sd,[Rn{, #Vi}]	Almacena datos de precisión simple en la memoria (registro+Offset).
VSTR.64	Dd,[Rn{, #Vi}]	Almacena datos de precisión doble en la memoria (registro+Offset).
VSUB.F32	{Sd,} Sn, Sm	Resta para datos con punto flotante.
Los registros del núcleo ARM están en el rango de R0-R12		

Fuente: elaboración propia, empleando Microsoft Word.

Cuando se convierte un valor de punto flotante a un valor entero se utilizan las reglas establecidas por el estándar IEEE754. En la figura 98 se muestra cómo se aproximan números de punto flotante a valores enteros.

Figura 98. **Aproximaciones basadas en el estándar IEEE 754**

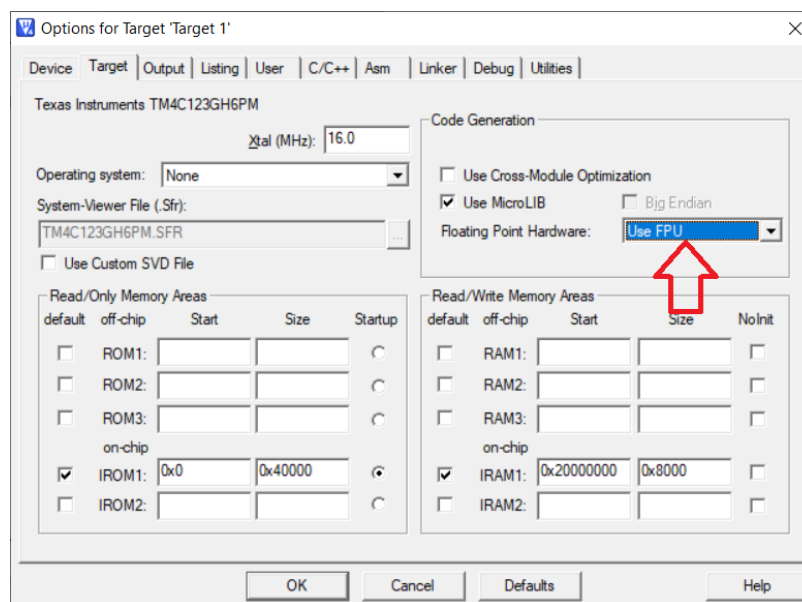
TIPO DE APROXIMACIÓN	VALORES			
	+11,5	+12,5	-11,5	-12,5
HACIA CERO	+11,0	+12,0	-11,0	-12,0
HACIA INFINITO POSITIVO	+12,0	+13,0	-11,0	-12,0
HACIA INFINITO NEGATIVO	+11,0	+12,0	-12,0	-13,0

Fuente: elaboración propia, empleando Adobe Photoshop.

3.1.12.4. Opciones de la línea de comandos del compilador

En la mayoría de los softwares para programar en lenguaje ensamblador las líneas de comando se configuran automáticamente para permitir el uso de FPU. En el caso de Keil® MDK-ARM, el IDE de uVision® establece automáticamente la opción en el compilador como “-cpu Cortex-M4.fp” para habilitar el uso de las instrucciones de punto flotante. Al momento de trabajar con datos con punto flotante se debe seleccionar dentro de Keil MDK-ARM la opción de FPU como se observa en la figura 99.

Figura 99. Uso de FPU en Keil MDK



Fuente: elaboración propia, empleando Snipping Tool.

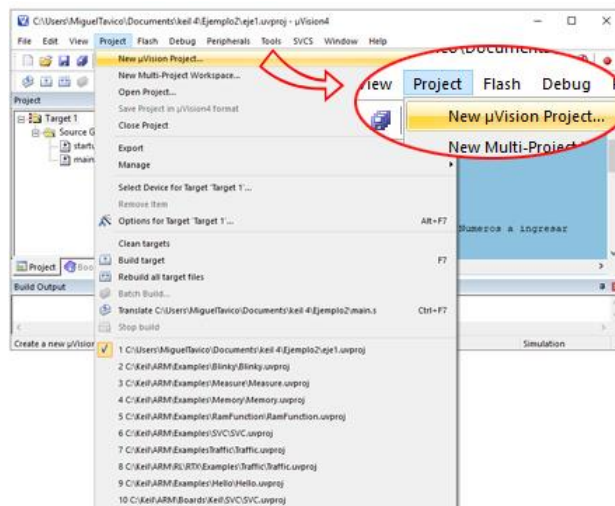
4. PROBLEMAS PROPUESTOS EN LENGUAJE ENSAMBLADOR

En el presente trabajo se estará programando el procesador Cortex®-M4F con la ayuda del entorno de desarrollo Keil® uVision.

4.1. Crear un proyecto nuevo en Keil® uVision

Se presenta la correcta forma de crear un proyecto nuevo en Keil®, luego de ejecutar el programa se debe de ir a la barra de opciones y seleccionar *Project* seguidamente seleccionar *New uVision Project*, como se muestra en la figura 100.

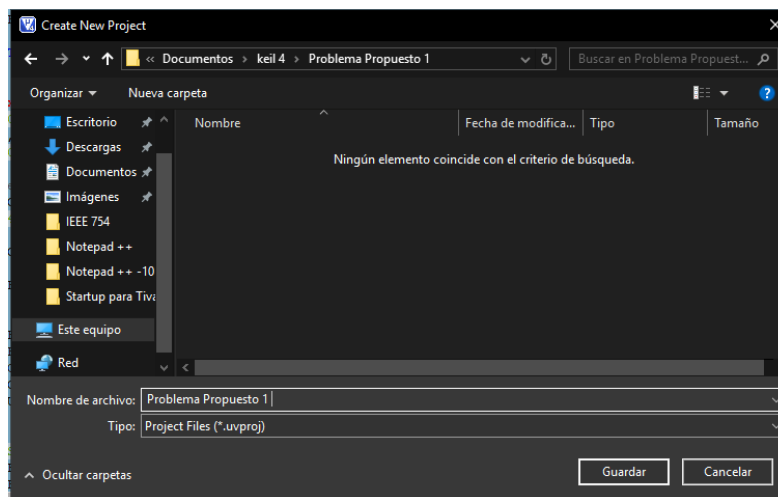
Figura 100. Nuevo Proyecto



Fuente: elaboración propia, empleando Snipping Tool.

Seguidamente se abrirá una ventana en donde se tiene que crear una carpeta en la cual se almacenarán todos los archivos del proyecto, también se debe asignar un nombre al archivo como se muestra en la figura 101 y dar clic en el botón guardar.

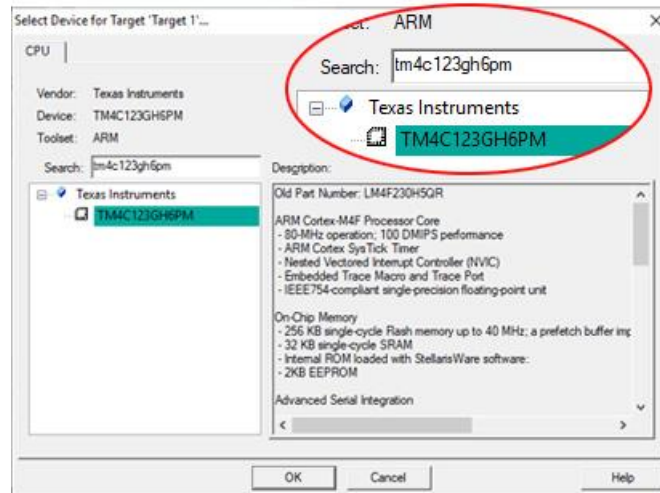
Figura 101. Selección del nombre de archivo



Fuente: elaboración propia, empleando Snipping Tool.

Luego de asignarle el nombre al proyecto se desplegará una ventana en donde se debe elegir la tarjeta de desarrollo que estemos usando en este caso se trabajará con Tiva™ C Series TM4C123GH6PM, por lo que se buscará dicho modelo, ver figura 102.

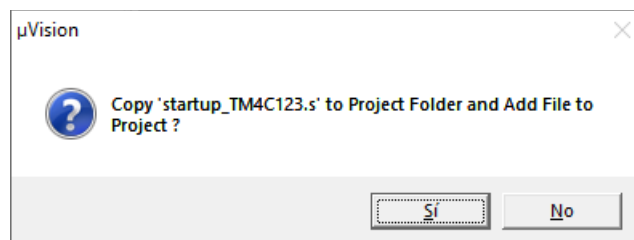
Figura 102. Selección de CPU



Fuente: elaboración propia, empleando Snipping Tool.

Posteriormente se desplegará un mensaje para copiar el archivo "startup_TM4C123.S" hacia el proyecto, se le dará al botón "Sí" como se muestra en la figura 103.

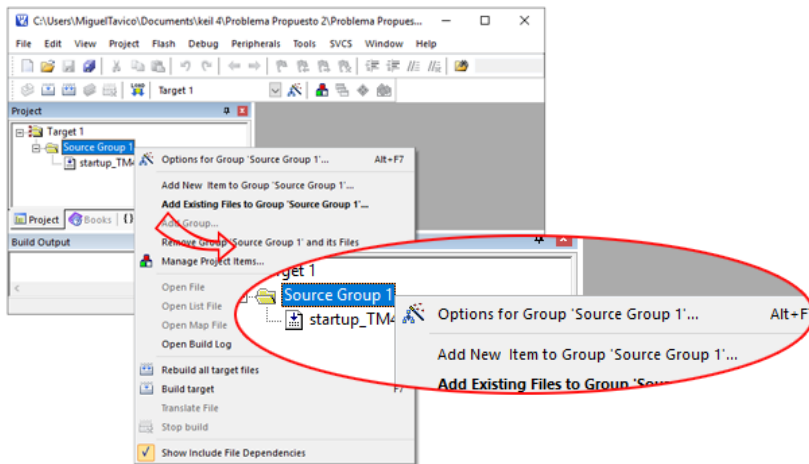
Figura 103. Copiar archivo Startup



Fuente: elaboración propia, empleando Snipping Tool.

Ahora solo falta agregar un ítem al archivo, se dará clic derecho sobre la carpeta llamada “*Source Group 1*” y se debe seleccionar “*Add New Item to Group Source Group 1*” como se muestra en la figura 104.

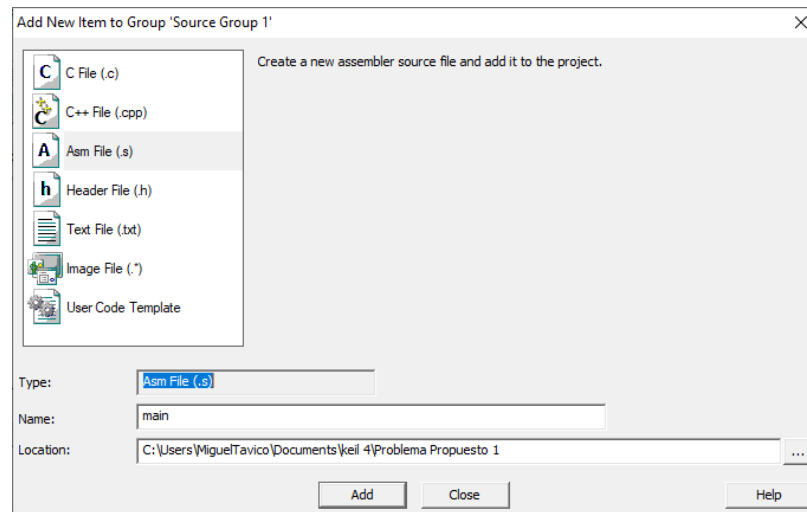
Figura 104. Nuevo ítem



Fuente: elaboración propia, empleando Snipping Tool.

Ahora se mostrará una ventana en donde se asignará un nombre al archivo en donde se colocará todas las líneas de código, para este y los demás problemas se seleccionará el tipo de archivo “*Asm File (.s)*” y se usará en nombre de “*main*”, ver figura 105.

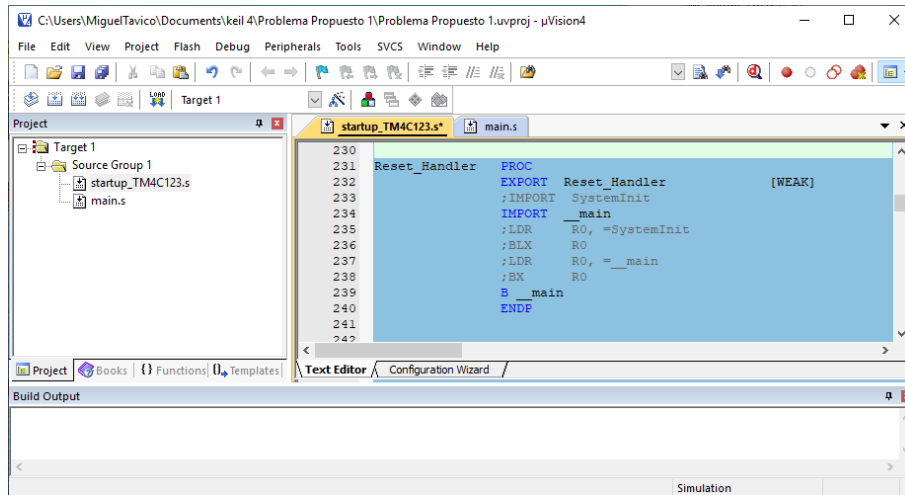
Figura 105. Nuevo ítem al grupo



Fuente: elaboración propia, empleando Snipping Tool.

Finalmente, para que todas las líneas escritas en lenguaje ensamblador se ejecuten correctamente se debe editar el archivo "startup_TM4C123.s", exactamente las líneas en el rango de 230 a 240 como se muestra en la figura 106.

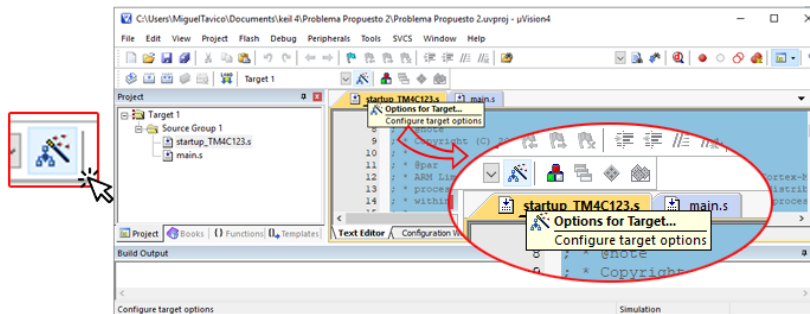
Figura 106. **Archivo Startup**



Fuente: elaboración propia, empleando Snipping Tool.

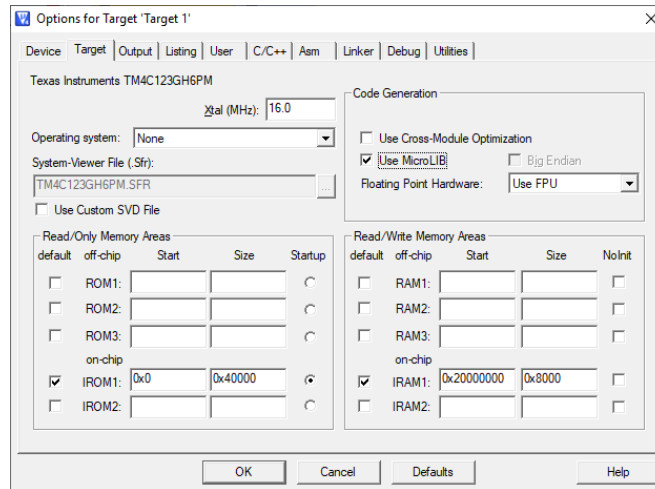
Se debe activar la opción de “Use MicroLIB” esto entrando a las opciones de “Target” como se muestra en las figuras 107 y 108, si solamente se usará el simulador de la tarjeta de desarrollo se debe seleccionar la opción “Use Simulator” como se muestra en la figura 109.

Figura 107. **Opción llamada Target**



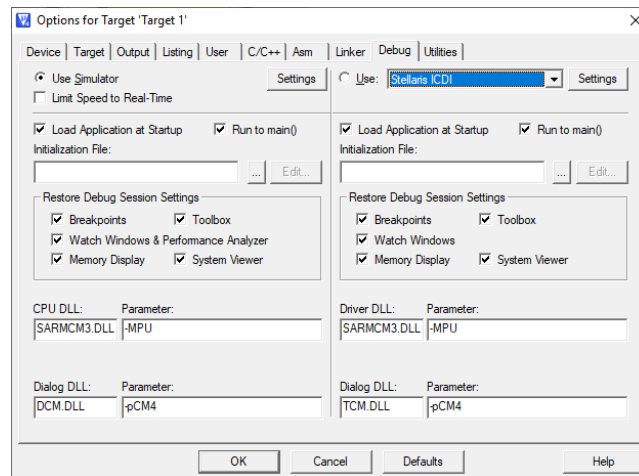
Fuente: elaboración propia, empleando Snipping Tool.

Figura 108. Selección de *Use Micro Lib*



Fuente: elaboración propia, empleando Snipping Tool.

Figura 109. Selección de *Use Simulator*

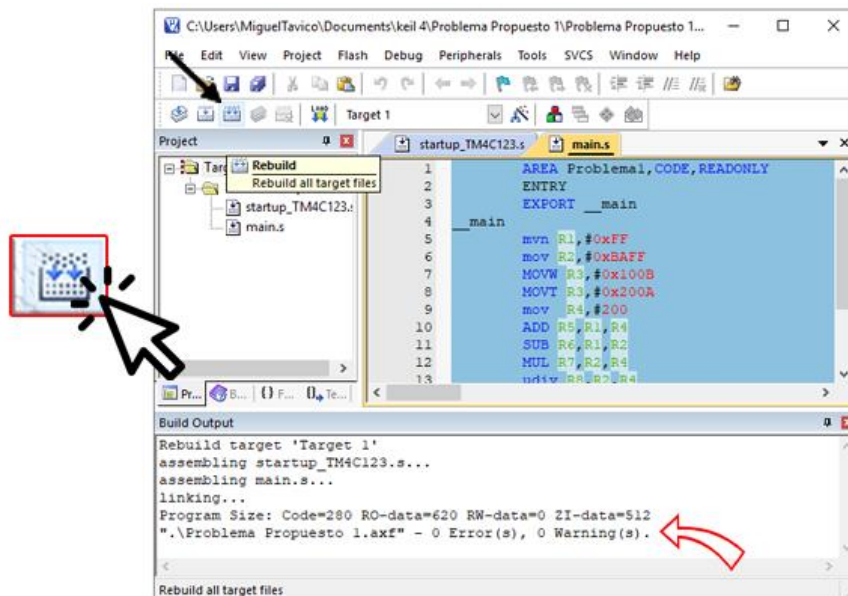


Fuente: elaboración propia, empleando Snipping Tool.

4.2. Constructor y Depurador

Al finalizar de escribir las líneas de código dar clic en el botón “*Rebuild*” como se muestra en la figura 110, esto con la finalidad de construir el programa y depurar errores.

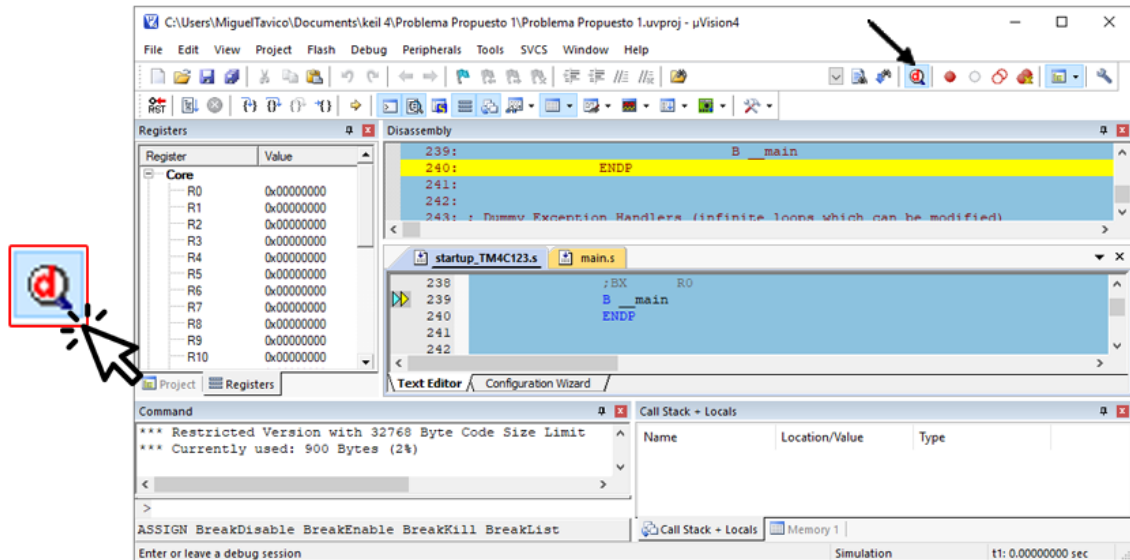
Figura 110. *Rebuild*



Fuente: elaboración propia, empleando Snipping Tool.

Para correr el programa, dar clic en el botón “*Start/Stop Debug Session*”, como se muestra en la figura 111, para hacer que las instrucciones se ejecuten se debe presionar la tecla F11.

Figura 111. **Debug**



Fuente: elaboración propia, empleando Snipping Tool.

4.3. Problema propuesto 1

Dados los siguientes valores, registros de propósito general y operaciones matemáticas, almacene los datos como se presentan en la tabla XXXI y realice las operaciones matemáticas presentes en la tabla XXXII.

Tabla XXXI. **Valores para operar**

Registros	Valores
R1	~0xFF
R2	0xBAFF
R3	0X200A100B
R4	200

Fuente: elaboración propia, empleando Microsoft Word.

Tabla XXXII. **Operaciones matemáticas**

Registros	Operaciones
R5	$R1+R4$
R6	$R1-R2$
R7	$R2*R4$
R8	$R2\div 24$

Fuente: elaboración propia, empleando Microsoft Word.

4.3.1. **Análisis del problema**

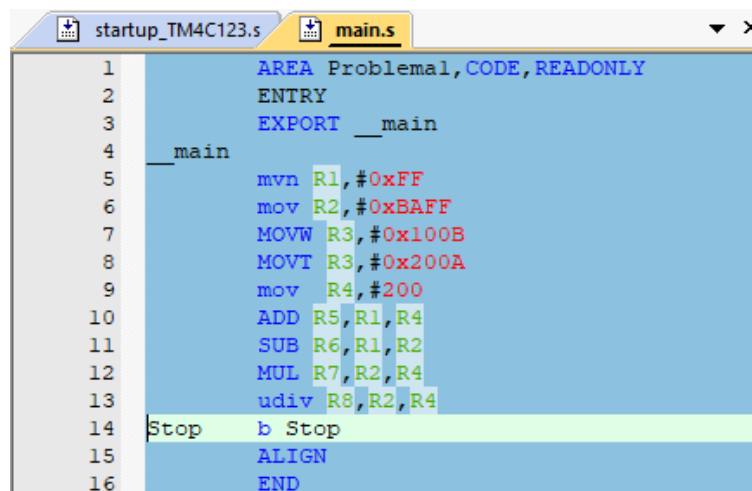
Para mover directamente valores a los registros se usará la instrucción MOV y sus variantes, en el caso de mover el valor negativo de 0XFF se usará la variante MVN. Para mover el valor 0xBAFF se usará MOV, esta instrucción puede trabajar con datos en el rango de 0x0 a 0xFFFF.

La instrucción MOV puede trabajar con valores con un tamaño de hasta 16 bits, si se intenta mover un valor con un tamaño mayor de 16 bit en un solo paso dará error. Por lo anterior, se requiere el uso de dos pasos para mover valores con tamaño mayor de 16 bits, esto se realiza con la instrucción MOVW y MOVT. MOVW mueve un valor con tamaño de 16 bits hacia un registro de 32 bits tomando en cuenta solo los primeros 16 bits. La instrucción MOVT mueve un valor de 16 bits a un registro de 32 bits tomando en cuenta los últimos 16 bits. Para la suma, resta y división se usará las instrucciones ADD, SUB, MUL y UDIV respectivamente.

4.3.2. Solución del problema 1

La figura 112 presenta el código ideal para dar solución al problema propuesto número 1.

Figura 112. Código fuente del problema 1



```
1      AREA Problemal, CODE, READONLY
2      ENTRY
3      EXPORT __main
4      __main
5      mvn R1, #0xFF
6      mov R2, #0xBAFF
7      MOVW R3, #0x100B
8      MOVT R3, #0x200A
9      mov R4, #200
10     ADD R5, R1, R4
11     SUB R6, R1, R2
12     MUL R7, R2, R4
13     udiv R8, R2, R4
14     Stop b Stop
15     ALIGN
16     END
```

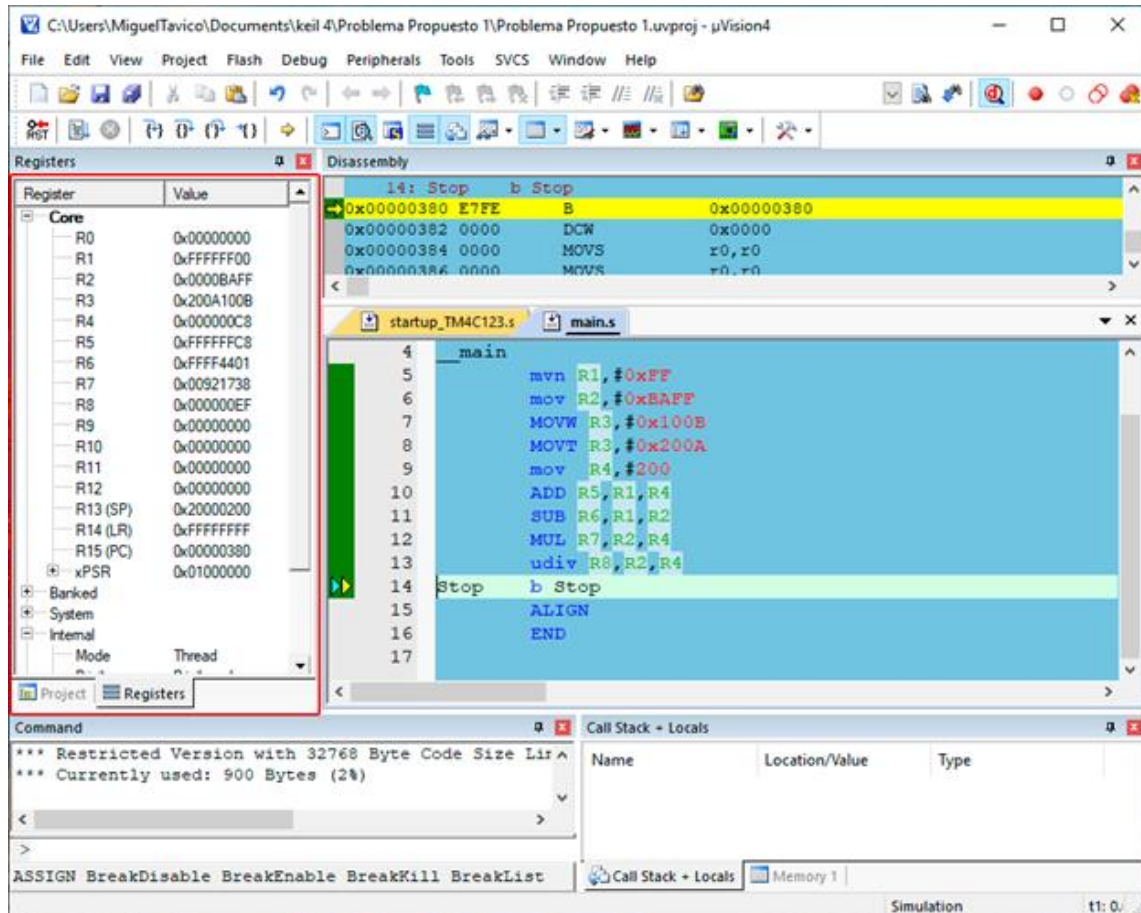
Fuente: elaboración propia, empleando Snipping Tool.

4.3.3. Análisis de la solución

Las líneas de código mostradas en la figura 112 se ejecutan de forma secuencial y se cargan exitosamente los valores mostrados en la tabla XXXI. Posteriormente se realizan las operaciones matemáticas presentes en la tabla XXXII.

En la figura 113 en la ventana “Registers” se ven correctamente almacenados los datos propuestos en los registros de propósito general, así como el resultado de las operaciones matemáticas.

Figura 113. Resultados del problema 1



Fuente: elaboración propia, empleando Snipping Tool.

4.4. Problema propuesto 2

En la tabla XXXIII se presentan dos direcciones en la memoria, guarde en ellas dos valores con tamaño de 32 bits utilizando las instrucciones LDR y MOV. Luego de almacenar los datos en memoria, recupere uno de ellos y guárdelo en un registro de propósito general.

Tabla XXXIII. Dirección en memoria

Primera dirección en memoria	0x20000000
Segunda dirección en memoria	0x2000000C

Fuente: elaboración propia, empleando Microsoft Word.

4.4.1. Análisis del problema

Lo primero que hay que hacer es definir las direcciones de la memoria como constante con ayuda de la directiva EQU, seguidamente almacenar los valores de 32 bits (seleccionados por el usuario) en los registros de propósito general. Con las instrucciones STR se procede a almacenar los dos valores en la memoria. Finalmente, con la instrucción LDR se recupera uno de los dos valores hacia un registro.

4.4.2. Solución del problema 2

La figura 114 presenta la solución del problema propuesto número 2.

Figura 114. Código fuente del problema 2

```

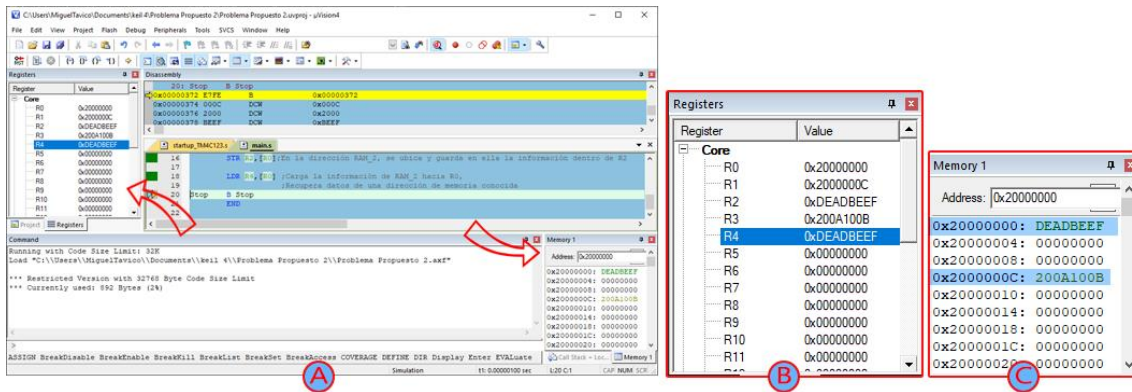
RAM_1 EQU 0x20000000
RAM_2 EQU 0x2000000C
AREA Problema2, CODE, READONLY
ENTRY
EXPORT __main
__main
;Se carga direcciones conocidas de memoria en R0 y R1
LDR R0,=RAM_1
LDR R1,=RAM_2
;Se carga valores en R2 y R3
LDR R2,=0xDEADBEEF ; LDR para valores mayores a 0xFFFF
MOVW R3,#0x100B
MOVT R3,#0x200A
STR R3,[R1];En la dirección RAM_1, se ubica y guarda en ella la información dentro de R3
STR R2,[R0];En la dirección RAM_2, se ubica y guarda en ella la información dentro de R2
LDR R4,[R0] ;Carga la información de RAM_2 hacia R0,Recupera datos de una dirección de memoria conocida
B Stop
END
    
```

Fuente: elaboración propia, empleando Snipping Tool.

4.4.3. Análisis de la solución

Como se muestra en la figura 114 los registros R0 y R1 son utilizados para almacenar las direcciones de memoria. Para cargar valores de 32 bits en los registros se utiliza LDR y la variante de MOV (MOVT y MOVW). Finalmente, con LDR se recupera en R4 el valor almacenado en la dirección de memoria igual a 0x20000000.

Figura 115. Datos en memoria



Fuente: elaboración propia, empleando Snipping Tool.

En la figura 115A se muestran los valores cargados en los registros de propósito general y en la ventana “Memory 1” (figura 115, C) se confirma el correcto almacenamiento de los datos. El valor almacenado en la memoria con dirección igual a 0x20000000 se almacena finalmente en R4 como se observa en la ventana “Registers” (figura 115, B).

4.5. Problema propuesto 3

Almacene en 5 direcciones de memoria consecutivas el valor mostrado en la tabla XXXIV.

Tabla XXXIV. **Dirección y valor**

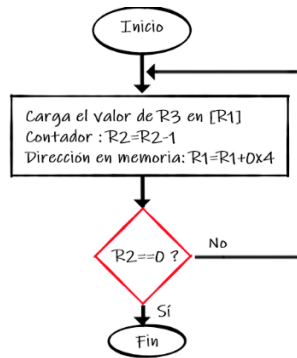
Dirección inicial en la memoria	0x20000000
Valor para almacenar	0xDEADBABE

Fuente: elaboración propia, empleando Microsoft Word.

4.5.1. Análisis del problema

Para resolver el problema planteado es necesario el uso de un contador, dicho contador irá decrementando desde cinco hasta llegar a cero. Al inicio se almacenará el valor de 0XDEADBABE en la memoria con dirección igual a 0x20000000. Como se muestra en la figura 116 a la dirección de memoria (almacenada en R1) se le sumará 0x4 (4 bytes) y al valor del contador se le restará 1 unidad. Luego se procederá a verificar el valor del contador, si el valor del contador es igual a cero se terminará el programa de lo contrario se repetirán los pasos anteriores.

Figura 116. Diagrama de flujo del problema 3



Fuente: elaboración propia, empleando Adobe Photoshop.

4.5.2. Solución de problema 3

La figura 117 presenta la solución del problema propuesto número 3.

Figura 117. Código fuente del problema 3

```
RAM1_ EQU 0x20000000
AREA Problema3, CODE, READONLY
ENTRY
EXPORT __main
__main
BL Almacena_Datos
Stop B Stop
Almacena_Datos
LDR R1,=RAM1_ ;Mueve el valor 0x20000000 al registro R1
MOV R2,#5 ;Inicia el contador en R2=5
LDR R3,=0xDEADBABE
LOOP1
STR R3,[R1]
ADD R1,R1,#0x4 ;Aumenta la dirección de memoria 0x04
SUBS R2,R2,#1 ;R2=R2-1, Valor inicial de R2 = 5
BNE LOOP1 ;Cuando Z==1, se tiene un resultado igual a cero
BX LR
ALIGN
END
```

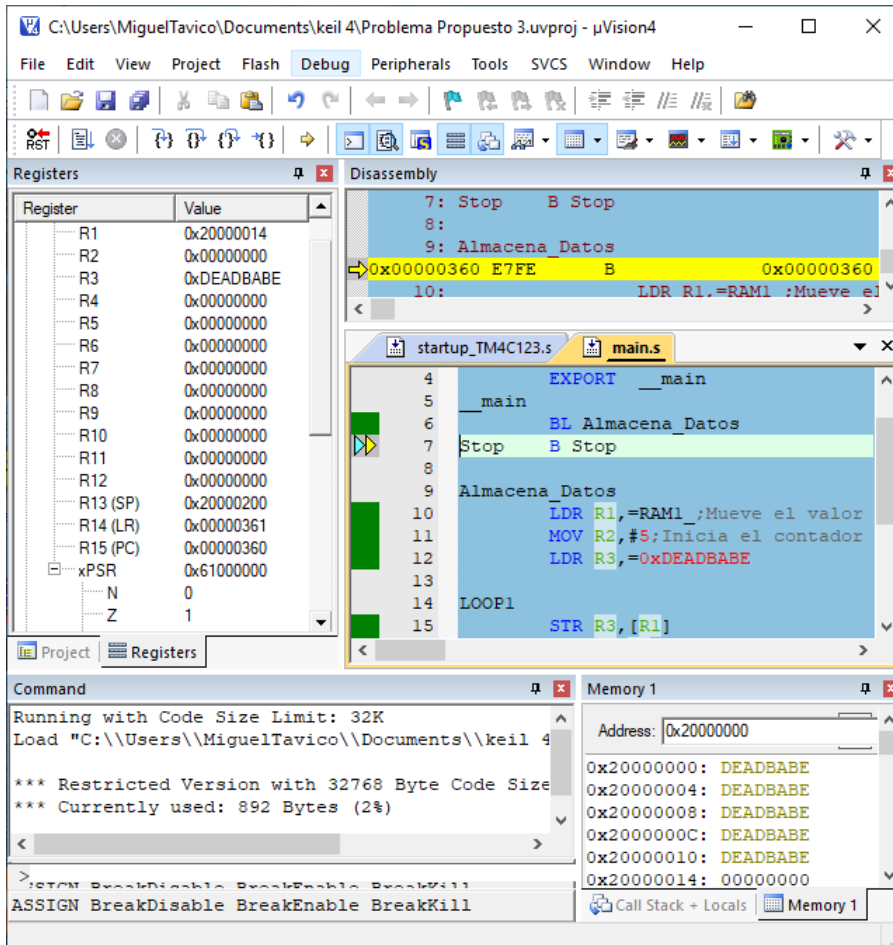
Fuente: elaboración propia, empleando Snipping Tool.

4.5.3. Análisis de la solución

Como se muestra en la figura 117 el valor inicial de la dirección de memoria se almacena en R1, el contador (R2) se inicia con un valor de 5 y el valor 0XDEADBABE es almacenado en R3. Seguido de la etiqueta LOOP1 se encuentra el código correspondiente para poder incrementar el valor de memoria y reducir el valor de R3 en cada iteración.

La instrucción BNE es de suma importancia ya que en cada iteración comprueba si luego de realizar la operación denotada por SUBS el resultado es cero (bandera Z=1), si el resultado no es cero salta a la etiqueta LOOP1 de lo contrario salta a la etiqueta posterior a BL (en este caso salta a la etiqueta Stop).

Figura 118. Resultado del problema 3



Fuente: elaboración propia, empleando Snipping Tool.

En la figura 118 se puede corroborar el correcto almacenamiento en memoria, 0xDEADBABE ha sido almacenado 5 veces en direcciones consecutivas de memoria. Así mismo el registro R2, el cual fue utilizado como contador tiene un valor final de cero y en el registro xPSR la bandera Z tiene un valor igual a 1 (lo que indica que una operación previa dio resultado igual a cero).

4.6. Problema Propuesto 4

En los registros R0 y R1 almacene los valores 0xDEADBEEF y 0xBABEFACE respectivamente, como se muestra en la tabla XXXV. Utilice únicamente operaciones lógicas para que los datos entre R0 y R1 se intercambien sin usar registros extras.

Tabla XXXV. **Valores propuestos**

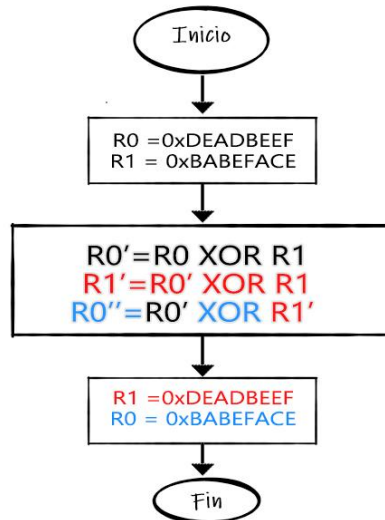
R0	0xDEADBEEF
R1	0xBABEFACE

Fuente: elaboración propia, empleando Microsoft Word.

4.6.1. Análisis del problema

Utilizando la compuerta lógica XOR se puede realizar el intercambio de datos entre dos registros en pocos pasos de forma secuencial, como se muestra en la figura 119.

Figura 119. Diagrama de flujo del problema 4



Fuente: elaboración propia, empleando Adobe Photoshop.

4.6.2. Solución de problema 4

La figura 120 presenta el código ideal para dar solución al problema propuesto número 4.

Figura 120. Código fuente del problema 4

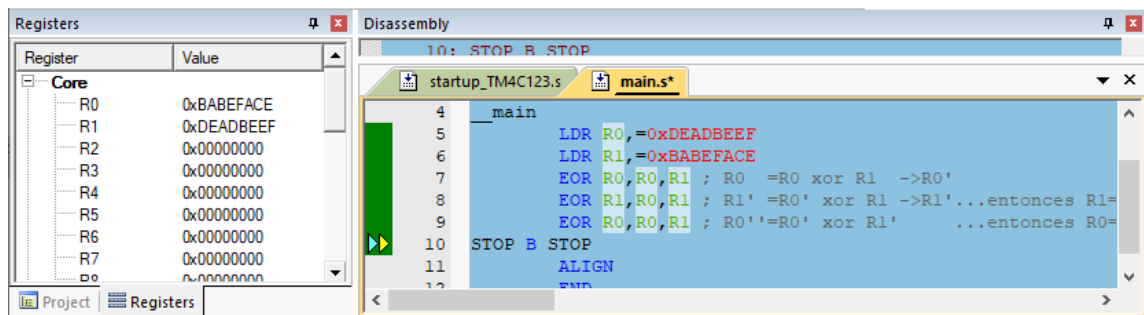
```
AREA Problema4, CODE, READONLY
ENTRY
EXPORT __main
__main
LDR R0, =0xDEADBEEF
LDR R1, =0xBABEFACE
EOR R0, R0, R1 ; R0 =R0 xor R1 ->R0'
EOR R1, R0, R1 ; R1' =R0' xor R1 ->R1'...entonces R1=DEADBEEF
EOR R0, R0, R1 ; R0''=R0' xor R1' ...entonces R0=BABEFACE
STOP B STOP
ALIGN
END
```

Fuente: elaboración propia, empleando Snipping Tool.

4.6.3. Análisis de la solución

El uso correcto de la compuerta lógica XOR nos permite hacer el intercambio de información entre dos registros, el código presente en la figura 120 es el idóneo para realizar el intercambio de una forma eficiente.

Figura 121. Resultado del problema 4



Fuente: elaboración propia, empleando Snipping Tool.

En la figura 121 se comprueba la correcta implementación del código presente de la figura 120. En la ventana *Registers* se observa que el valor 0xBABEFACE está en R0 y 0xDEADBEEF está en R1.

4.7. Problema Propuesto 5

Con base en la ecuación de la figura 122 y sabiendo que el valor de $Q=2$, $R=4$ y $S=42333222$ resuelva para encontrar el valor de P , tomar en cuenta que los valores de Q , R y S deben de estar almacenados en memoria.

Figura 122. **Ecuación propuesta del problema 5**

$$P = Q + S + R$$

Fuente: elaboración propia, empleando Adobe Photoshop.

4.7.1. **Análisis del problema**

En el presente problema se tendrán dos áreas distintas, la primera será de código y la segunda será de datos en memoria. Con ayuda de la instrucción ADRL se obtendrá y guardará en R1 la dirección en memoria en donde se encuentre el valor de Q. Para obtener el valor de S se necesita ubicar la posición almacenada en R1 y sumarle 4 bytes, para ubicar el valor de R se necesitará sumarle a R1 8 bytes.

4.7.2. **Solución de problema 5**

La figura 123 presenta la solución del problema propuesto número 5.

Figura 123. **Código fuente del problema 5**

```
AREA Problema5, CODE, READONLY
ENTRY
EXPORT __main
__main
ADRL R4,Valores
;R4 apunta a el espacio de memoria de 4 Bytes (llamado Valores)
LDR R1,[R4]; Sin Offset
LDR R2,[R4,#4]; Offset 0x4
LDR R3,[R4,#8]; Offset 0x8

ADD R0,R1,R2 ; -> R0=2+4
ADD R0,R0,R3 ; -> R0'=R0+42333222= 42,333,481
Stop
B Stop
AREA Problema5, DATA, READONLY
Valores DCD 42333222 ;->0x285F426 ; WORD "4 Bytes"
DCD 4 ;->0x4
DCD 255 ;->0xFF
ALIGN
END
```

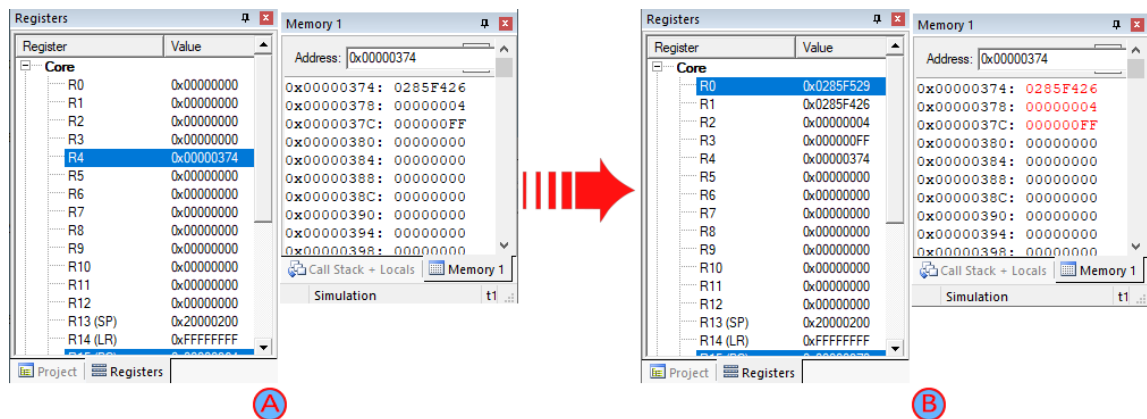
Fuente: elaboración propia, empleando Snipping Tool.

4.7.3. Análisis de la solución

Como se observa en la figura 123 se tiene dos áreas, en el área llamada DATA es en donde se almacenan los valores de Q, S y R en memoria, debemos tomar en cuenta que para almacenar los datos se utilizó la directiva DCD. La directiva DCD almacena datos en memoria con tamaño de una palabra (Word, 4 bytes) y los alinea al límite de 4 bytes.

En el área llamada CODE los valores de Q, S y R son recuperados y almacenados en los registros R1, R2, y R3 respectivamente. Finalmente, los valores recuperados se suman, con el fin de encontrar el valor total de P.

Figura 124. Resultados del problema 5



Fuente: elaboración propia, empleando Snipping Tool.

La figura 124 muestra los valores de S, Q y R almacenados en memoria, el primer valor fue almacenado en la dirección 0x00000374. En la figura 124B muestra los valores recuperados de la memoria y almacenados en R1, R2 y R3. La suma algebraica de R1, R2 y R3 es almacenada en R0. El resultado final es 0x0285F529 (42 333,481 en sistema decimal).

4.8. Problema Propuesto 6

Con base en la ecuación de la figura 125 y sabiendo que el valor de $Q=2$, $R=4$ y $S=255$ resuelva para encontrar el valor de P , tomar en cuenta que los valores de Q , R y S deben de estar almacenados en memoria con un tamaño de 1 byte.

Figura 125. **Ecuación propuesta del problema 6**

$$P = Q + S + R$$

Fuente: elaboración propia, empleando Adobe Photoshop.

4.8.1. Análisis del problema

Al observar los valores propuesto nos damos cuenta de que cada uno se pueden almacenar perfectamente en 1 Byte, por tal razón para almacenar los valores en memoria se usará la directiva DCB y al recuperarlos de la memoria se utilizará LDRB (instrucción para cargar 1 byte a la memoria). Lo anterior con la finalidad de optimizar el uso de la memoria.

4.8.2. Solución del problema 6

La figura 126 presenta el código ideal para dar solución al problema propuesto número 6.

Figura 126. Código fuente del problema 6

```
;Q=2, R=4, S=255
AREA Problema6, CODE, READONLY
ENTRY
EXPORT __main
main
ADRL R4,MiniValores ;R4 apunta a el espacio de memoria de 1 byte llamado MiniValores
LDRB R1,[R4] ;Sin Offset
LDRB R2,[R4,#1]:Offset 0x1 Byte
LDRB R3,[R4,#2]:Offset 0x2 Byte

ADD R0,R1,R2 ; -> R0=2+4
ADD R0,R0,R3 ; -> R0'=R0+255
Stop
B Stop
AREA Problema6, DATA, READWRITE
MiniValores DCB 2 ;->0x2
DCB 4 ;->0x4
DCB 255 ;->0xFF
;DCB -> Almacena en 1 Byte (CHAR) 0-255
ALIGN
END
```

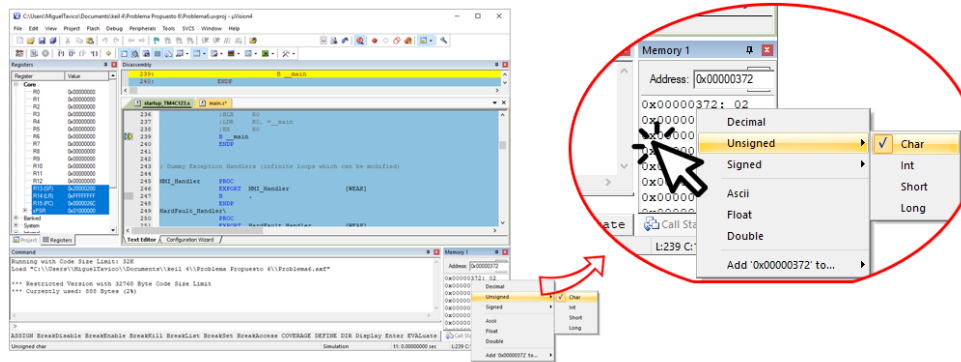
Fuente: elaboración propia, empleando Snipping Tool.

4.8.3. Análisis de la solución

Como se observa en la figura 126 los valores 2, 4 y 255 fueron almacenados en memoria con la ayuda de la directiva DCB, de esta forma los valores se almacenan en un byte así mismo se alinean en los límites de un byte. Para recuperar los valores almacenados en memoria se requiere de la instrucción LDRB, así el incremento en la dirección de memoria será de tan solo 1 byte como se muestra en la figura 126

Al momento de entrar al modo *Debug* es necesario ajustar la memoria a los límites de 1 byte de esta forma será más fácil visualizar los valores almacenados en memoria. Se debe de dar clic derecho sobre la ventana llamada *Memory 1* y seguidamente seleccionar *Char* como se muestra en la figura 127

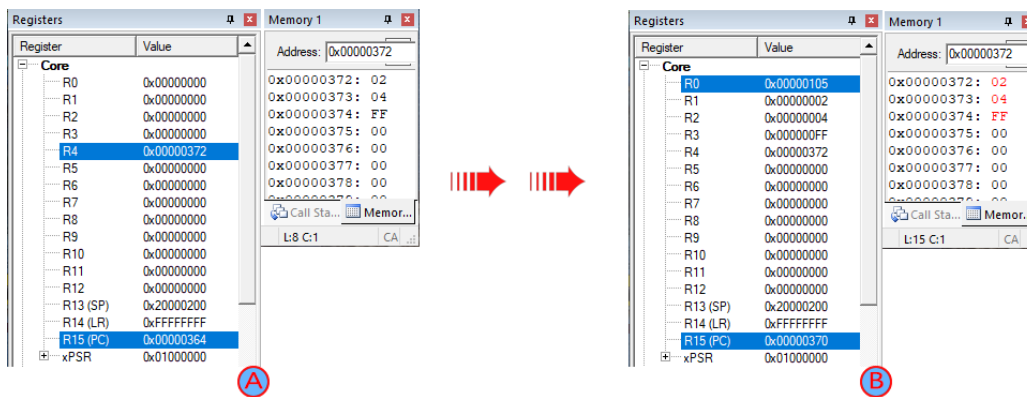
Figura 127. Cambio de vista de la memoria



Fuente: elaboración propia, empleando Snipping Tool.

Como se muestra en la figura 128A el primer valor Q es almacenado en la dirección de memoria igual a 0x00000372 a un límite de 1 byte. En la figura 128B se observa el correcto almacenamiento de los valores Q, S Y R así mismo el resultado final del registro R0. El valor final del registro R0 contiene la suma algebraica de Q, S y R la cual es igual a 0x00000105 (261 en sistema decimal).

Figura 128. Resultados del problema 6



Fuente: elaboración propia, empleando Snipping Tool.

4.9. Problema Propuesto 7

Demuestre la principal diferencia entre el uso de la instrucción SUB y SUBS mediante operaciones matemáticas, así mismo explique el uso de la instrucción ADC. Se requiere la implementación de los datos de la tabla XXXVI

Tabla XXXVI. **Registros y valores del problema 7**

Registros	Valores
R1	0xA
R2	0x5
R9	0

Fuente: elaboración propia, empleando Microsoft Word.

4.9.1. Análisis del problema

Debe de recordar que al momento de agregarle el sufijo “S” a la instrucción SUBS se actualizará automáticamente el registro xPSR (el cual contiene las banderas Z, N, C y V). Para visualizar la diferencia SUB y SUBS debemos de realizar operaciones matemáticas, el resultado de estas cambia el estado de las banderas dentro de xPSR como se muestra en la tabla XXXVII.

Tabla XXXVII. **Banderas del registro xPSR**

Bandera	Resultado
Z=1	Es cero
Z=0	Es distinto a cero
C=1	Se produjo un acarreo
C=0	No se produjo acarreo
N=1	Es un valor negativo
N=0	No es un valor negativo
V=1	Ocurrió un desbordamiento de memoria
V=0	No se tiene desbordamiento de memoria

Fuente: elaboración propia, empleando Microsoft Word.

4.9.2. Solución del problema 7

La figura 129 presenta el código ideal para dar solución al problema propuesto número 7.

Figura 129. **Código fuente del problema 7**

```

1  AREA Problema7, CODE, READONLY
2  ENTRY
3  EXPORT __main
4  __main
5      MOV R1, #0xA
6      MOV R2, #0x5
7      MOV R9, #0
8      SUB R3, R1, R2; R3= 10 - 5 = 5
9      SUB R3, R3, R2; R3= 5 - 5 = 0
10
11     SUBS R4, R1, R2 ; R4= 10-5 ,La operación produce un acarreo
12     ADC R9, R9, #0 ; R9=R9+0+ Carry
13     SUBS R4, R4, R2 ; R4= 5-5 0, La operación produce un resultado ==0, acarreo C=1
14     ADC R9, R9, #0 ; R9=R9+0+ Carry
15
16     SUBS R5, R1, #11 ; R5 = 10-11=-1 La operacion da como resultado un valor negativo
17     ADDS R6, R1, R2 ; R6=10+5 Sin acarreo ,valor positivo !=0
18 STOP B STOP
19 END
20 ALIGN

```

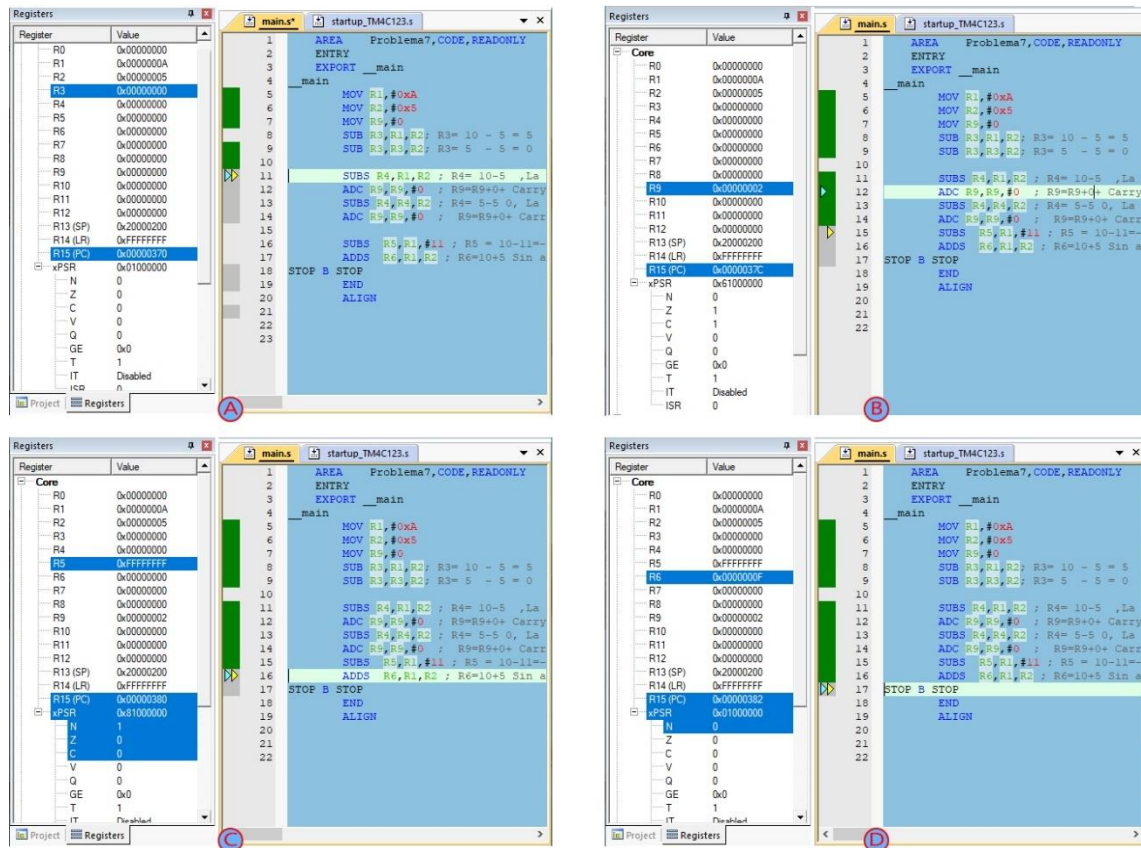
Fuente: elaboración propia, empleando Snipping Tool.

4.9.3. Análisis de la solución

Las líneas de código 8 y 9 de la figura 129 realizan restas mediante la instrucción SUB, la cual no actualiza el registro xPSR. Las líneas de código posteriores a la línea 10 utilizan SUBS la cual permite actualizar las banderas del registro xPSR así mismo la instrucción ADC lleva la suma de los acarreos producidos en las restas.

Como se muestra en la figura 130 A a pesar de tener resultado igual a cero y haber tenido acarreo las banderas del registro xPSR no cambiaron de estado. Por el otro lado, en la figura 130 B se observa un cambio en el estado de las banderas Z y C del registro xPSR esto debido a la implementación de SUBS. Así mismo al utilizar la instrucción ADC, en el registro R9 se tiene un valor de 2 por lo que se infiere que se realizó dos operaciones que tuvieron acarreo.

Figura 130. Resultado del problema 7



Fuente: elaboración propia, empleando Snipping Tool.

En la figura 130 C se muestra que la bandera N = 0 esto debido a que la operación ejecutada en la línea 15 dio como resultado un valor negativo igual a -1. Finalmente, en la figura 130 D todas las banderas del registro xPSR regresan a cero esto debido a que la instrucción ADDS (en la línea 16 de la figura 130 D) no realizó acarreo, dio resultado negativo, resultado igual a cero o se tuvo un desbordamiento en memoria.

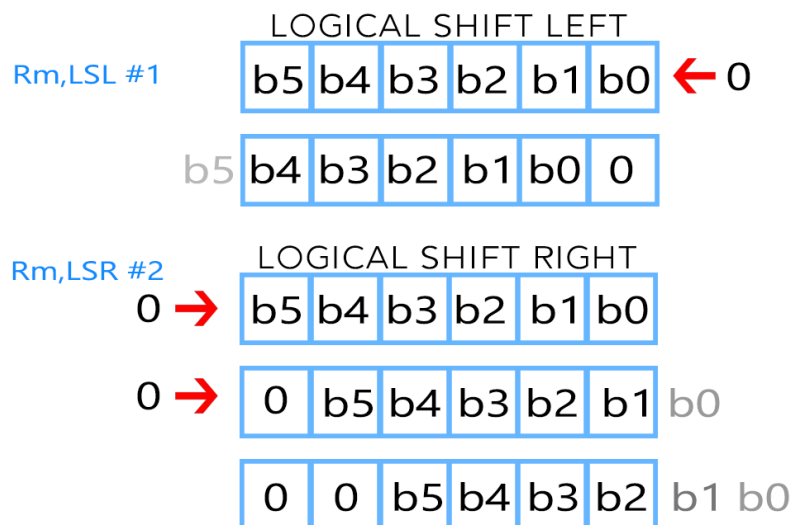
4.10. Problema Propuesto 8

Demuestre el uso correcto de la instrucción LSL y LSR mediante la multiplicación y división indirecta de valores almacenados en registros de propósito general.

4.10.1. Análisis del problema

Las instrucciones *Logical Shift Left* y *Logical Shift Right* permiten un corrimiento en los bits de un valor almacenado como se muestra en la figura 131. El valor inmediato seguido de la instrucción “LSL” o “LSR” denota cuantas posiciones el valor en bits será desplazado.

Figura 131. LSL y LSR



Fuente: elaboración propia, empleando Adobe Photoshop.

Cuando los bits son desplazados hacia la izquierda se utiliza la instrucción LSL e indirectamente se obtiene una multiplicación del número almacenado (denotado por m en sistema decimal) y las posiciones desplazadas (denotada por n) como se observa en la figura 132

Figura 132. **Multiplicación indirecta con LSL**

$$m * 2^n$$

Fuente: elaboración propia, empleando Adobe Photoshop.

Cuando los bits son desplazados hacia la derecha se utiliza la instrucción LSR e indirectamente se obtiene una división del número almacenado (denotado por m en sistema decimal) y las posiciones desplazadas (denotada por n) como se observa en la figura 133.

Figura 133. **División indirecta con LSR**

$$\frac{m}{2^n}$$

Fuente: elaboración propia, empleando Adobe Photoshop.

4.10.2. Solución de problema 8

La figura 134 presenta el código ideal para dar solución al problema propuesto número 8.

Figura 134. Código fuente del problema 8

```
1      AREA      Problema8, CODE, READONLY
2      ENTRY
3      EXPORT   __main
4      main
5      MOV R0, #0x10 ; binario      1|0|0|0|0|0|      <--0
6      LSL R1, R0, #1 ;              1|0|0|0|0|0|0|      <--0 10x2^1
7      LSL R2, R0, #2 ;              1|0|0|0|0|0|0|0|      <--0 10x2^2
8      LSL R3, R0, #3 ;              1|0|0|0|0|0|0|0|0|      <--0 10x2^3
9      LSL R4, R0, #4 ;              1|0|0|0|0|0|0|0|0|0|      <--0 10x2^4
10
11     MOV R5, #0x100; 0-->          1|0|0|0|0|0|0|0|0|0|0|      , 0x100
12     LSR R6, R5, #2 ; 0-->0-->    0|0|1|0|0|0|0|0|0|0|      , 0x40
13     LSR R7, R5, #3 ; 0-->        0|0|0|1|0|0|0|0|0|0|      , 0x20
14     LSR R8, R5, #4 ; 0-->        0|0|0|0|0|1|0|0|0|0|0|      , 0x10
15     STOP B STOP
16     END
```

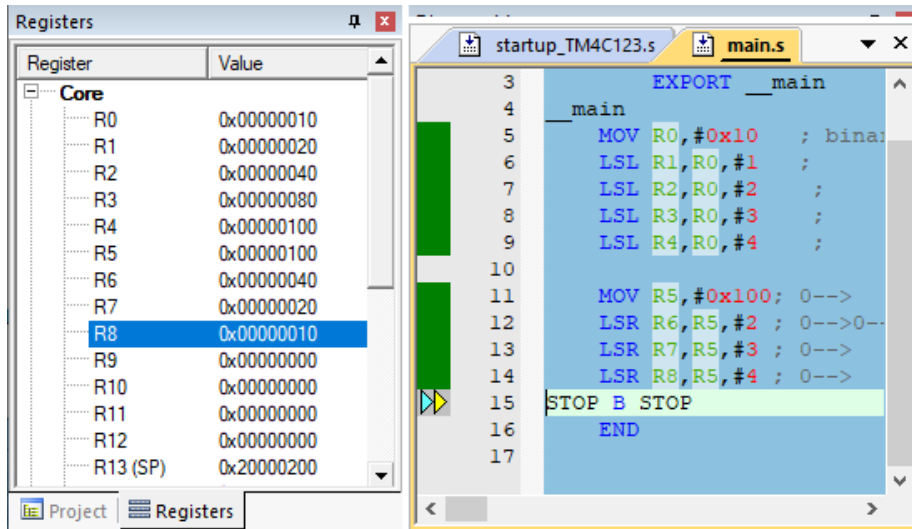
Fuente: elaboración propia, empleando Snipping Tool.

4.10.3. Análisis de la solución

Como se muestra en la figura 134 los valores a ser manipulados son 0x10 almacenado en R0 y 0x100 almacenado en R5. R0 será desplazado hacia la izquierda y por consiguiente se tendrá una multiplicación con base en la ecuación de la figura 132. R5 por otro lado tendrá un desplazamiento hacia la derecha y por consiguiente se tendrá una división con base a la ecuación de la figura 133.

Como se observa en la figura 135 el valor de R0 se multiplicó y el valor de R5 se divide, observe la figura 136 para más detalles.

Figura 135. Resultados del problema 8



Fuente: elaboración propia, empleando Snipping Tool.

Figura 136. Resultados matemáticos del problema 8

REGISTRO	VALOR INICIAL	OPERACIÓN	VALOR FINAL	VALOR FINAL EN DECIMAL
R0	0X10	$16 * 2^1$	0X20	32
	0X10	$16 * 2^2$	0X40	64
	0X10	$16 * 2^3$	0X80	128
	0X10	$16 * 2^4$	0X100	256
R5	0X100	$\frac{256}{2^2}$	0X40	64
	0X100	$\frac{256}{2^3}$	0X20	32
	0X100	$\frac{256}{2^4}$	0X10	16

Fuente: elaboración propia, empleando Adobe Photoshop.

4.11. Problema Propuesto 9

Los valores presentes en la tabla XXXVIII deben de ser almacenados en memoria y alineados a un límite de una palabra (Word,4 Bytes). Seguidamente se debe encontrar el valor menor de todos ellos y finalmente ese valor debe de ser almacenado en un registro de propósito general.

Tabla XXXVIII. **Valores del problema 9**

Valores
0XF
0x6
0x8

Fuente: elaboración propia, empleando Microsoft Word.

4.11.1. Análisis del problema

Para comprender mejor la solución se utiliza la directiva RN con fin de renombrar los registros de propósito general, como se muestran en la tabla XXXIX.

Tabla XXXIX. **Registros renombrados con RN**

Registro	Nombre asignado
R0	COUNT
R1	MIN
R2	POINTER
R3	NEXT

Fuente: elaboración propia, empleando Microsoft Word.

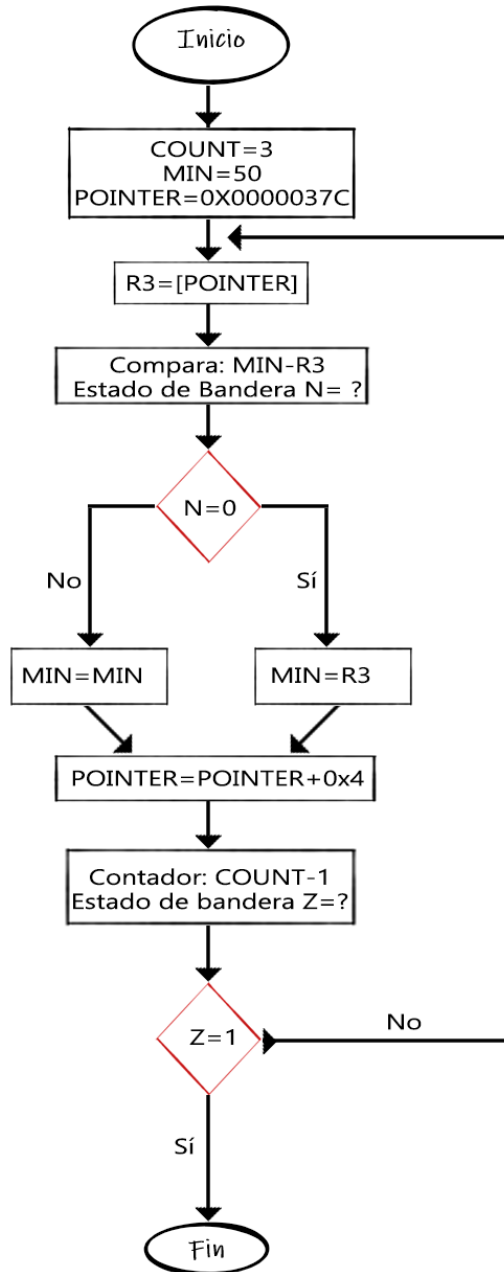
Ahora bien, COUNT almacenará el número de datos ingresados (en este caso es 3), MIN almacenará el valor mínimo encontrado, POINTER señalará la dirección de memoria en donde se ha ubicado los valores en memoria y NEXT almacenará el valor a ser comparado.

La lógica utilizada para resolver el presente problema propuesto se muestra en la figura 137. Al inicio se carga automáticamente los siguientes valores: COUNT = 3, MIN = 50 y POINTER= 0x0000037C. Luego se recupera el valor almacenado en memoria con dirección igual a 0x0000037C y se almacena en NEXT (R3).

Primera comparación, se realiza una resta entre MIN y NEXT si el resultado produce un valor positivo (bandera N = 0) el valor NEXT se almacenará en MIN de lo contrario MIN mantendrá su valor. Para ir comprando los demás valores es necesario sumarle al valor POINTER 4 bytes, de esta manera señala el siguiente valor guardado en memoria.

Segunda comparación, se debe detener el programa al finalizar de comparar los 3 valores ingresado esto con ayuda de COUNT, el cual en cada iteración decrementa su valor hasta llegar a cero. Si COUNT produce un resultado igual a cero (bandera Z = 1) el programa habrá finalizado de lo contrario se repite la primera comparación y segunda comparación.

Figura 137. Diagrama de flujo del problema 9



Fuente: elaboración propia, empleando Adobe Photoshop.

4.11.2. Solución del problema 9

La figura 138 presenta el código ideal para dar solución al problema propuesto número 9.

Figura 138. Código fuente del problema 9

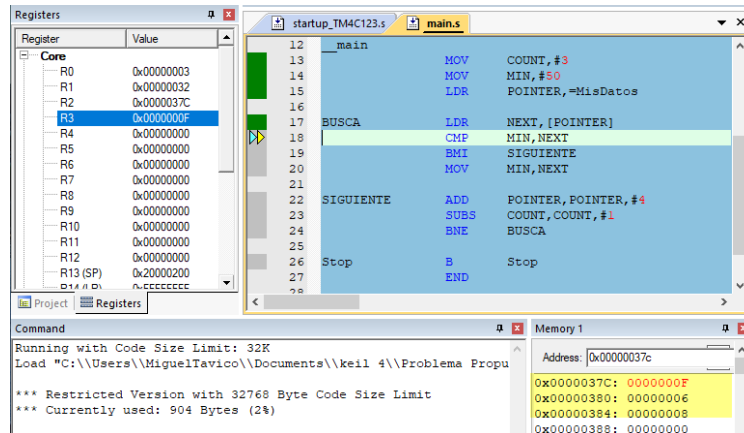
```
1  COUNT      RN      R0
2  MIN        RN      R1
3  POINTER    RN      R2
4  NEXT       RN      R3
5
6
7          AREA     Valores, DATA, READONLY
8  MisDatos  DCD     15, 6, 8
9          AREA     BuscaMin, CODE, READONLY
10         ENTRY
11         EXPORT  __main
12  __main
13         MOV     COUNT, #3
14         MOV     MIN, #50
15         LDR     POINTER, =MisDatos
16
17  BUSCA     LDR     NEXT, [POINTER]
18         CMP     MIN, NEXT
19         BMI     SIGUIENTE
20         MOV     MIN, NEXT
21
22  SIGUIENTE ADD     POINTER, POINTER, #4
23         SUBS    COUNT, COUNT, #1
24         BNE     BUSCA
25
26  Stop     B       Stop
```

Fuente: elaboración propia, empleando Snipping Tool.

4.11.3. Análisis de la solución

La figura 139 muestra que los valores son cargados de forma exitosa en memoria, el contador (R3) inicializado en 3, la dirección de memoria cargada en R2, valor mínimo propuesto almacenado en R1 (0x32) y se inicia la búsqueda del menor valor.

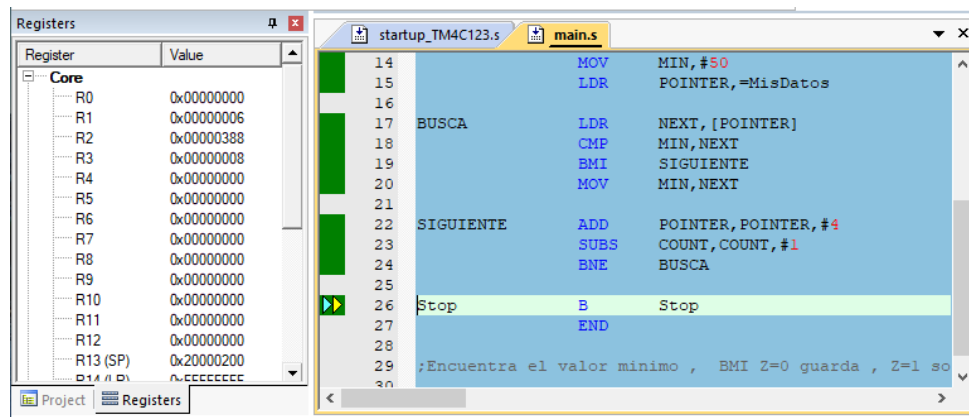
Figura 139. Valores en memoria del problema 9



Fuente: elaboración propia, empleando Snipping Tool.

En la figura 140 se observa la finalización de la búsqueda, el registro R3 tiene un valor igual a cero y el registro R1 almacena el valor más pequeño de todos los presentes en la tabla XXXVII. El valor más pequeño es R3=0x6.

Figura 140. Resultado del problema 9



Fuente: elaboración propia, empleando Snipping Tool.

4.12. Problema propuesto 10

Calcule el factorial de un número “n” suministrado por el usuario.

4.12.1. Análisis del problema

Recordando que el factorial de un entero positivo denotado por “n” es el producto de todos los números que le anteceden excluyendo el cero, la expresión matemática para encontrar el factorial se representa en la figura 141.

Figura 141. **Factorial de un número**

$$n! = n * (n - 1)(n - 2) ... (1)$$

Fuente: Elaboración propia, empleando Adobe Photoshop.

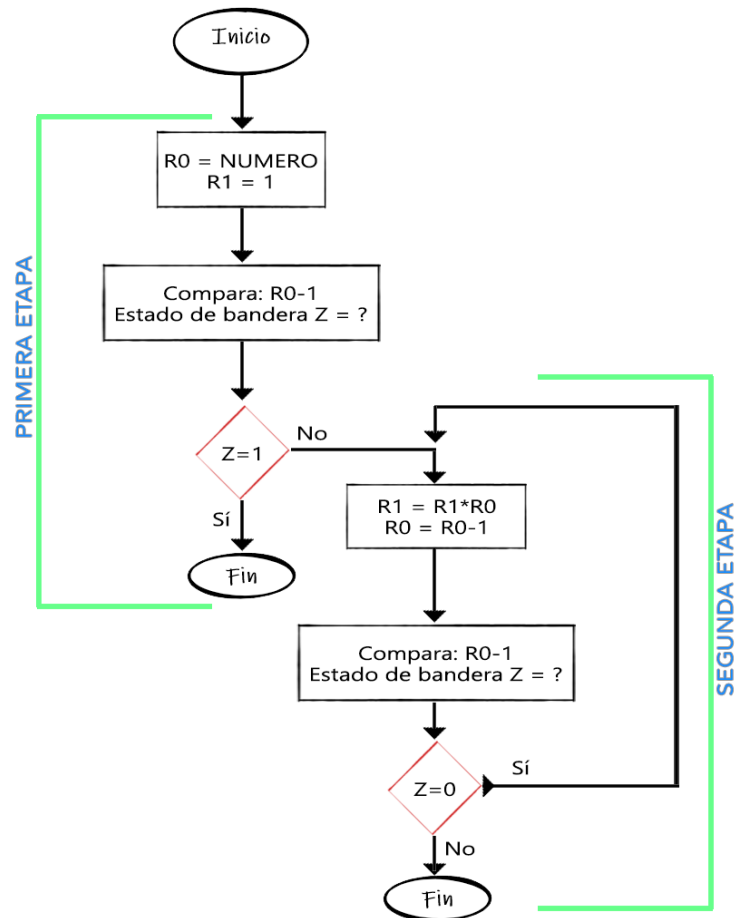
En la figura 142 se presenta el diagrama de flujo a implementar, lo primero es cargar el valor propuesto por el usuario en el registro R0 y en R1 almacenar una unidad.

Primera etapa, se procede a restar al registro R0 el valor de R1 (R0-1), si el resultado es igual a cero (bandera Z = 1) se finaliza el programa y se concluye que el valor ingresado por el usuario fue 1, ahora bien, si el resultado es distinto a cero (bandera Z = 0) se procede a cargar valores en R1 y R0.

Segunda etapa, luego de la primera comparación si el resultado es distinto a cero se procede a almacenar en R1 la multiplicación de R1 con R0 (R1*R0) y al registro R0 se le resta una unidad (R0-1). Se procede a realizar una segunda comparación mediante una resta, a R0 se le resta una unidad. Si el resultado es

igual a cero (bandera $Z = 1$) el programa habrá finalizado de lo contrario (bandera $Z = 0$) se realizará de nuevo la segunda etapa, tal y como se muestra en la figura 142.

Figura 142. Diagrama de flujo del problema 10



Fuente: elaboración propia, empleando Adobe Photoshop.

4.12.2. Solución del problema 10

La figura 143 presenta el código ideal para dar solución al problema propuesto número 10.

Figura 143. Código fuente del problema 10

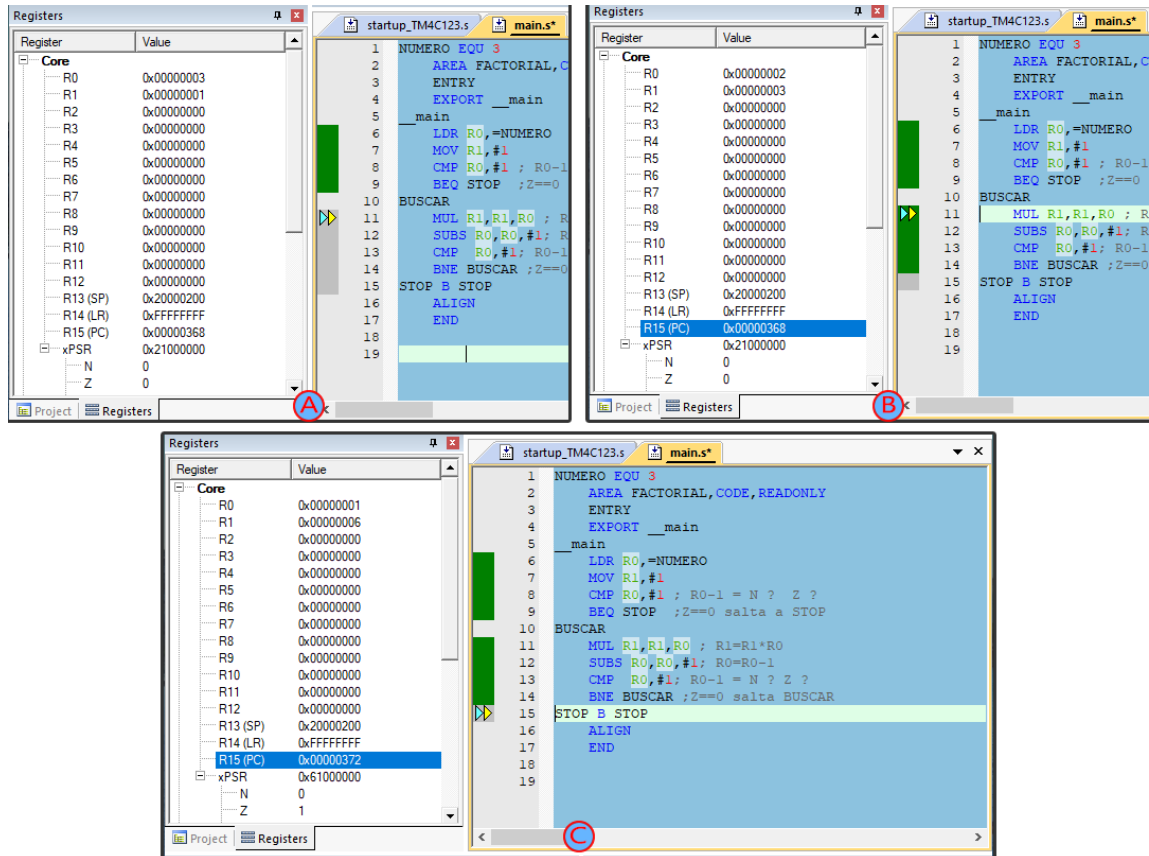
```
NUMERO EQU 3
AREA FACTORIAL, CODE, READONLY
ENTRY
EXPORT __main
__main
LDR R0, =NUMERO
MOV R1, #1
CMP R0, #1 ; R0-1 = N ? Z ?
BEQ STOP ; Z==0 salta a STOP
BUSCAR
MUL R1, R1, R0 ; R4=R4*R0
SUBS R0, R0, #1; R0=R0-1
CMP R0, #1; R0-1 = N ? Z ?
BNE BUSCAR ; Z==0 salta BUSCAR
STOP B STOP
ALIGN
END
```

Fuente: elaboración propia, empleando Snipping Tool.

4.12.3. Análisis de la solución

En la figura 114A se muestran los valores iniciales de R0 y R1 siendo estos 0x3 y 0x1 respectivamente. Al momento de iniciar las operaciones correspondientes el valor de R1 aumentará y el valor del R0 usado como contador irá disminuyendo como se observa en la figura 114B, observe que la bandera Z del registro xPSR se mantiene en cero. Finalmente, como se muestra en la figura 114C, al restar una unidad a R0 se produce un valor igual cero (bandera Z=1). Al momento de que Z obtiene un valor igual a 1 se procederá a finalizar el programa y el valor final calculado será almacenado en R1, R1 finaliza en este caso con un valor igual a 0x6 (factorial de 3 es igual a 6).

Figura 144. Resultado del problema 10



Fuente: elaboración propia, empleando Snipping Tool.

4.13. Problema Propuesto 11

Hallar la multiplicación de 0x41200000 por 0x40490FDA, los cuales deben ser almacenados en memoria, cabe resaltar que los dos valores son números de punto flotante. Recupere los valores almacenados en memoria y cárguelos en registros de punto flotante, como se muestra en la tabla XL.

Tabla XL. **Valores para problema 11**

Registro	Valor hexadecimal	en	Valor en sistema decimal
S1	0x41200000		10,0
S3	0x40490FDA		3,1415925025939940

Fuente: elaboración propia, empleando Microsoft Word.

4.13.1. Análisis del problema

Por construcción, al momento de utilizar la unidad para punto flotante es necesario inicializar la unidad, esto debido a que por ahorro de energía esta unidad viene deshabilitada por defecto. Es necesario configurar el registro CPACR con las instrucciones mostradas en la figura 145.

Figura 145. **Configuración del registro CPACR**

```
LDR R0, =0xE000ED88; Registro CPACR
LDR R1, [R0]
ORR R1, R1, # (0xF<<20)
STR R1, [R0]
```

Fuente: elaboración propia, empleando Snipping Tool.

Para la resolución del presente problema se procederá a usar las instrucciones para datos con punto flotante. Debe tomar en cuenta que todas las operaciones son de 32 bits por lo que siempre se utilizara el sufijo “F32” y se utilizaran los registros en el rango de S0-S30.

4.13.2. Solución del problema 11

La figura 146 presenta el código ideal para dar solución al problema propuesto número 11.

Figura 146. Código fuente del problema 11

```
AREA Problema11, CODE, READONLY
ENTRY
EXPORT __main
__main

LDR R0, =0xE000ED88 ;Registro CPACR
LDR R1, [R0]
ORR R1, R1, # (0xF<<20)
STR R1, [R0]

ADRL R1, VALOR1
VLDR.F32 S2, [R1]
VLDR.F32 S3, [R1, #4]
VMUL.F32 S4, S2, S3 ; 31.415924

STOP B STOP

VALOR1 DCD 0x41200000 ; 10.0
VALOR2 DCD 0x40490FDA ; 3.1415925025939940

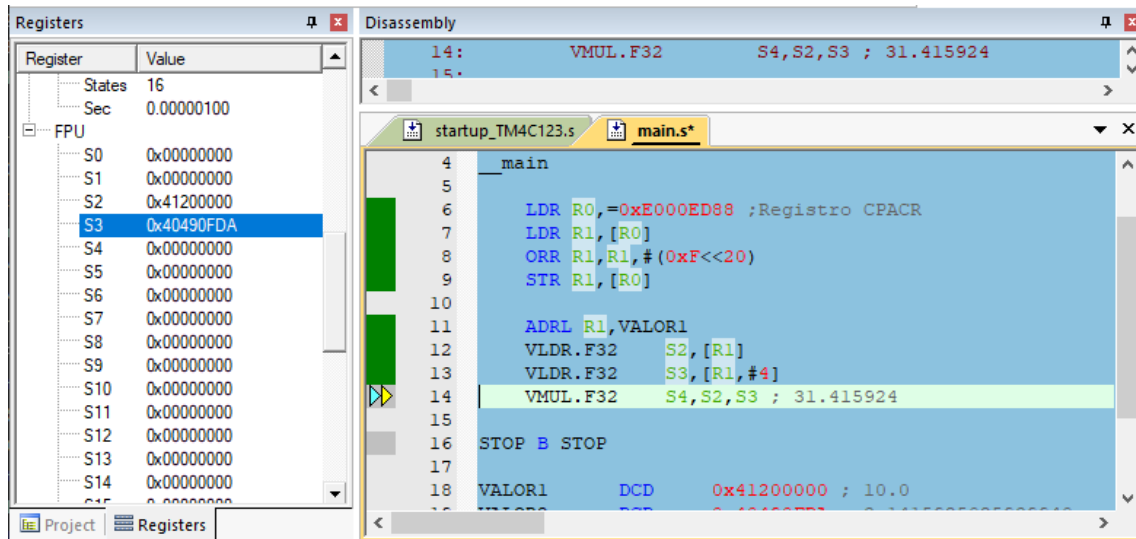
ALIGN
END
```

Fuente: elaboración propia, empleando Snipping Tool.

4.13.3. Análisis de la solución

En el inicio del código se procederá a habilitar la unidad de punto flotante, seguidamente se recuperará los valores almacenados en memoria y son cargados en los registros S2 y S3 como se muestra en la figura 147. Nótese que ahora se están usando los registros para datos con punto flotante en la FPU.

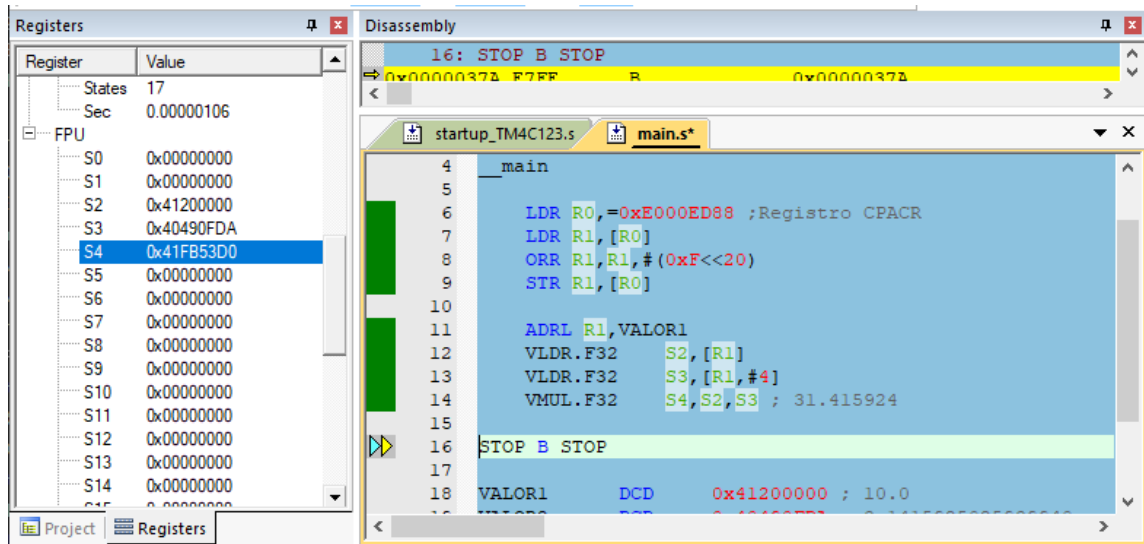
Figura 147. Valores dentro de FPU



Fuente: elaboración propia, empleando Snipping Tool.

Finalmente, en el registro S4 se almacena la multiplicación de S2 por S3, como se muestra en la figura 148. El valor obtenido de la multiplicación es 0x41FB53D0 (31,415924 sistema decimal).

Figura 148. Resultados del problema 11



Fuente: elaboración propia, empleando Snipping Tool.

4.14. Problema Propuesto 12

Demuestre el uso correcto de las instrucciones PUSH, VPUSH, POP y VPOP.

4.14.1. Análisis del problema

El uso principal de PUSH o VPUSH es para conservar sin cambios un valor guardado en un registro en particular. Esas instrucciones son utilizadas para guardar el contenido de un registro en memoria, cuando se procederá a llamar a una función o subrutina y dicho valor en registro será reutilizado. La instrucción PUSH guardar el dato en memoria y con la instrucción POP se recupera cuando el usuario lo requiera.

4.14.2. Solución de problema 12

La figura 149 presenta la solución del problema propuesto número 12.

Figura 149. Código fuente del problema 12

```
1      AREA Problema12, CODE, READONLY
2      ENTRY
3      EXPORT __main
4
5      __main
6      LDR R0,=0xE000ED88
7      LDR R1,[R0]
8      ORR R1,R1,#(0xF<<20)
9      STR R1,[R0]
10
11     VLDR.F32 S0,=3.14159265359 ; 0x40490FDA
12     VLDR.F32 S1,=2.71828182846 ; 0x402DF854
13     LDR R0,=0xDEADBEEF
14     LDR R1,=0xBABEFACE
15     VPUSH {S0} ; 3.14159265359 ; 0x40490FDA
16     VPUSH {S1} ; 2.71828182846 ; 0x402DF854
17     PUSH {R0} ; DEADBEEF
18     PUSH {R1} ; BABEFACE
19
20     BL ENTEROS
21     BL DECIMALES
22     POP {R1}
23     POP {R0}
24     VPOP {S1}
25     VPOP {S0}
26     B STOP
27
28     ENTEROS
29     MUL R0,R0,R1
30     ADD R1,R1,R0
31     BX LR
32
33     DECIMALES
34     VMUL.F32 S0,S0,S1
35     VADD.F32 S1,S1,S0
36     BX LR
37
38     STOP B STOP
39
40     ALIGN
41     END
```

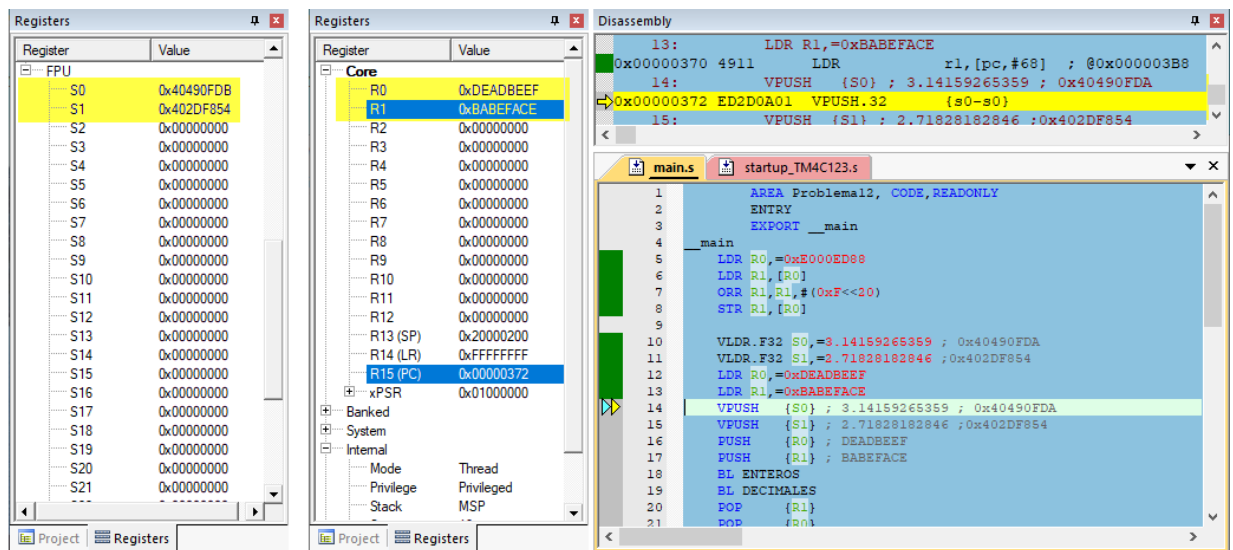
Fuente: elaboración propia, empleando Snipping Tool.

4.14.3. Análisis de solución

Como se observa en la figura 150 los valores propuestos por el usuario han sido almacenados en los registros de propósito general y registros de punto flotante. Seguidamente se procede a almacenar los cuatro valores propuestos en la memoria, con el fin de guardar la información y no perderla en próximas rutinas

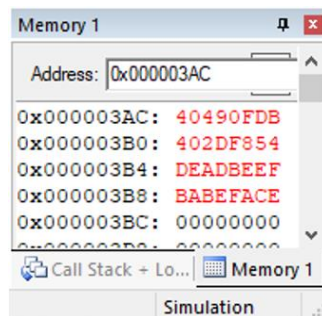
o funciones, los valores quedan almacenados en memoria con la ayuda de la instrucción PUSH (instrucción para datos enteros) y V PUSH (instrucción para datos con punto flotante) como se muestra en la figura 151.

Figura 150. Valores cargados en los registros



Fuente: elaboración propia, empleando Snipping Tool.

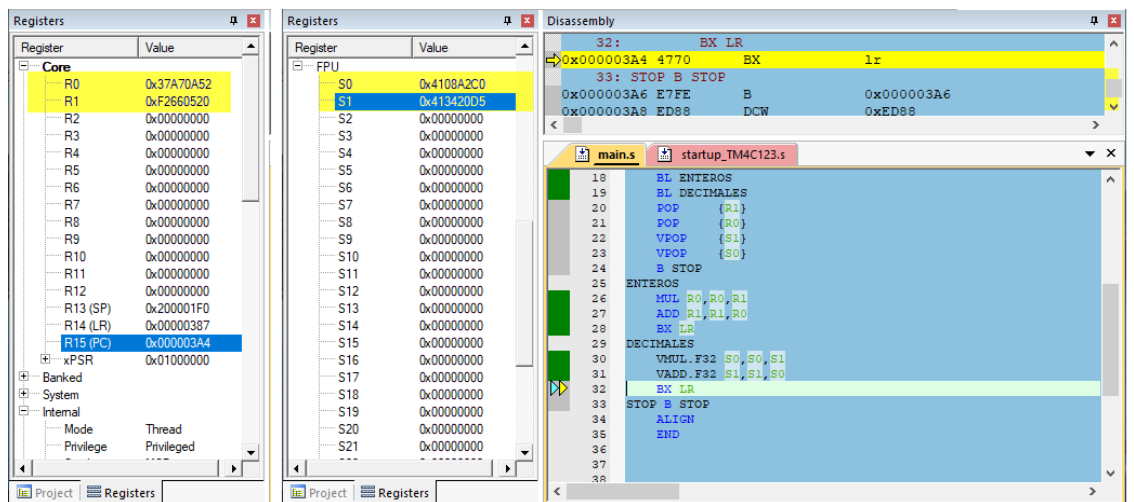
Figura 151. Valores cargados en memoria



Fuente: elaboración propia, empleando Snipping Tool.

Luego, como se observa en las líneas 18 y 19 de la figura 149 los registros R0, R1, S0 y S1 son reutilizados y el contenido inicial de todos ellos cambia como se observa en la figura 152.

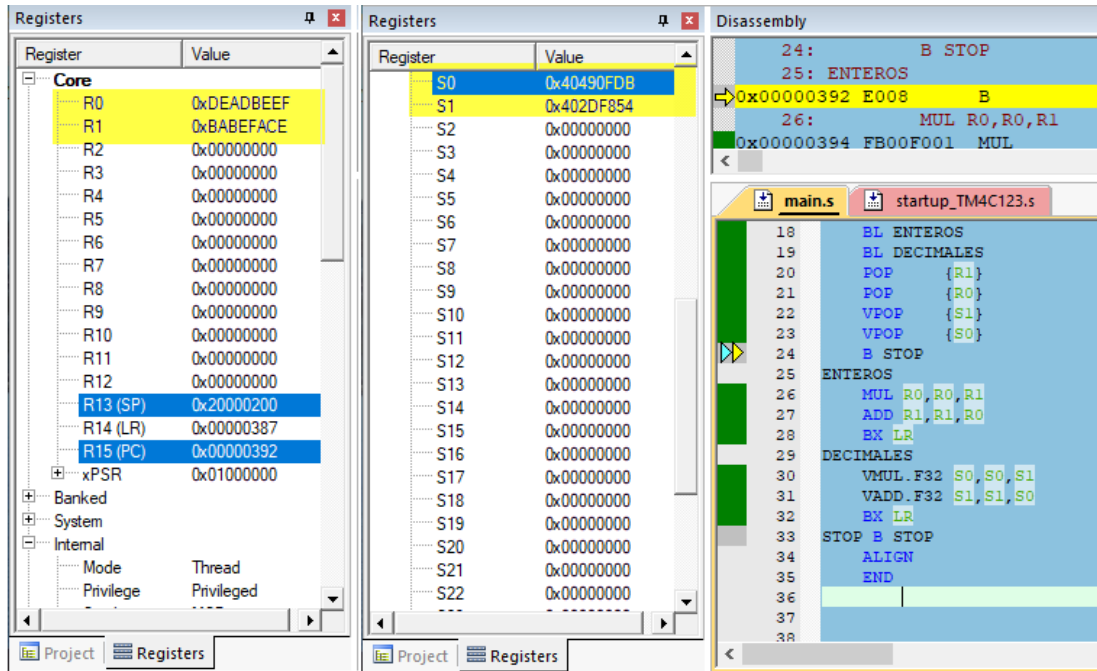
Figura 152. Manipulación de valores en los registros



Fuente: elaboración propia, empleando Snipping Tool.

Finalmente, al terminar de realizar las subrutinas llamadas ENTEROS y DECIMALES se procede a recuperar los valores iniciales de R0, R1, S1 y S0, esto con las instrucciones POP y VPOP correspondientemente. Como se observa en la figura 153, los valores de los registros regresan a su valor inicial.

Figura 153. Resultado del problema 12



Fuente: elaboración propia, empleando Snipping Tool.

4.15. Problema Propuesto 13

Una esfera de 0,600 kg tiene una rapidez de 15 m/s ¿Cuál es su energía cinética?

Figura 154. Energía cinética

$$\text{Energía Cinética} = K = \frac{1}{2}mv^2$$

Fuente: elaboración propia, empleando Adobe Photoshop.

4.15.1. Análisis del problema

Recuerde que la energía cinética de un objeto surge del trabajo neto realizado sobre él. Para encontrar la energía cinética de la esfera se debe utilizar la ecuación de la figura 154, en donde $m=0,600\text{kg}$ y $v=15\text{ m/s}$. Matemáticamente hablando es una ecuación simple por resolver sin embargo en lenguaje ensamblador se requiere de pasos extras.

Debemos notar que el número 15 es de tipo entero o *integer* y 0,600 de tipo decimal. Por facilidad se convertirá el valor entero a decimal, esto con el uso de la instrucción VCVT.F32.U32 seguidamente se procederá a realizar todas las operaciones matemáticas requeridas.

4.15.2. Solución de problema 13

La figura 155 presenta la solución del problema propuesto número 13.

Figura 155. Código fuente del problema 13

```
1 VELOCIDAD RN R1
2 MASA SN S1
3 AREA Problema13, CODE, READONLY
4 ENTRY
5 EXPORT __main
6 __main
7 LDR R0,=0xE000ED88
8 LDR R1,[R0]
9 ORR R1,R1,#(0xF<<20)
10 STR R1,[R0]
11
12 VLDR.F32 S0,=2.0
13 MOV VELOCIDAD,#15
14 VLDR.F32 MASA,=0.600
15 VMOV S2,VELOCIDAD ; Copia los bits de R1 (Integer) A S2
16 VCVT.F32.U32 S4,S2; Convierte los bits de S2 a S4 (Floating point)
17 ;S4=15.0
18 VMUL.F32 S5,MASA,S4
19 VMUL.F32 S5,S5,S4
20 VDIV.F32 S5,S5,S0
21 STOP B STOP
22 ALIGN
23 END
```

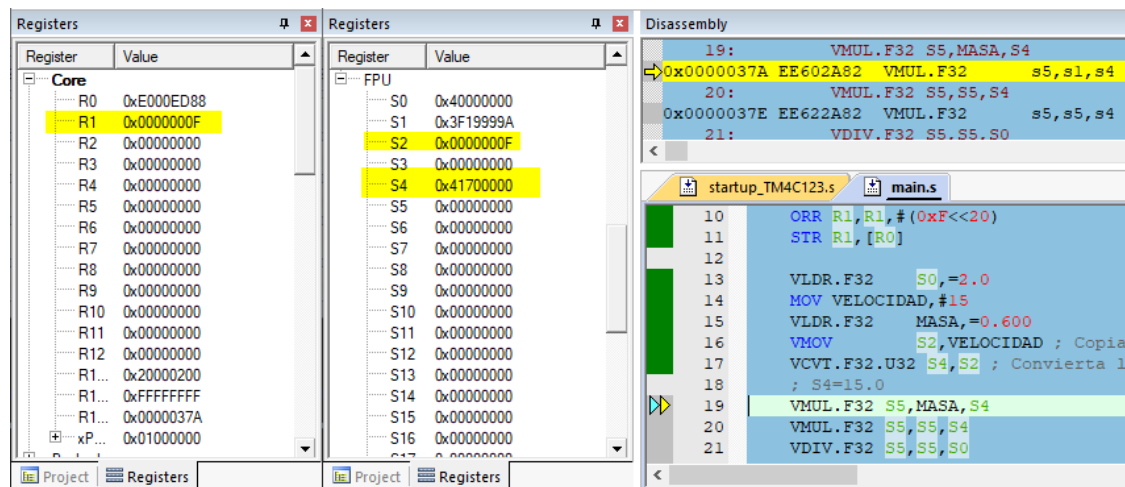
Fuente: elaboración propia, empleando Snipping Tool.

4.15.3. Análisis de la Solución

Como se observa en la línea 13 de la figura 155 se movió el valor 15 hacia el registro renombrado como VELOCIDAD (R1) el cual es de tipo “integer”, debe tener claro que es imposible operar datos de tipo “integer” con datos con punto flotante. Los bits cargados en R1 posteriormente serán copiados a S2 y luego los bits son convertidos a un valor de punto flotante y almacenado en S4 con ayuda de la instrucción VCVT.F32.U32. El valor de 15 en VELOCIDAD finalmente se almacenó en S4 ya como un valor de punto flotante (15,0).

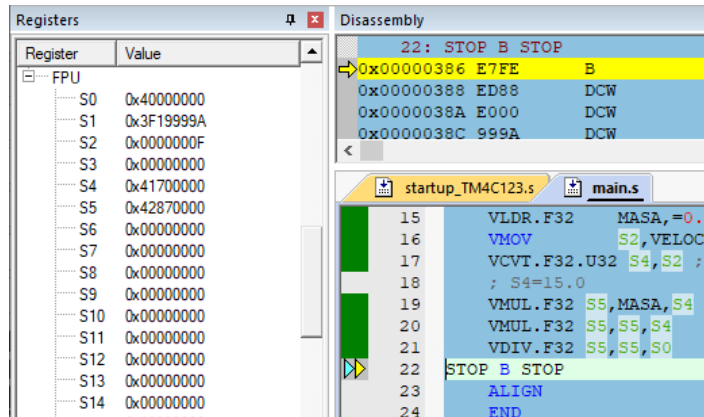
Observe en la figura 156 que R1 contiene el valor de 15 (0xF en sistema hexadecimal), seguidamente se confirma el correcto almacenamiento de 15,0 (0x41700000 en sistema hexadecimal) en S4.

Figura 156. Conversión a decimal



Fuente: elaboración propia, empleando Snipping Tool.

Figura 157. Resultado del problema 13



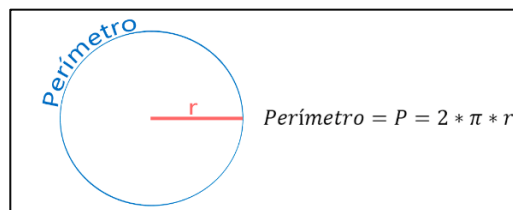
Fuente: elaboración propia, empleando Snipping Tool.

Finalmente se logra encontrar la energía cinética de la esfera la cual es igual a $S5=0x42870000$ (67,50 Julios en sistema decimal), como se muestra en la figura 157.

4.16. Problema Propuesto 14

Calculo del perímetro para un círculo de radio igual a 4,12 cm.

Figura 158. Perímetro de un círculo



Fuente: elaboración propia, empleando Adobe Photoshop.

4.16.1. Análisis del problema

En este caso, el valor 4,12 será almacenado en memoria y posteriormente será recuperado con la ayuda de la instrucción VLDR.F32 la cual es especial para recuperar datos con punto flotante. Luego, con base en la ecuación presente en la figura 158 se procede a dar solución al problema, cabe destacar que el número 2 es un valor entero. La instrucción VCVT.F32.U32 nos ayuda a convertir un valor entero a un valor con punto flotante, de esta forma 2 pasa a ser 2,0.

4.16.2. Solución del problema 14

La figura 159 presenta el código ideal para dar solución al problema propuesto número 14.

Figura 159. Código fuente del problema 14

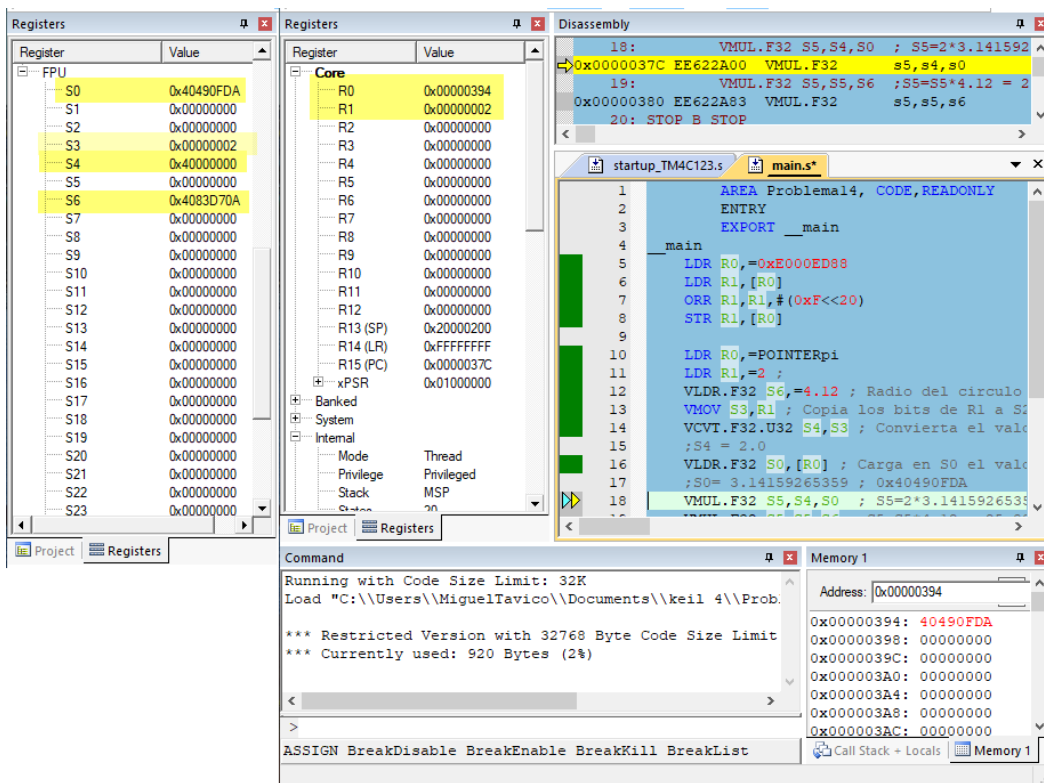
```
1 AREA Problema14, CODE, READONLY
2 ENTRY
3 EXPORT __main
4
5 __main
6 LDR R0, =0xE000ED88
7 LDR R1, [R0]
8 ORR R1, R1, # (0xF<<20)
9 STR R1, [R0]
10
11 LDR R0, =POINTERpi
12 LDR R1, =2 ;
13 VLDR.F32 S6, =4.12 ; Radio del circulo en centímetros
14 VMOV S3, R1 ; Copia los bits de R1 a S2
15 VCVT.F32.U32 S4, S3 ; Convierta el valor de R1 a S4 {Floating point}
16 ;S4 = 2.0
17 VLDR.F32 S0, [R0] ; Carga en S0 el valor econtrado en la direccion que apunta POINTERpi
18 ;S0= 3.14159265359 ; 0x40490FDA
19 VMUL.F32 S5, S4, S0 ; S5=2*3.14159265359
20 VMUL.F32 S5, S5, S6 ; S5=S5*4.12 = 25.886720657348633
21 STOP B STOP
22 AREA EQ_2, DATA, READONLY
23 POINTERpi DCD 0x40490FDA ; pi->3.14159265359;0x40490FDA
24 ALIGN
25 END
```

Fuente: elaboración propia, empleando Snipping Tool.

4.16.3. Análisis de la solución

Como se observa en la figura 159 el valor de 2 es cargado en R1 posteriormente los bits son copiados a S3 y luego los bits son convertidos a un valor de punto flotante y almacenado en S4 esto con ayuda de la instrucción VCVT.F32.U32. El valor de pi se encuentra almacenado en memoria, se recupera y se almacena en S0.

Figura 160. Valores almacenados en registros

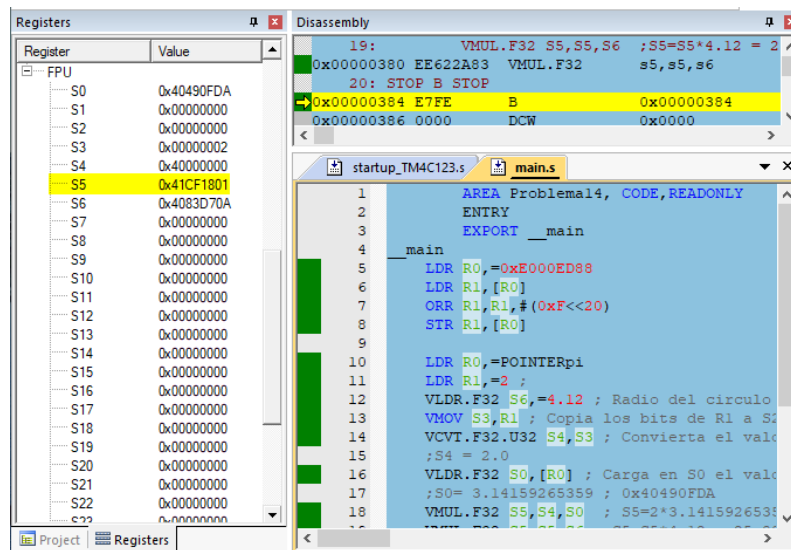


Fuente: elaboración propia, empleando Snipping Tool.

Como se observa en la figura 160 los valores en S4, S0 y S6 son los correctos, finalmente se procede a realizar las operaciones aritméticas

correspondientes para encontrar el perímetro. El perímetro encontrado y almacenado en S5 es 0x41CF1801 (25,886 cm en sistema decimal), como se muestra en la figura 161.

Figura 161. Resultado del problema 14

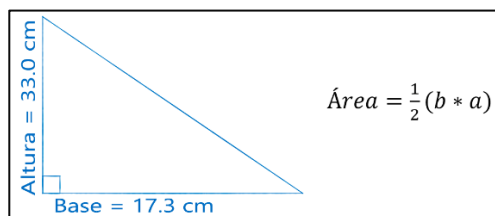


Fuente: elaboración propia, empleando Snipping Tool.

4.17. Problema Propuesto 15

Calculo del área del triángulo rectángulo mostrado en la figura 162.

Figura 162. **Área de un triángulo rectángulo**



Fuente: elaboración propia, empleando Adobe Photoshop.

4.17.1. Análisis del problema

Para encontrar el área del triángulo equilátero se utiliza la ecuación mostrada en la figura 162, se dará solución mediante las multiplicaciones de tres términos (S0, S1 y S2) como se muestra en la figura 163.

4.17.2. Solución del problema 15

La figura 163 presenta el código ideal para dar solución al problema propuesto número 15.

Figura 163. Código fuente del problema 15

```
AREA Problema15, CODE ,READONLY
ENTRY
EXPORT __main
__main
LDR R0,=0xE000ED88
LDR R1,[R0]
ORR R1,R1,#(0xF<<20)
STR R1,[R0]

VMOV.F32 S0,#0.5
VLDR.F32 S1,=17.3 ; Base
VLDR.F32 S2,=33.0 ; Altura

VMUL.F32 S3,S0,S1
VMUL.F32 S3,S3,S2

STOP B STOP

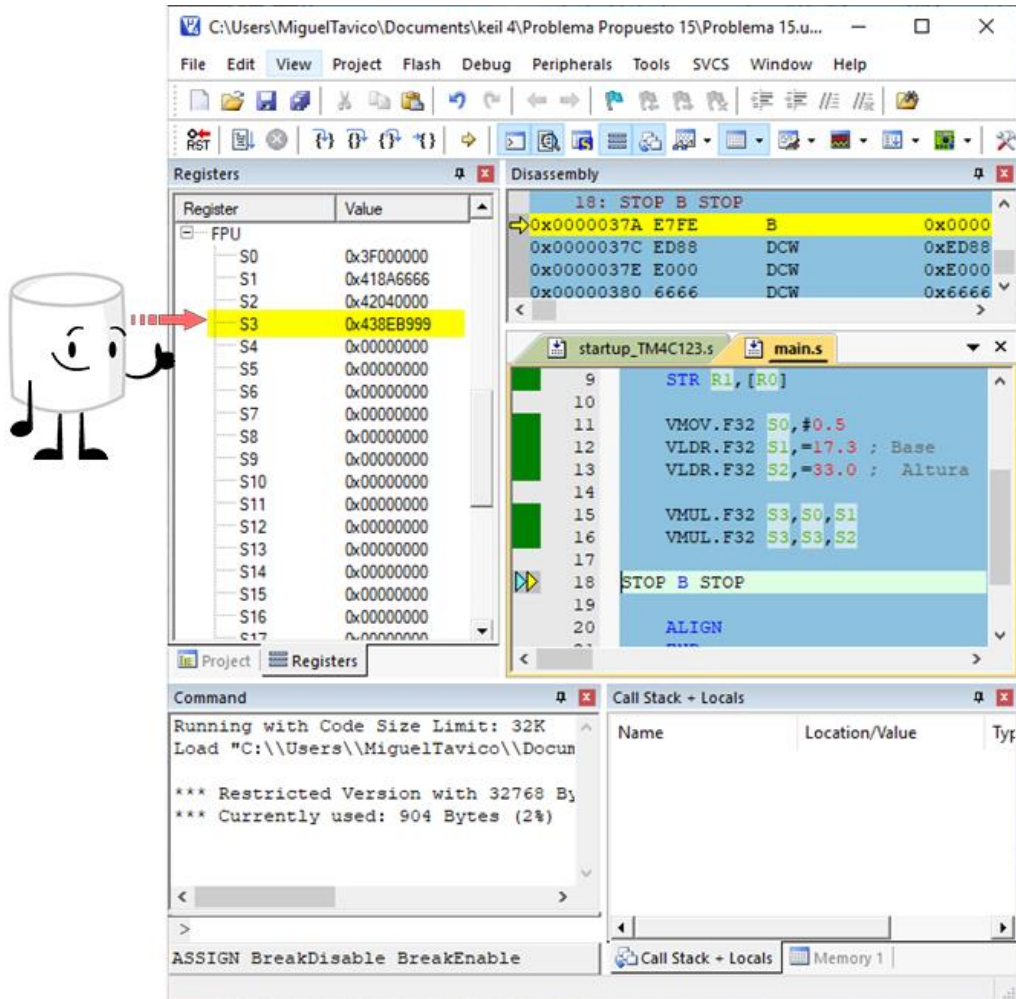
ALIGN
END
```

Fuente: elaboración propia, empleando Snipping Tool.

4.17.3. Análisis de la solución

Las medidas del triángulo fueron almacenadas en S1 y S2, la solución final se almacenó en el registro S3 como se muestra en la figura 164. El área del triángulo rectángulo es $S3 = 0x438EB999$ (285,45 cm² en sistema decimal).

Figura 164. Resultado del problema 15

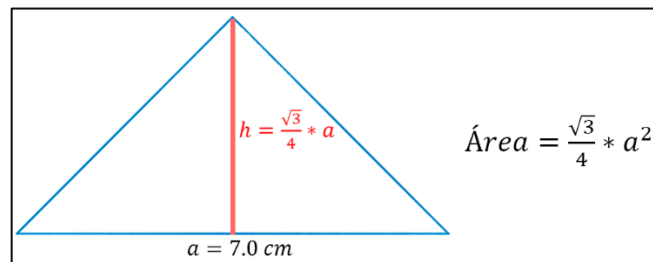


Fuente: elaboración propia, empleando Snipping Tool.

4.18. Problema propuesto 16

Calcule el área del triángulo equilátero mostrado en la figura 165.

Figura 165. **Área de un triángulo equilátero**



Fuente: elaboración propia, empleando Adobe Photoshop.

4.18.1. Análisis del problema

Para hallar el área del triángulo equilátero propuesto se debe utilizar la ecuación mostrada en la figura 165. Las operaciones matemáticas para valores con punto flotante a utilizar son las siguientes: VSQRT.F32, VMUL.F32 y VDIV.F32.

4.18.2. Solución del problema 16

La figura 166 presenta el código ideal para dar solución al problema propuesto número 16.

Figura 166. Código fuente del problema 16

```
1      AREA Problema16, CODE, READONLY
2      ENTRY
3      EXPORT __main
4  __main
5      LDR R0, =0xE000ED88
6      LDR R1, [R0]
7      ORR R1, R1, # (0xF<<20)
8      STR R1, [R0]
9
10     VMOV.F32 S0, #4.0
11     VLDR.F32 S1, =3.0
12     VLDR.F32 S2, =7.0 ; a
13     VSQRT.F32 S1, S1 ; Raiz cuadrada de S1
14     VMUL.F32 S2, S2, S2 ; S2=S2*S2
15
16     VMUL.F32 S3, S1, S2
17     VDIV.F32 S3, S3, S0
18     STOP B STOP
19     ALIGN
20     END
```

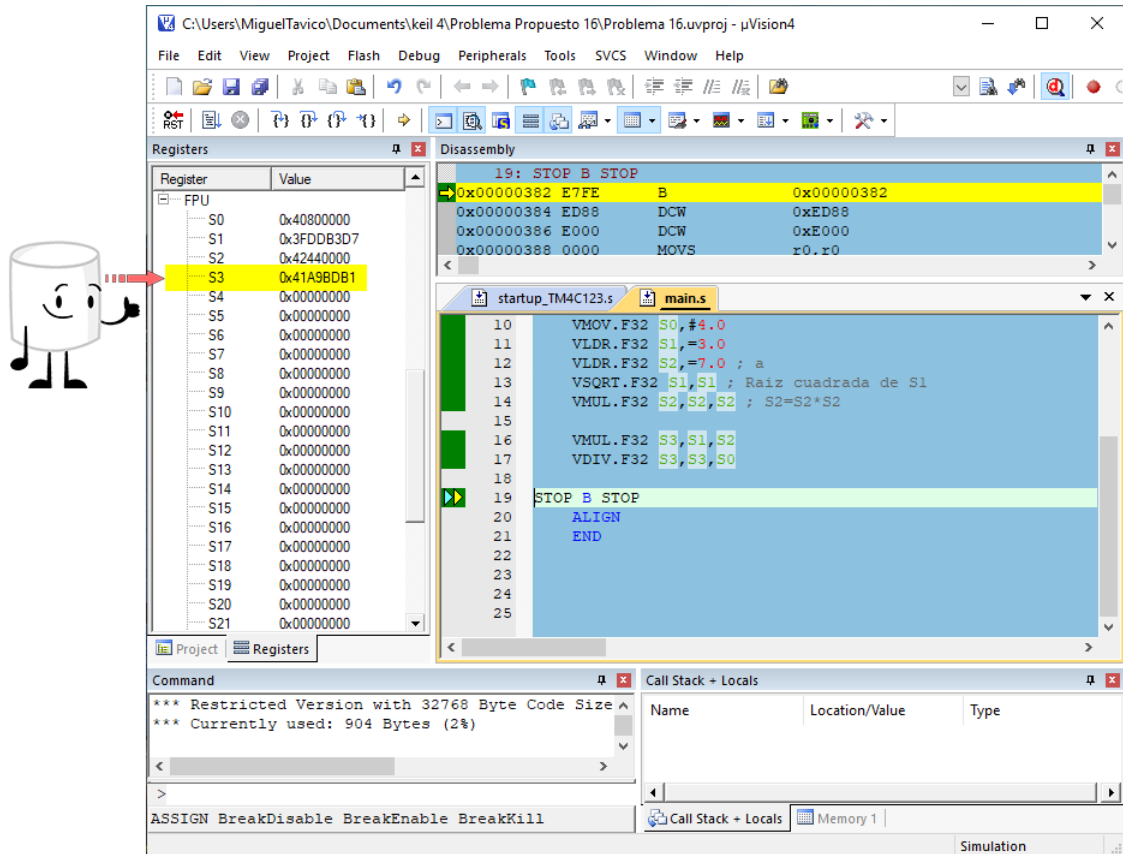
Fuente: elaboración propia, empleando Snipping Tool.

4.18.3. Análisis de la solución

Como se muestra en la figura 166 los valores necesarios para encontrar el área del triángulo equilátero fueron ingresados en los registros S0, S1 y S2. Los registros fueron manipulados con operaciones matemáticas para encontrar el área.

En la figura 167 se presenta el resultado final, el área total es S3=0x41A9BDB1 (21,217623 cm² en sistema decimal).

Figura 167. Solución del problema 16



Fuente: elaboración propia, empleando Snipping Tool.

4.19. Problema Propuesto 17

Encuentre el tiempo promedio que se tarda un auto en dar la vuelta a la pista, ver la tabla XLI.

Tabla XLI. **Vueltas y tiempos**

Vuelta	Tiempo (segundos)
Primera	39,15
Segunda	35,50
Tercera	33,32
Cuarta	37,33

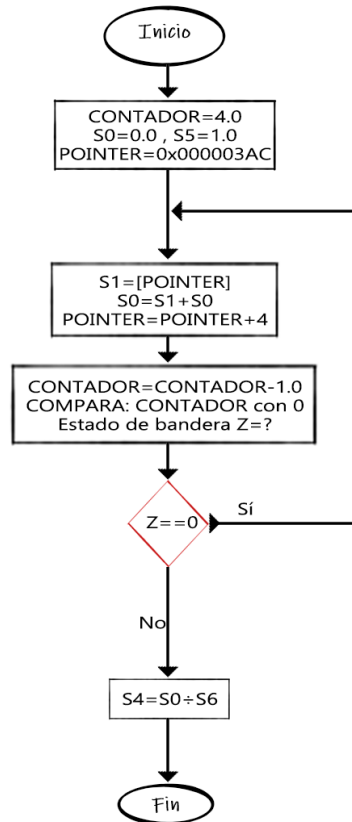
Fuente: elaboración propia, empleando Microsoft Word.

4.19.1. Análisis del problema

Los valores mostrados en la tabla XLI serán almacenados en memoria con ayuda de la directiva DCFS. La directiva DCFS trabaja con valores de punto flotante de precisión simple y los coloca a un límite de una palabra (en inglés *word*, 4 bytes). Para visualizar mejor la solución se procederá a utilizar la directiva RN y SN las cual ayuda a renombrar los registros.

Se utilizará la instrucción VCVT.U32.F32 la cual convierte un valor de punto flotante a un valor tipo entero (en inglés *Integer*) además se utilizará la instrucción VMRS APSR_nzcv,FPSCR para copiar el estado de las banderas de la unidad de punto flotante hacia el registro xPSR en el núcleo.

Figura 168. Diagrama de flujo del problema 17



Fuente: elaboración propia, empleando Adobe Photoshop.

Como se muestra en el diagrama de flujo en la figura 168, el contador inicia en 4,0, S0 en 0,0, S5 en 1,0 y la dirección en memoria se almacena en POINTER.

Primer paso, en cada ciclo S0 sumará los valores almacenados en memoria mientras a POINTER se le suma 4 bytes, esto ya que los datos están alineados al límite de una palabra.

Segundo paso, el valor de CONTADOR se decrementará en 1,0 y luego se comparará con el valor de 0,0, lo que hará un cambio en las banderas de punto

flotante (FPSCR). El estado de las banderas en FPSCR son necesarias copiarlas al registro central xPSR.

Mientras el valor de la bandera $Z = 0$ se repetirá el primer y segundo paso de lo contrario (bandera $Z = 1$) el valor final almacenado en $S0$ se dividirá entre 4,0.

4.19.2. Solución del problema 17

La figura 169 presenta el código ideal para dar solución al problema propuesto número 17.

Figura 169. Código fuente del problema 17

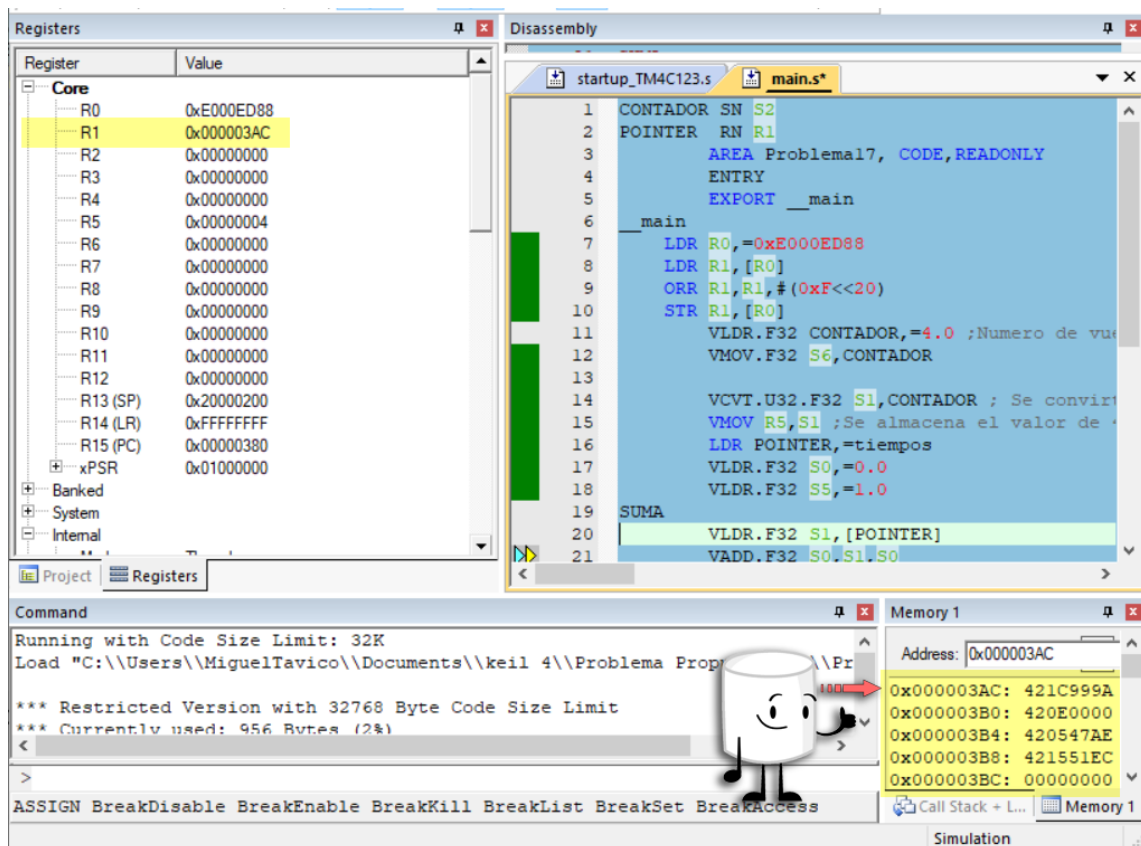
```
1  CONTADOR SN S2
2  POINTER RN R1
3      AREA Problema17, CODE, READONLY
4      ENTRY
5      EXPORT __main
6  __main
7      LDR R0,=0xE000ED88
8      LDR R1,[R0]
9      ORR R1,R1,#(0xF<<20)
10     STR R1,[R0]
11     VLDR.F32 CONTADOR,=4.0 ;Numero de vueltas
12     VMOV.F32 S6,CONTADOR
13
14     VCVT.U32.F32 S1,CONTADOR ; Se convirtio 4.0 a 4 (integer)
15     VMOV R5,S1 ;Se almacena el valor de 4 en R5
16     LDR POINTER,=tiempos
17     VLDR.F32 S0,=0.0
18     VLDR.F32 S5,=1.0
19
20     SUMA
21     VLDR.F32 S1,[POINTER]
22     VADD.F32 S0,S1,S0
23     ADD POINTER,POINTER,R5 ; 4 bytes
24     VSUB.F32 CONTADOR,CONTADOR,S5
25     VCMP.F32 CONTADOR,#0
26     VMRS APSR_nzcv,FPSCR
27     BNE SUMA
28
29     DIVIDE
30     VDIV.F32 S4,S0,S6
31     B STOP
32
33     STOP B STOP
34
35     AREA Valores, DATA, READONLY
36     tiempos DCFS 39.15,35.50,33.32,37.33
37
38     ALIGN
39
40     END
```

Fuente: elaboración propia, empleando Adobe Photoshop.

4.19.3. Análisis de la solución

En la figura 170 se observa el correcto almacenamiento de los datos en memoria, el primer dato es almacenado en la memoria con dirección igual a 0x000003AC.

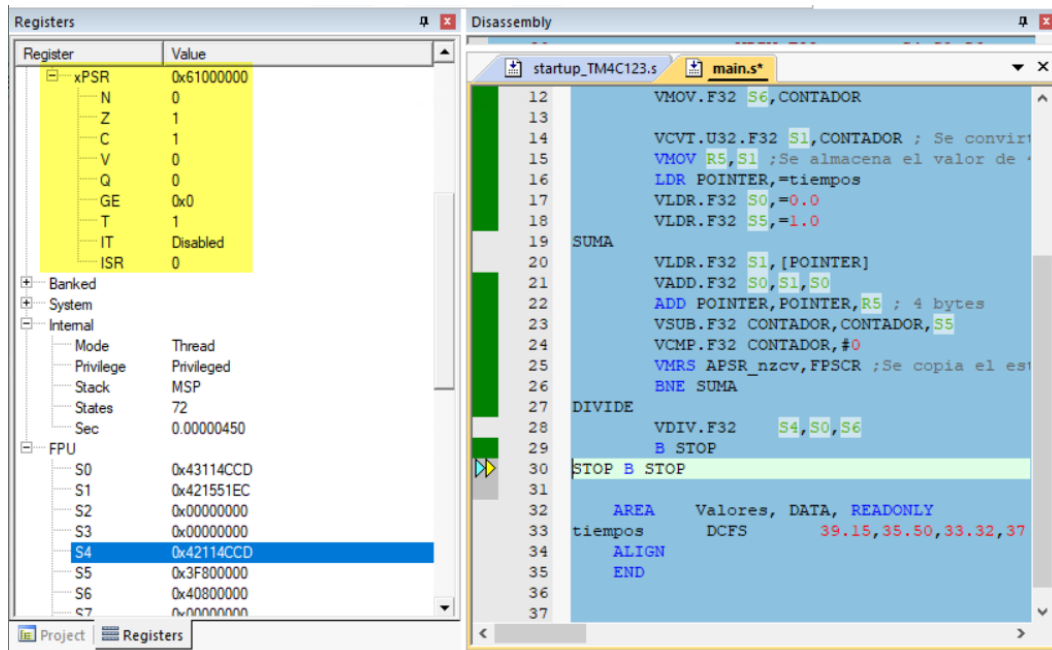
Figura 170. Datos del problema 17



Fuente: elaboración propia, empleando Snipping Tool.

En la figura 171 se observa que las banderas del registro xPSR han sido copiadas con éxito, al momento de tener la bandera Z = 1 sale del ciclo que tiene por etiqueta SUMA y salta a la etiqueta DIVIDE en donde se procede a dividir la suma acumulada de S4 entre S6 (4,0). El valor final se almacena en S4=0x42114CCD (36,325 segundos en sistema decimal).

Figura 171. Resultado del problema 17



Fuente: elaboración propia, empleando Snipping Tool.

4.20. Problema Propuesto 18

Calcule el $\text{sen}(x=1,08381)$ mediante polinomios de Taylor.

Figura 172. **Aproximación de Sen(x)**

$$\text{Sin}(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}; \forall x \in R$$

Fuente: elaboración propia, empleando Adobe Photoshop.

4.20.1. Análisis del problema

Calcular el $\text{sen}(x=4)$ con una calculadora científica es muy sencillo, pero muchas veces se tienen limitaciones y no se puede recurrir a un segundo equipo para solucionar problemas matemáticos en tiempo real.

En el momento de usar procesadores de la familia Cortex®-M se vuelve un tanto complicado darle solución a una función $\text{sin}(x)$ ya que esa función no está disponible en el conjunto de instrucciones como lo son SDIV, VMUL, SDIV y otras.

Cuando no se dispone directamente de la función $\text{sen}(x)$, se procede a realizar una aproximación de la función mediante una serie de potencias o suma de potencias conocidas como polinomios de Taylor. Al realizar el correcto tratamiento para calcular la función de $\text{sen}(x)$ se obtiene la suma de potencias mostrada en la figura 173.

Simplificando la ecuación 172, para $n = 3$ se obtiene:

Figura 173. **Aproximación de Sen(x) con $n = 3$**

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}$$

Fuente: elaboración propia, empleando Adobe Photoshop.

4.20.2. Solución del problema 18

La figura 174 presenta el código ideal para dar solución al problema propuesto número 18.

Figura 174. Código fuente del problema 18

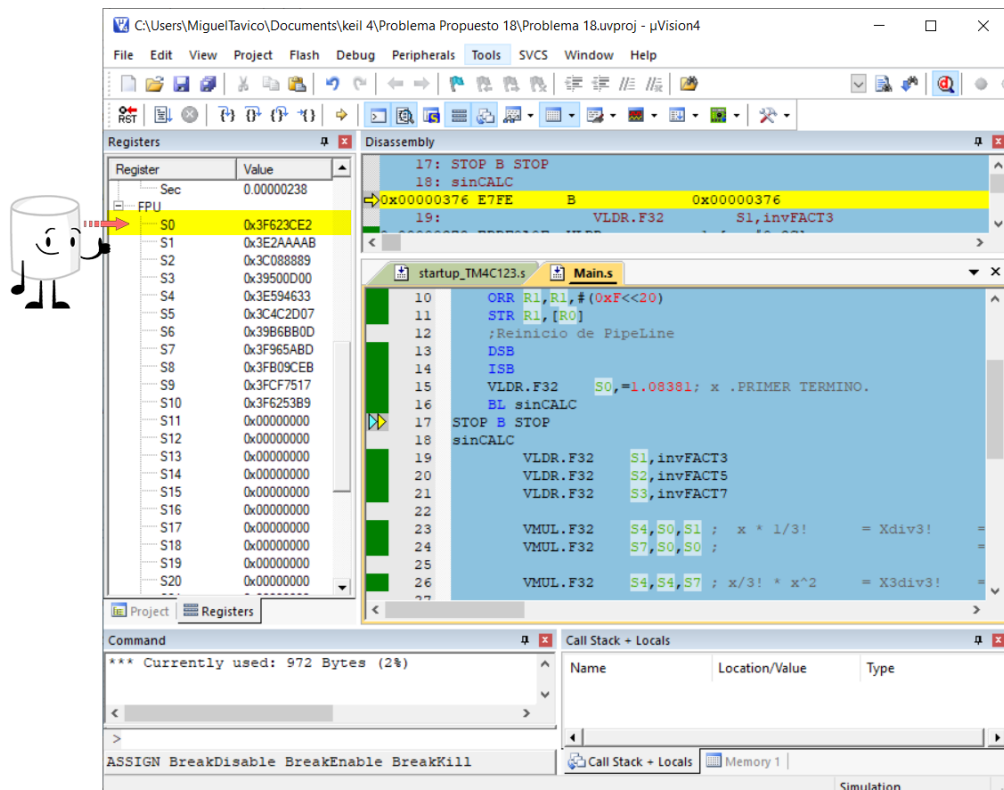
```
startup_TM4C123.s  Main.s
1  ;Para X en [0,pi/2]
2  ;Al definir los valores 1/3!, 1/5! y 1/7! como constantes se ahorra ciclos de reloj
3  AREA Problema18, CODE,READONLY
4  ENTRY
5  EXPORT __main
6  __main
7
8  LDR R0,=0xE000ED88
9  LDR R1,[R0]
10 ORR R1,R1,#(0xF<<20)
11 STR R1,[R0]
12 ;Reinicio de PipeLine
13 DSB
14 ISB
15 VLDR.F32 S0,=1.08381; x .PRIMER TERMINO.
16 BL sinCALC
17 STOP B STOP
18 sinCALC
19 VLDR.F32 S1,FACT3
20 VLDR.F32 S2,FACT5
21 VLDR.F32 S3,FACT7
22
23 VMUL.F32 S4,S0,S1 ; x * 1/3! = Xdiv3! = x/3!
24 VMUL.F32 S7,S0,S0 ; = x^2
25
26 VMUL.F32 S4,S4,S7 ; x/3! * x^2 = X3div3! = x^3/3! .SEGUNDO TERMINO.
27
28 VMUL.F32 S5,S0,S2 ; x * 1/5! = Xdiv5! = x/5!
29 VMUL.F32 S8,S7,S7 ; x^2 * x^2 = x^4
30 VMUL.F32 S5,S5,S8 ; x/5! * x^4 =x5div5! = x^5/5! .TERCER TERMINO.
31
32
33 VMUL.F32 S6,S0,S3 ; x * 1/7! =xdiv7! = x/7!
34 VMUL.F32 S9,S7,S8 ; x^2 *x^4 = x^6
35 VMUL.F32 S6,S6,S9 ; x/7!* x^6 = x7div7! = x^7/7! .CUARTO TERMINO.
36
37 VSUB.F32 S10,S0,S4; x - x^3/3! =Termino(1-2)= x - x^3/3!
38 VADD.F32 S10,S10,S5; (x - x^3/3!) + x^5/5! = TERM(1-2+3) = x - x^3/3! + x^5/5!
39 VSUB.F32 S0,S10,S6 ; (x - x^3/3! + x^5/5!) - x^7/7! ; TERM (1-2+3-4)
40 BX LR
41 FACT3 DCFS 0.166666666666 ;1/3!
42 FACT5 DCFS 0.008333333333 ;1/5!
43 FACT7 DCFS 0.00019841269 ;1/7!
44 ALIGN
45 END
46
47
```

Fuente: elaboración propia, empleando Snipping Tool.

4.20.3. Análisis de la Solución

Los valores 1/3!, 1/5! Y 1/7! se han almacenado como constantes y posteriormente cargados en los registros S1, S2 y S3 respectivamente. Como se muestra en la línea 26 de la figura 174 se ha encontrado el segundo, en la línea 30 se encuentra el tercero, en la línea 35 se encuentra el ultimo. Finalmente, en la línea 39 se almacena el resultado de todas las respuestas anteriores. El valor de $\sin(x=1,08381)$ se ha almacenado en el registro $S0=0x3F623CE2$ (0,88374 en sistema decimal) como se muestra en la figura 175.

Figura 175. Resultado del problema 18



Fuente: elaboración propia, empleando Snipping Tool.

CONCLUSIONES

1. Se logró realizar ejemplos claros en lenguaje ensamblador para el microcontrolador TM4C123GH6PM.
2. La cantidad de memoria disponible en los procesadores Cortex®-M depende plenamente del fabricante del procesador.
3. Se puede implementar dentro del lenguaje ensamblador operaciones complejas como seno y coseno mediante aproximaciones matemáticas.
4. Se logró explicar claramente cómo es posible guardar y recuperar datos en la memoria del procesador.

RECOMENDACIONES

1. Utilizar Keil uVision® 4 ya que trae preinstalados todos los paquetes a utilizar de forma nativa.
2. Guardar en carpetas distintas cada uno de los proyectos de Keil uVision®.
3. Inicializar correctamente el registro CPACR al momento de trabajar con números con punto flotante.
4. Comprender claramente el comportamiento de las banderas de registro xPSR
5. Tener a la mano el manual del microcontrolador TM4C123GH6PM para corroborar direcciones en la memoria, espacios para datos y código.

BIBLIOGRAFÍA

1. ACUÑA, Sergio. *uVision IDE*. [en línea]. <<https://seacunab.cl/uvision-ide/>>. [Consulta: 1 de abril de 2020].
2. ARM KEIL. *Condition Codes*. [en línea]. <https://www.keil.com/support/man/docs/armasm/armasm_dom1359731158738.html>. [Consulta: 13 de agosto de 2020].
3. _____. *DN and SN*. [en línea]. <https://www.keil.com/support/man/docs/armasm/armasm_pge1415643653321.html>. [Consulta: 13 de agosto de 2020].
4. _____. *Symbols, Literals, Expressions, and Operators*. [en línea]. <https://www.keil.com/support/man/docs/armasm/armasm_dom1359731172022.html>. [Consulta: 13 de agosto de 2020].
5. _____. *Writing ARM Assembly Language*. [en línea]. <https://www.keil.com/support/man/docs/armasm/armasm_dom1359731144635.html>. [Consulta: 13 de agosto de 2020].
6. ARM LTD. *ARM Instruction Set Manual*. [en línea]. <<https://iitd-plos.github.io/col718/ref/arm-instructionset.pdf>>. [Consulta : 15 de julio de 2020].

7. _____. *ARM @v7-M Architecture Reference Manual*. [en línea]. <https://web.eecs.umich.edu/~prabal/teaching/eecs373-f10/readings/ARMv7-M_ARM.pdf>. [Consulta : 15 de julio de 2020].
8. _____. *ARM Architecture Reference Manual Thumb-2 Supplement*. [en línea]. <<http://class.ece.iastate.edu/cpre288/resources/docs/Thumb-2SupplementReferenceManual.pdf>>. [Consulta : 15 de julio de 2020].
9. _____. *CMSIS-Core (Cortex-A)*. [en línea]. <https://www.keil.com/pack/doc/CMSIS/Core_A/html/group__CMSIS__CPSR.html>. [Consulta : 15 de enero de 2021].
10. HARRIS, David; HARRIS, Sarah. *Digital Design and Computer Architecture*. 2a ed. EE. UU.: Morgan Kaufmann. 2013. 720 p.
11. AZEIRA. *Conditional Execution*. [en línea]. <<https://azeria-labs.com/arm-conditional-execution-and-branching-part-6/>>. [Consulta: 1 de agosto de 2020].
12. *Calculadora conversor Decimal a hexadecimal*. [en línea]. <<https://www.calculadoraconversor.com/decimal-a-hexadecimal/>>. [Consulta: 20 de enero de 2021].
13. CCRS LAB. *Cortex-M4 Processor Overview with ARM Processors and Architecture*. [en línea]. <http://ccrs.hanyang.ac.kr/webpage_limdj/embedded/Cortex-M.pdf>. [Consulta : 1 de mayo de 2020].

14. CHAVEZ, Dennis. *Manual para citar y referenciar fuentes en textos de ingeniería* [en línea]. <https://repositorio.continental.edu.pe/bitstream/20.500.12394/6431/5/IV_UC_LI_Manual_para_citar_y_referenciar_fuentes_en_textos_2019.pdf>. [Consulta : 25 agosto de 2021].
15. CONSTANTINIDES, George. *Lecture 4 Assembly Language Programming Basics*. [en línea]. <<http://cas.ee.ic.ac.uk/people/gac1/Architecture/Lecture4.pdf>>. [Consulta : 1 de febrero de 2020].
16. _____. *Lecture 6 Stacks and Subroutines*. [en línea]. <<http://cas.ee.ic.ac.uk/people/gac1/Architecture/Lecture6.pdf>>. [Consulta : 1 de febrero de 2020].
17. GARCIA, Ignacio. *Cálculo Tema 4 Series de Taylor y MacLaurinv*. [en línea]. <<https://www.cartagena99.com/recursos/alumnos/apuntes/Tema%204.%20Series%20de%20Taylor%20y%20MacLaurin%20Ed.2.pdf>>. [Consulta: 1 de agosto de 2020].
18. GBTI, Israel. *Introduction to ARM Cortex-M Assembly Programming (FREE)*. [en línea]. <<https://www.udemy.com/course/arm-cortex-m-assembly-programming/>>. [Consulta: 13 de agosto de 2020].
19. IBM Corporation. *Tabla de conversión de valores ASCII, decimales, hexadecimales, octales y binarios*. [en línea]. <<https://www.ibm.com/docs/es/aix/7.1?topic=adapters-ascii-decimal-hexadecimal-octal-binary-conversion-table>>. [Consulta: 25 de septiembre de 2020].

20. JONES, Jeremy. *Arm Assembly Language*. [en línea]. <<https://www.scss.tcd.ie/Jeremy.Jones/CS1021/3%20ArmAssemblyLanguage.pdf>>. [Consulta: 2 de junio de 2020].
21. LEWIS, Daniel. *ARM® and Thumb®-2 Instruction Set Quick Reference Card*. [en línea]. <https://www.cse.scu.edu/~dlewis/book3/docs/ARM_and_Thumb-2_Instruction_Set.pdf>. [Consulta: 26 de agosto de 2020].
22. _____. *Cortex-M4F Instructions used in ARM Assembly for Embedded Applications*. [en línea]. <https://www.cse.scu.edu/~dlewis/book3/docs/ARM_Cortex-M4F_Instruction_Summary.pdf>. [Consulta: 26 de agosto de 2020].
23. _____. *Floating-Point Compares*. [en línea]. <<https://www.cse.scu.edu/~dlewis/book3/worksheets/9-FloatingPointCompares.pdf>>. [Consulta: 26 de agosto de 2020].
24. _____. *Thumb® 16-bit Instruction Set Quick Reference Card*. [en línea]. <<https://www.cse.scu.edu/~dlewis/book3/docs/Thumb%2016-bit%20Instruction%20Set.pdf>>. [Consulta: 26 de agosto de 2020].
25. _____. *Vector Floating Point Instruction Set Quick Reference Card*. [en línea]. <https://www.cse.scu.edu/~dlewis/book3/docs/Vector_Floating_Point_Instruction_Set.pdf>. [Consulta: 26 de agosto de 2020].

26. MARRACO, Mariano. *Desarrollo en serie de Taylor*. [en línea]. <http://recursostic.educacion.es/descartes/web/materiales_didacticos/Desarrollo_serie_taylor/Desarrollo_en_serie_de_taylor.html>. [Consulta: 1 de diciembre de 2020].
27. MARTIN, Matthew. *CISC vs RISC: Difference Between Architectures, Instruction Set*. [en línea]. <<https://www.guru99.com/risc-vs-cisc-differences.html>>. [Consulta: 3 de agosto de 2021].
28. MCDIARMID, Alisdair. *ARM immediate value encoding*. [en línea]. <<https://alisdair.mcdiarmid.org/arm-immediate-value-encoding/>>. [Consulta: 26 de agosto de 2020].
29. MDP Project. *Conditional Branch Instructions*. [en línea]. <http://www-mdp.eng.cam.ac.uk/web/library/enginfo/mdp_micro/lecture3/lecture3-3-3.html>. [Consulta: 22 de octubre de 2020].
30. MIT OpenCourseWare. *Taylor's Series of sin x*. [en línea]. <https://ocw.mit.edu/courses/mathematics/18-01sc-single-variable-calculus-fall-2010/unit-5-exploring-the-infinite/part-b-taylor-series/session-99-taylors-series-continued/MIT18_01SCF10_Ses99c.pdf>. [Consulta: 1 de febrero de 2020].
31. NELSON, Victor. *Arm Processor*. [en línea]. <https://www.eng.auburn.edu/~nelsovp/courses/elec5260_6260/slides/Chapter2%20ARM.pdf>. [Consulta: 18 de julio de 2020].

32. _____. *Embedded Computing Systems*. [en línea]. <https://www.eng.auburn.edu/~nelsovp/courses/elec5260_6260/slides/Chapter1a%20Embedded%20System%20Intro.pdf>. [Consulta: 18 de julio de 2020].
33. SERRA, Bernat. *Área de un triángulo*. [en línea]. <<https://www.universoformulas.com/matematicas/geometria/area-triangulo/>>. [Consulta: 11 de octubre de 2020].
34. SILICON LABS. *Which ARM Cortex Core Is Right for Your Application: A, R or M?* [en línea]. <<https://www.silabs.com/documents/public/white-papers/Which-ARM-Cortex-Core-Is-Right-for-Your-Application.pdf>>. [Consulta: 1 de junio de 2020].
35. STACK OVERFLOW. *What are callee and caller saved registers?* [en línea]. <<https://stackoverflow.com/questions/9268586/what-are-callee-and-caller-saved-registers>>. [Consulta: 1 de noviembre de 2020].
36. STANFORD. *Risc Architecture. Risc vs. Cisc*. [en línea]. <<https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/riscisc/>>. [Consulta : 11 de febrero de 2020].
37. STREAMS, David. *Conversor de Decimal - Binario - Octal – Hexadecimal*. [en línea]. <<https://www.davidstreams.com/baul/conversor/>>. [Consulta : 18 de enero de 2020].

38. Texas Instrument. *Cortex-M3/M4F Instruction Set*. [en línea]. <http://users.ece.utexas.edu/~valvano/EE345L/Labs/Fall2011/CortexM_InstructionSet.pdf>. [Consulta: 13 de noviembre de 2020].
39. Tutorialspoint. *Assembly - Addressing Modes*. [en línea]. <https://www.tutorialspoint.com/assembly_programming/assembly_addressing_modes.html>. [Consulta : 11 de febrero de 2020].
40. VALVANO, Jonathan. *Introduction to ARM® Cortex®-M Microcontrollers*. 5a ed. EE. UU.: Jonathan W. Valvano, 2019. 507 p.
41. VICKERY, Christopher. *IEEE-754 Floating-Point Conversion*. [en línea]. <<https://babbage.cs.qc.cuny.edu/IEEE-754.old/Decimal.html>>. [Consulta : 20 de agosto de 2020].
42. WICKERT, Mark. *The Cortex-M Series: Hardware and Software*. [en línea] <http://ece.uccs.edu/~mwickert/ece5655/lecture_notes/ARM/ece5655_chap2.pdf>. [Consulta : 1 de febrero de 2020].
43. YIU, Joseph. *The Definitive Guide to ARM Cortex®-M3 and Cortex®-M4 Processor*. 3a ed. United Kingdom: Elseiver, 2013. 864 p.

