

UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de
Ingeniería y Sistemas de Telecomunicación



PROYECTO FIN DE GRADO

MODELO SIMULABLE EN PYTHON DE UN
PROCESADOR ARM CORTEX-M0 PARA
SU APLICACIÓN EN CODERUNNER

CÉSAR ESTEBARANZ CARRETERO

Grado en Ingeniería Electrónica de Comunicaciones
Junio 2024

MODELO SIMULABLE EN PYTHON DE UN PROCESADOR
ARM CORTEX-M0PARA SU APLICACIÓN EN CODERUNNER
CÉSAR ESTEBARANZ CARRETERO

JUNIO
2024

PROYECTO FIN DE GRADO

TÍTULO: Modelo simulable en Python de un procesador ARM Cortex-M0 para su aplicación en CodeRunner

AUTOR/A: César Estebaranz Carretero

TITULACIÓN: Grado en Ingeniería Electrónica de Comunicaciones

TUTOR/A: Manuel César Rodríguez Lacruz

DEPARTAMENTO: Departamento de Ingeniería Telemática y Electrónica

VºBº TUTOR/A

Miembros del Tribunal Calificador:

PRESIDENTE/A: Marta Sánchez Agudo

TUTOR/A: Manuel César Rodríguez Lacruz

SECRETARIO/A: José Antonio Herrera Camacho

Fecha de lectura:

Calificación:

El Secretario/La Secretaria,

Resumen

La educación en línea o a distancia, impulsada por las nuevas tecnologías, ha revolucionado la forma en que las personas acceden y adquieren conocimientos. Este cambio se refleja en la eliminación de barreras geográficas y temporales, permitiendo a estudiantes de todo el mundo participar en cursos de instituciones de otras regiones sin necesidad de trasladarse físicamente. Además, la educación en línea ha mejorado la experiencia de aprendizaje al incorporar recursos multimedia, como videos y simulaciones interactivas, que facilitan la comprensión de conceptos.

En este contexto, la aplicación web Moodle se destaca como una plataforma utilizada por la Universidad Politécnica de Madrid. Moodle cumple con requisitos fundamentales, como accesibilidad, flexibilidad, interactividad y adaptabilidad. Los docentes pueden aprovechar sus funciones, como cuestionarios en línea, asignación de tareas, foros de discusión, encuestas y un sistema de calificación integrado. Moodle también permite la personalización a través de complementos, brindando a los usuarios la posibilidad de adaptar la aplicación a sus necesidades.

El proyecto se centra en la utilización de cuestionarios en Moodle, destacando la importancia de la herramienta CodeRunner. Este plugin permite la evaluación automatizada de habilidades en programación y desarrollo informático, ofreciendo un entorno de evaluación de código en línea. Los profesores pueden definir casos de prueba, restricciones de tiempo y criterios de evaluación específicos para cada pregunta, proporcionando una evaluación detallada y personalizada. CodeRunner soporta varios lenguajes de programación, como Python, C++, Java y más.

La problemática surge al limitar su uso a lenguajes específicos, pero el proyecto propone la ampliación de CodeRunner para evaluar cualquier tipo de pregunta. En el contexto de la asignatura de Microprocesadores en la Universidad Politécnica de Madrid, se busca crear un tipo de pregunta que evalúe los conocimientos de los estudiantes sobre el lenguaje de ensamblaje en un microcontrolador ARM Cortex M0.

En conclusión, la educación en línea ha experimentado transformaciones significativas gracias a las nuevas tecnologías y plataformas como Moodle. El proyecto propuesto se enfoca en mejorar la evaluación de conocimientos en programación mediante el uso de CodeRunner, ampliando su capacidad para abordar un nuevo lenguaje y adaptándolo a las necesidades específicas.

Abstract

Online or distance education, driven by new technologies, has revolutionized how people access and acquire knowledge. This shift is evident in the removal of geographical and temporal barriers, allowing students worldwide to participate in courses from institutions in other regions without the need for physical relocation. Furthermore, online education has enhanced the learning experience by incorporating multimedia resources, such as videos and interactive simulations, which facilitate the understanding of concepts.

In this context, the web application Moodle stands out as a platform used by the Universidad Politécnica de Madrid. Moodle meets fundamental requirements such as accessibility, flexibility, interactivity, and adaptability. Educators can leverage its features, including online quizzes, assignment allocation, discussion forums, surveys, and an integrated grading system. Moodle also allows customization through plugins, giving users the ability to tailor the application to their needs.

The project focuses on the use of quizzes in Moodle, emphasizing the importance of the CodeRunner tool. This plugin enables the automated assessment of programming skills and computer development, providing an online code evaluation environment. Teachers can define test cases, time constraints, and specific evaluation criteria for each question, offering a detailed and personalized assessment. CodeRunner supports various programming languages, such as Python, C++, Java, and more.

The challenge arises when restricting its use to specific languages, but the project proposes expanding CodeRunner to evaluate any type of question. In the context of the Microprocessors course at the Universidad Politécnica de Madrid, the goal is to create a question type that assesses students' knowledge of assembly language in an ARM Cortex M0 microcontroller.

In conclusion, online education has undergone significant transformations thanks to new technologies and platforms like Moodle. The proposed project focuses on enhancing the evaluation of programming knowledge through the use of CodeRunner, expanding its capability to handle a new language and adapting it to specific needs.

Índice de contenidos

Resumen	i
Abstract	iii
Índice de contenidos.....	v
Índice de figuras.....	vii
Índice de tablas.....	ix
Lista de acrónimos	xi
1 Introducción, antecedentes y objetivos	1
1.1 Introducción	1
1.2 Antecedentes	2
1.3 Objetivos	3
2 Antecedentes o “marco tecnológico”	5
2.1 CodeRunner.....	5
2.2 Servidor JOBE	5
3 Especificaciones y restricciones de diseño	9
4 Descripción de la solución propuesta	13
4.1 Condiciones iniciales	13
4.1.1 Python.....	13
4.1.2 ARM Cortex M0.....	13
4.2 Planteamiento	16
4.3 Diseño de <i>software</i>	17
4.4 Clases	17
4.4.1 Preprocesador.....	18
4.4.2 Unidad Central de Procesamiento.....	19
4.4.3 Unidad Aritmeticológica.....	22
4.4.4 Unidad de Control	22
4.4.5 Registros	23
4.4.6 Memoria	23
4.4.7 Utilidades	23
4.4.8 Excepciones:.....	24
5 Presupuesto	25
6 Impacto del proyecto	27

7	Conclusiones	29
8	Referencias	31
9	Anexos	33
A	Manual de usuario.....	35
A.1	¿Qué es este simulador?.....	35
A.2	Guía rápida	35
A.3	Ejemplo.....	41
B	Documentación principal	47
B.1	Módulo preprocess.....	47
B.2	Módulo armsim	49

Índice de figuras

Fig. 1. Marco tecnológico inicial.....	6
Fig. 2. ¿Dónde encaja el proyecto?	7
Fig. 3. Solución 1. Modificando el servidor JOBE.....	9
Fig. 4. Solución 2. Utilizar Moodle y CodeRunner.	10
Fig. 5. Diagrama de bloques de ARM Cortex M0.	15
Fig. 6. Arquitectura de von Neumann.	16
Fig. 7. Modelo simplificado UML del simulador.	17
Fig. 8. Ejemplo devolución de memoria.	21
Fig. 9. Ejemplo devolución registros.	21
Fig. 10. Ejemplo devolución instrucciones procesadas.	22
Fig. 11. Plantilla genérica para el prototipo de pregunta.	36
Fig. 12. Información para la creación del prototipo de pregunta.....	36
Fig. 13. El simulador como un recurso del curso.....	36
Fig. 14. El simulador es parte de los recursos del curso.....	37

Índice de tablas

Tabla 1. Características ARM Cortex M0.....	14
Tabla 2. Orden normal frente orden natural.....	23

Lista de acrónimos

ALU

Arithmetic Logic Unit, 19

API

Application Programming Interface,
18, 24

CPU

Central Processing Unit, 38, 45

JOBE

JOB Engine, v, vii, 5, 6, 7, 9, 10, 11,
29, 31

UML

Unified Modeling Language, 17

UPM

Universidad Politécnica de Madrid,
25

1 Introducción, antecedentes y objetivos

1.1 Introducción

Las nuevas tecnologías han traído cambios significativos en la educación, ha transformado la forma en que las personas acceden y adquieren conocimientos; es la denominada educación en línea o a distancia. Esta transformación se ve plasmada en varios aspectos, uno de ellos es el acceso universal a la educación para personas de todo el mundo, así, estudiantes de diferentes lugares pueden cursar asignaturas de instituciones en otras regiones del planeta sin necesidad de trasladarse físicamente. Otro aspecto es que la educación en línea ha eliminado las barreras temporales, permitiendo que los estudiantes accedan a materiales educativos y participen en clases en cualquier momento, facilitando la compatibilización de los estudios con otras responsabilidades.

Dadas estas mejoras se necesitan plataformas digitales que hagan que todo esto sea posible y, además, se ganan otras mejoras considerables. Estas plataformas ya no son soportes físicos con información estática, las nuevas tecnologías han enriquecido los recursos de aprendizaje con contenido multimedia, como videos, simulaciones interactivas, presentaciones multimedia, entre otros. Esto mejora la experiencia de aprendizaje y ayuda a los estudiantes a comprender conceptos de manera más efectiva. A los alumnos en estas nuevas plataformas educativas en línea se les ofrecen herramientas interactivas que fomentan su participación como estudiante, de modo que, se pueden incluir foros de discusión, sesiones de chat en vivo, evaluaciones en línea y actividades colaborativas, que enriquecen la interacción entre estudiantes y con los profesores.

Otro elemento importante en la educación en línea es el modo en que se accede a la información, ya que el acceso a través de dispositivos móviles ha facilitado todo aún más y los estudiantes pueden participar en clases, acceder a materiales y realizar tareas desde sus *smartphones* o *tablets*, brindando la posibilidad de llevar todo el material con ellos en cualquier momento.

A la vez que aumentan la flexibilidad, los docentes pueden expandir la adaptación y personalización de los materiales educativos de acuerdo con las necesidades únicas de cada alumno mediante sistemas de aprendizaje adaptativo. Estos sistemas utilizan algoritmos para personalizar el contenido en base al progreso y habilidades individuales de cada estudiante. Muchas plataformas educativas recolectan grandes cantidades de datos sobre el desempeño, acceso a los contenidos y patrones de aprendizaje de los alumnos para ajustar los cursos a cada uno.

En resumen, las nuevas tecnologías han revolucionado la educación en línea al proporcionar accesibilidad, flexibilidad, interactividad y adaptabilidad, mejorando así la calidad y la experiencia de aprendizaje para los estudiantes.

1.2 Antecedentes

Aunque existen una gran cantidad de plataformas de educación en línea y se podría elegir cualquiera para dar una solución a este aspecto educativo, este proyecto, se va a basar en la aplicación web Moodle. Esta elección viene dada por ser la plataforma utilizada como aula virtual por la Universidad Politécnica de Madrid.

Moodle cumple con los requisitos de accesibilidad, flexibilidad, interactividad y adaptabilidad de las plataformas de aprendizaje en línea. Y, aunque desde que se establece la plataforma ya se tiene acceso a una serie de complementos para explotar estas características; la propia plataforma brinda un entorno en el cual los usuarios pueden aumentar la personalización de la aplicación a través de complementos.

Las funciones extra que aportan estos complementos pueden ir desde juegos visuales hasta la posibilidad de añadir la creación de certificados de los cursos realizados, pero también existen una buena cantidad de complementos para mejorar la evolución de los alumnos. Inclusive, la documentación de Moodle explica cómo se pueden crear complementos propios. Los complementos son también llamados extensiones o, en inglés, *addons* o *plugins*. Todo está pensado en pro de la personalización para optimizar el aprendizaje.

Enfocando la utilización de Moodle en el personal docente podemos enumerar varios aspectos en los cuales destaca esta plataforma:

1. **Cuestionarios:** Los profesores pueden crear cuestionarios en línea con preguntas de opción múltiple, verdadero/falso, respuesta corta, ensayos, entre otras.
2. **Tareas:** Los profesores pueden asignar tareas en línea donde los estudiantes pueden cargar documentos, presentaciones u otros trabajos.
3. **Foros de Discusión:** Los foros permiten la participación de los estudiantes en discusiones en línea.
4. **Encuestas:** Se pueden crear encuestas para recopilar información y opiniones de los estudiantes.
5. **Calificaciones y Libro de Calificaciones:** Moodle cuenta con un sistema de calificación integrado que permite a los profesores asignar calificaciones y sirve como una visión general del rendimiento de cada estudiante.
6. **Repositorio de Archivos:** Los profesores pueden alimentar el curso con material multimedia para complementar el curso.

En todas estas características el profesor también puede participar activamente si lo desea y así convertir el curso en algo bidireccional como en la educación clásica. Así, los profesores pueden llevar a cabo evaluaciones, hacer comentarios en actividades, o adaptarse a diferentes estilos de aprendizaje y tipos de evaluación. Ya sea individual o en grupo.

1.3 Objetivos

Uno de los puntos fuertes de Moodle es la elaboración de cuestionarios, los cuales tienen varias características, las más importantes serían diversos tipos de pregunta, aleatorización de las preguntas, tiempo y fechas de entrega, retroalimentación automática por parte de los profesores, medidas de seguridad para proteger la integridad, integración con el libro de calificaciones e informar a los profesores.

Existen muchos complementos para Moodle que permiten la creación de tipos de pregunta que no están por defecto implementados en el sistema. Uno de esos *plugins* es CodeRunner, el cual proporciona un entorno de evaluación de código en línea y fue diseñado para facilitar la evaluación automatizada de las destrezas en programación y el desarrollo de habilidades informáticas, es decir, CodeRunner permite a los profesores crear preguntas que requieren que los estudiantes escriban código en diferentes lenguajes.

Los profesores pueden definir casos de prueba para verificar la corrección y eficiencia del código del estudiante, restricciones de tiempo y criterios de evaluación específicos para cada pregunta, ofreciendo una evaluación detallada y personalizada. CodeRunner también incluye características de seguridad, como la ejecución en entornos aislados, para garantizar la integridad del sistema.

Los lenguajes soportados por CodeRunner son Python2, Python3, C, C++, Java, PHP, Pascal, JavaScript, Octave y Matlab. Siempre que se utilice uno de estos lenguajes el *plugin* se comportará adecuadamente y dará solución a su uso; y es precisamente aquí donde también comienza su problemática, si se desea evaluar en cualquier otro lenguaje no sirve. O sí, ya que haciendo uso de su capacidad de ejecutar estos lenguajes de programación como un subproceso del sistema se puede compilar el nuevo lenguaje (si fuese necesario) y después evaluar el código en el nuevo lenguaje de programación. Así se puede ampliar CodeRunner para crear casi cualquier tipo de pregunta.

Y es precisamente este el escenario en cual se va a desarrollar este proyecto. En la Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación de la Universidad Politécnica de Madrid se imparte en el 4º semestre la asignatura llamada Microprocesadores y como dice la guía de aprendizaje “La asignatura está basada en un microprocesador de 32 Bits (ARM CortexM). Está enfocada al diseño de aplicaciones con microprocesador y con un enfoque eminentemente práctico, donde el alumno desarrolla durante el curso el diseño de un sistema basado en microprocesador. Se utiliza este diseño para ir introduciendo los diferentes conceptos teóricos necesarios para el desarrollo de este tipo de aplicaciones.” [1]. Por lo tanto, para evaluar correctamente al alumno se quiere crear un tipo de pregunta que sirva para dimensionar los conocimientos del alumnado sobre el lenguaje de ensamble en un microcontrolador ARM Cortex M0 dentro de esta asignatura.

2 Antecedentes o “marco tecnológico”

Ante de comenzar con la explicación de como se ha planteado dar solución al problema de cómo desarrollar un intérprete de lenguaje de ensamble para un microcontrolador ARM Cortex M0, es preciso explicar cuál es el marco tecnológico sobre el cual se va a desenvolver el proyecto.

2.1 CodeRunner

Según su propia página de documentación [2], CodeRunner es un complemento para Moodle desarrollado por la Universidad de Canterbury y se utiliza en ella desde hace más de diez años y ha realizado millones de cuestionarios en Python, C, JavaScript, PHP, Octave y Matlab. Se utiliza en trabajos de laboratorio, tareas, pruebas y exámenes de múltiples cursos. En los últimos años CodeRunner se ha extendido por todo el mundo y en enero de 2021 está instalado en más de 1800 sitios Moodle en todo el mundo.

CodeRunner soporta los siguientes lenguajes: Python2 (considerado obsoleto), Python3, C, C++, Java, PHP, Pascal, JavaScript (NodeJS), Octave y Matlab.

En los últimos tiempos CodeRunner, y gracias a la capacidad de ejecutar código, está experimentando con la posibilidad que a través de inteligencia artificial sea capaz de evaluar preguntas con respuestas con textos de distinta longitud dadas por el alumno.

CodeRunner puede utilizarse con seguridad en un servidor web con Moodle, y para realizar la ejecución de código es necesario un servidor para dicho propósito, se llama servidor JOBE.

2.2 Servidor JOBE

JOBE (acrónimo de *Job Engine*) [3] es un servidor que da soporte para la realización de pequeños trabajos de compilación y ejecución en la variedad de lenguajes de programación usados por CodeRunner. Por lo tanto, es en la máquina que corre este servidor donde se deben tener instalados los intérpretes y compiladores de los diferentes lenguajes de programación.

Este nuevo servidor se recomienda, por razones de seguridad, que esté instalado en una máquina separada; así el código se ejecuta de manera aislada y se mantiene fuera de los posibles riesgos que pudiese suponer para el servidor principal, lo que se conoce como *sandbox*.

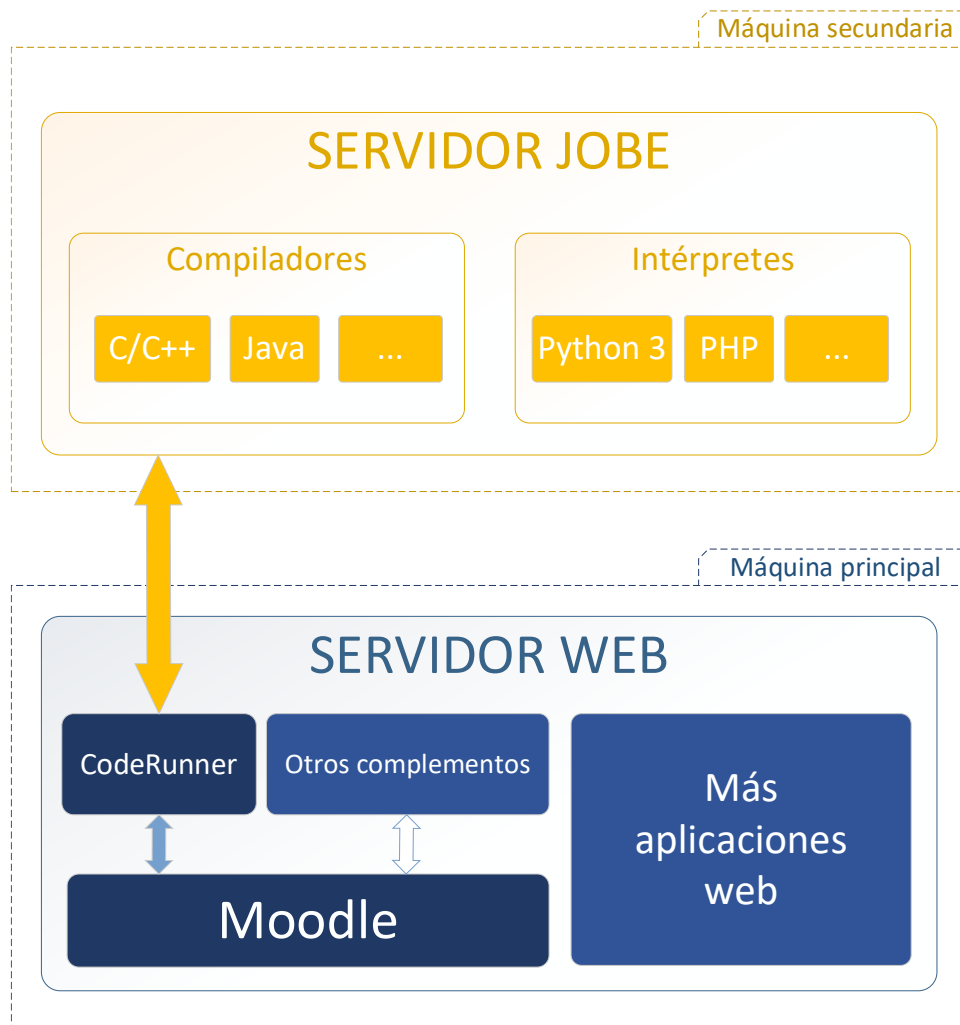


Fig. 1. Marco tecnológico inicial.

En la Fig. 1. Se puede observar cómo queda implementado el marco inicial sobre el cual se va a desarrollar el proyecto. Aquí se aprecia como son necesarios dos servidores, por un lado, el servidor web el cual tendrá instalada la aplicación web de Moodle junto con todos sus complementos, incluido CodeRunner. En este servidor web pueden, además, estar instaladas más aplicaciones o alojar sitios web. Junto a este servidor es necesario un servidor JOBE el cual se encargará de realizar las tareas de compilación y ejecución de la variedad de lenguajes de programación usados por CodeRunner. Otra cosa apreciable es que CodeRunner es el único que mantiene contacto con este otro servidor, siendo transparente para el resto del servidor web, de Moodle y de los otros *plugins*.

Aquí comienza la primera condición en el diseño de la solución de este proyecto y es que se debe encajar en esta estructura el lugar donde se situará el programa que evaluará el código de los alumnos en lenguaje de ensamble.

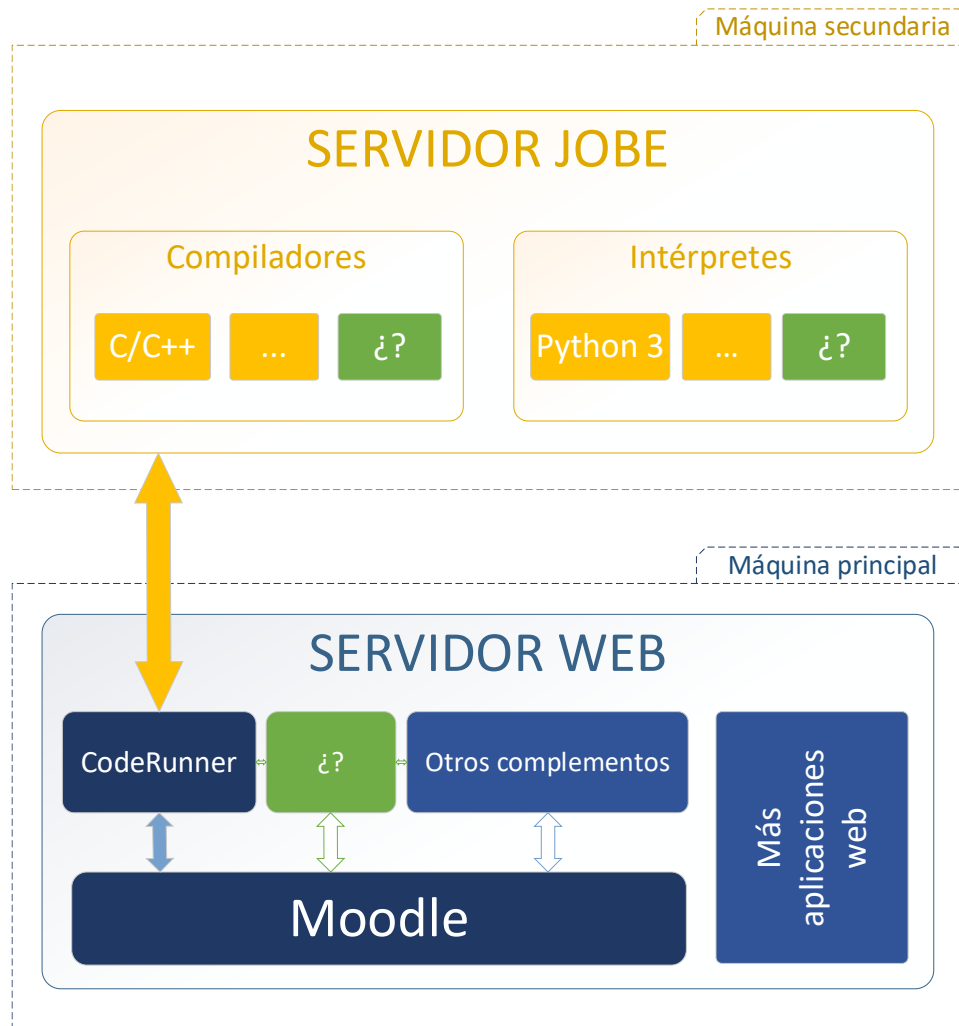


Fig. 2. ¿Dónde encaja el proyecto?

Consecuentemente, existen dos opciones dónde puede incrustarse el proyecto, una opción en cada servidor como se ilustra en la se Fig. 2. La primera posibilidad es crear un compilador o interprete de código en el servidor JOBE, y la segunda opción sería crear una pieza de rompecabezas que encaje con Moodle, CodeRunner y todo lo demás del servidor web.

Esta es la primera elección que se debe tener en cuenta ya que marcará el devenir del proyecto. A lo largo del siguiente punto se irá dando solución a esta y otras cuestiones como parte de las restricciones para diseñar una solución adecuada.

3 Especificaciones y restricciones de diseño

Se debe dar una primera solución al diseño del compilador o intérprete del lenguaje de ensamble para saber en qué lugar colocarlo e ir acotando sus especificaciones técnicas.

Existe una primera solución que sería el añadir el intérprete o compilador al servidor JOBE; esto puede considerarse un riesgo, puesto que hay que modificar ciertos ficheros en el servidor y añadir nuevos. Todo esto puede suponer que se produzcan errores y el servidor deje de funcionar y, además, en el momento que se actualice el JOBE es muy probable que los archivos modificados vuelvan a un estado previo o se reescriban perdiendo el trabajo realizado y volviendo a comenzar el proceso.

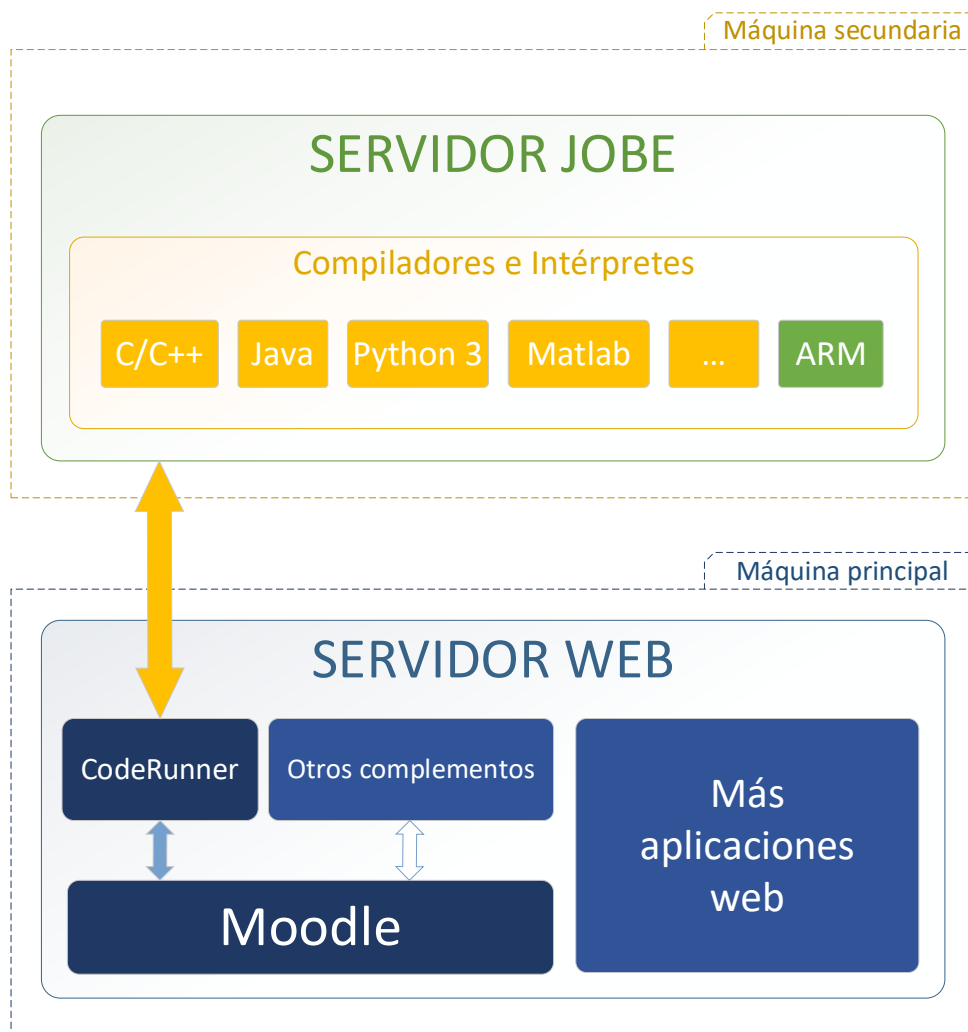


Fig. 3. Solución 1. Modificando el servidor JOBE.

En la Fig. 3 puede verse una representación de esta solución, en verde se marca todo lo que implica. Por un lado, modificar el servidor JOBE y por otro añadir el compilador/intérprete. Esta solución también tiene beneficios, Moodle, o más

concretamente CodeRunner, mostraría el lenguaje nuevo de una manera nativa. Esto haría para el usuario final (docente) todo mucho más transparente y fácil de utilizar.

La segunda solución, y la recomendada por la propia documentación de CodeRunner, es utilizar uno de sus lenguajes base [2] y crear un programa convirtiéndolo en un prototipo de pregunta que use la respuesta dada por el alumno para interpretarla y pasarle al servidor JOBE las instrucciones de qué hacer con él. Para el usuario es más tedioso, ya que cada vez que implemente un tipo de pregunta nueva debe crear un programa y en el supuesto de añadir archivos adjuntos estos deben subirse cada pregunta del tipo nuevo. A pesar de todo este trabajo el beneficio es claro, el servidor JOBE se mantendrá intacto.

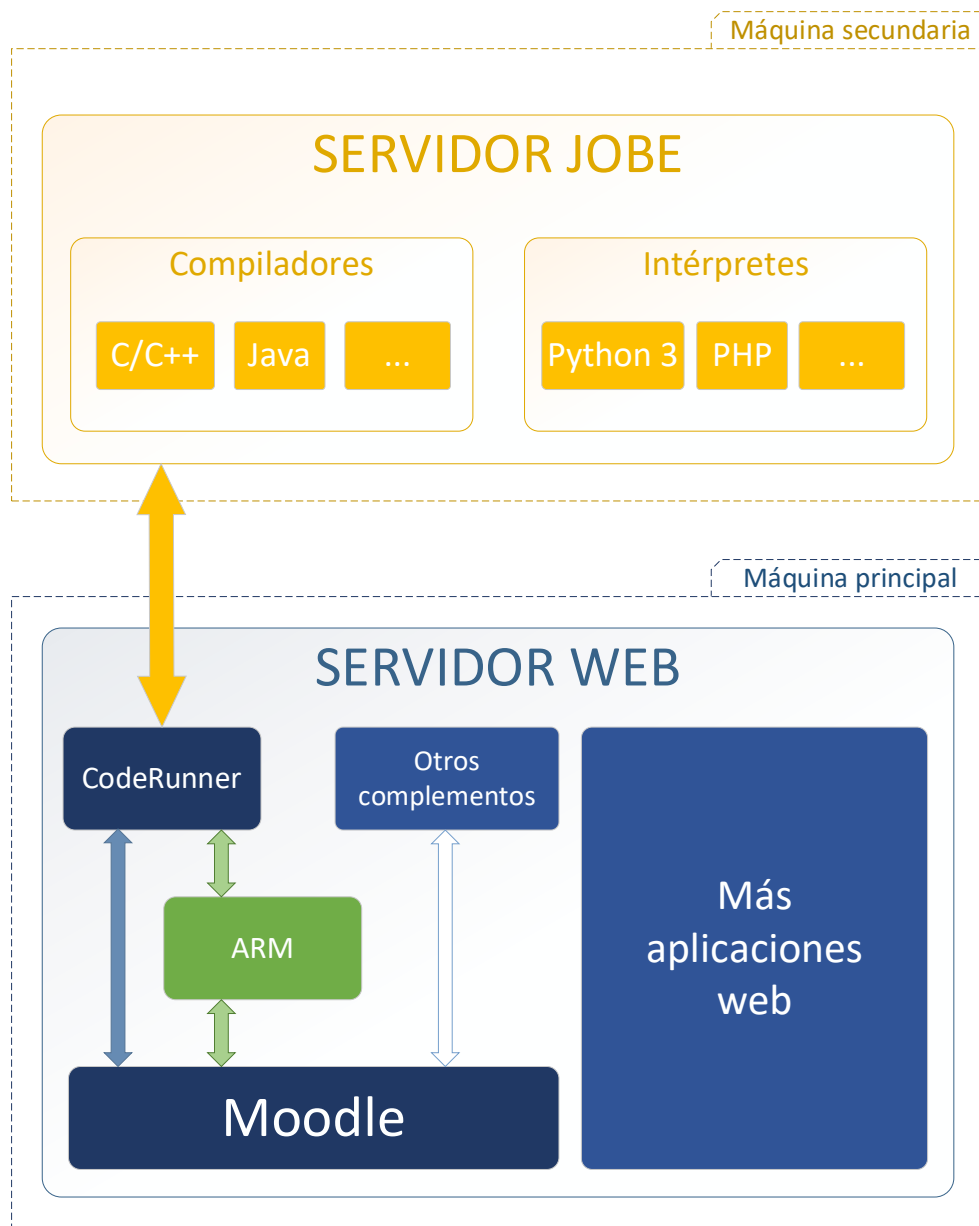


Fig. 4. Solución 2. Utilizar Moodle y CodeRunner.

En la Fig. 4 vemos esta solución marcada en verde y, como se mencionaba, la segunda opción es la propuesta en la documentación de CodeRunner y será la utilizada en este proyecto, el intérprete o compilador ahora ya conocemos que deberá estar en el servidor web junto a Moodle y haciendo uso del propio *plugin* de CodeRunner.

La primera restricción de diseño ha quedado resuelta y a su vez conlleva a una restricción, utilizar CodeRunner implica elegir un lenguaje de los nativos para crear la solución del proyecto; sólo están disponibles unos pocos lenguajes, por lo que, el intérprete o compilador del lenguaje de ensamble para el microcontrolador ARM Cortex M0 deberá estar creado en uno de ellos. Aunque se puede abrir un debate en torno a la elección de lenguaje más apropiado, desde la propuesta del tema del proyecto, ya se especificaba el uso del lenguaje Python en su versión 3 y, además, la elección de Python también es la recomendación de la documentación de CodeRunner.

Teniendo en cuenta esto se puede asumir que, para dar la solución correcta, lo que se va a crear es un intérprete o una especie de simulador; que cogerá el código introducido por el alumno y lo desmenuzará para recrear que es lo que ocurriría en un microcontrolador ARM Cortex M0. Esto lo realizará desde un cuestionario y a través de un nuevo tipo de pregunta basada en el *plugin* de CodeRunner.

Concretando, se creará un simulador del lenguaje de ensamble para el microcontrolador ARM Cortex M0, este irá integrado en Moodle haciendo uso del *plugin* CodeRunner y se utilizará el lenguaje Python en su versión 3 para su desarrollo.

Para finalizar, es importante tener en cuenta que información debe devolver el intérprete del código en lenguaje de ensamble. Esta información básicamente será devolver los valores de registros, estado de la ALU, estado de la Unidad de Control y valores guardados en la memoria del modo que lo haría el microcontrolador ARM Cortex M0 en la vida real. Por supuesto, ya que es un intérprete programado, se añadirá información extra tanto a la entrada del microcontrolador como a la salida de este para facilitar la creación y evaluación de casos prácticos. A la entrada se podrá especificar al simulador cuáles son sus condiciones iniciales en la Unidad Aritmeticológica, Registros, valores de Memoria y un valor de tiempo límite para la ejecución del código de modo que evitaremos dejar al simulador en un bucle infinito y por lo tanto bloquear un subproceso de manera indefinida en el servidor JOBE y disminuyendo su rendimiento. Para la salida, al ser información que deberá ser interpretada se dará la información anteriormente mencionada de los valores de registros, estado de la ALU, estado de la Unidad de Control y valores guardados en la memoria, pero se ha pensado en la generación una información extra que de una capacidad extrema y precisa para evaluar al alumnado de la manera más optima. Esta información extra consiste en contabilizar las instrucciones que ha realizado el simulador y se le pueda pedir información de si el código del alumno ha cumplido una serie de condiciones de uso y responda con verdadero o falso.

En el anexo A, Manual de usuario., se explica de manera práctica como se implementa la solución y de qué modo el docente debe crear el prototipo de pregunta y la pregunta en si.

4 Descripción de la solución propuesta

4.1 Condiciones iniciales

Haciendo un resumen del apartado anterior, las principales restricciones del diseño son, el uso del lenguaje Python en su versión 3 para la creación de un programa que interprete el código en lenguaje de ensamble del ARM Cortex M0 y devuelvan los resultados de valores de registros, estados y memoria del microprocesador al finalizar, junto con información útil para la evaluación.

A continuación, se va a exponer las cualidades que tiene el lenguaje Python en su versión 3 y desarrollar la información sobre el *hardware* de un microcontrolador ARM Cortex M0.

4.1.1 Python

Como se ha comentado anteriormente, desde la propuesta del tema se especifica el uso de Python [4]. Ha día de hoy existen coetáneamente, las versiones 2 y 3. Ya desde la documentación de CodeRunner se plantea la opción de la versión 2 como obsoleta [5], por lo que la versión base de Python será la versión 3. Este lenguaje aporta varias cualidades interesantes y se enumeran a continuación [6]:

- Lenguaje de propósito general.
- Es multiplataforma.
- Es un lenguaje interpretado.
- Es multiparadigma.
- Es fuertemente tipado.
- Es de tipado dinámico.
- Es orientado a objetos.
- El formato del código es estructural.
- Permite la extensión a través de librerías.

Al ser un lenguaje interpretado el código estará siempre visible para el usuario del simulador y al ser orientado a objetos nos permite crear una estructura.

4.1.2 ARM Cortex M0

El microcontrolador sobre el cual se va a trabajar es este, el modelo Cortex en su versión M0 de la marca ARM. La documentación de este *hardware* será la base por donde comenzar a desarrolla [7].

La traducción de la presentación de la propia página web del fabricante dice:

“El procesador ARM Cortex-M0 es uno de los procesadores ARM más pequeños disponibles.

El Cortex-M0 tiene un área de silicio excepcionalmente pequeña, un bajo consumo de energía y un número de líneas (huella) de código mínima, lo que permite a los desarrolladores conseguir un rendimiento de 32 bits a un precio de 8 bits, evitando el paso por dispositivos de 16 bits. El número de puertas ultra bajo del procesador permite su implantación en dispositivos analógicos y de señal mixta.”

También se muestra una tabla con las especificaciones:

Tabla 1. Características ARM Cortex M0.

Arquitectura	Armv6M
Interfaz de bus	AHBLite, arquitectura de bus Von Neumann
Soporte ISA	Subconjunto Thumb/Thumb2
Pipeline	De 3 etapas
Manipulación de bits	La región de bandas de bits se puede implementar con Corstone Foundation IP
Interrupciones	Interrupción no enmascarable (NMI) + 1 a 32 interrupciones físicas
Controlador de Interrupciones Wakeup	Opcional
Instrucciones mejoradas	Opción de multiplicación de un ciclo por hardware (32x32)
Modos de reposo	Instrucciones WFI y WFE integradas y capacidad de Sleep On Exit Señales de reposo y sueño profundo Modo de retención opcional con el kit de gestión de energía de ARM
Depuración	Puertos de depuración JTAG y Serial Wire opcionales. Hasta 4 puntos de interrupción y 2 puntos de vigilancia
Paquete de referencia o ejemplo de sistema	Corstone101

Y un diagrama de bloques:

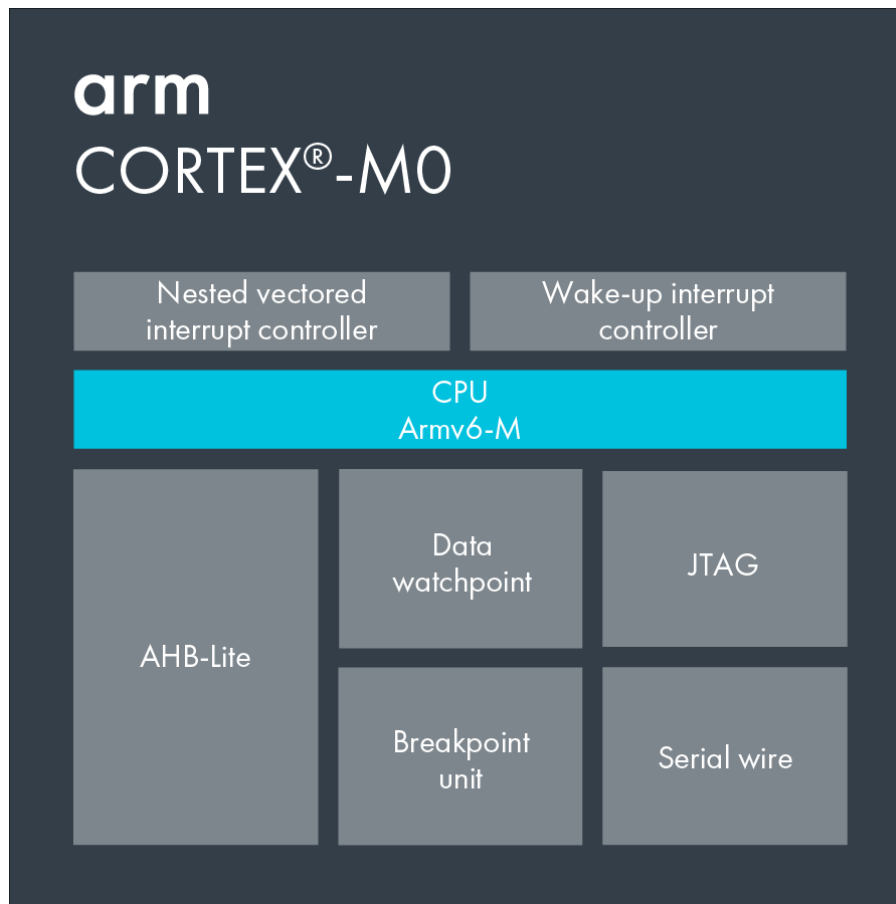


Fig. 5. Diagrama de bloques de ARM Cortex M0.

Como se comentaba todo esto es un extracto de la propia documentación de ARM. En este repositorio existe un manual técnico el cual explica toda la información del microcontrolador en cuestión. Este manual técnico [8] es la base de todo el desarrollo realizado, se muestran desde la organización de memoria hasta las instrucciones admitidas por el dispositivo.

4.2 Planteamiento

Basándose en la tabla del fabricante se observa que está basado en la arquitectura de von Neumann (Fig. 4),

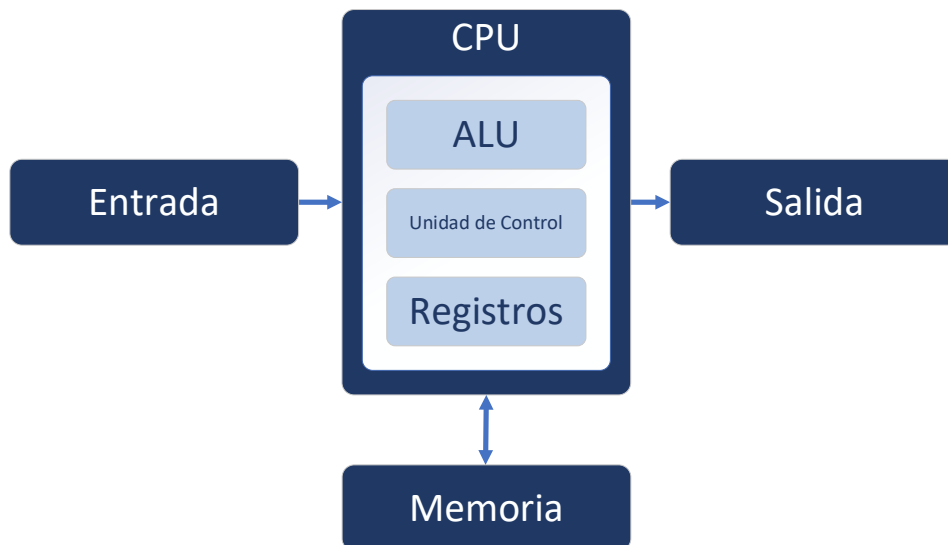


Fig. 6. Arquitectura de von Neumann.

La Fig. 6 muestra varios bloques y conociendo la característica de Python de ser un lenguaje orientado a objetos, cada uno de estos bloques se convertirán en clases para el simulador. De aquí se obtienen las clases:

- Unidad Central de Procesamiento
 - Unidad Aritmeticológica
 - Unidad de Control
 - Registros
- Memoria

Si el lector compara la arquitectura de von Neumann de la Fig. 6 con las clases resultantes mostradas en la Fig. 7, se puede observar claramente, que falta la representación de la entrada y la salida. Esto tiene una explicación, al ser un simulador del código en el dispositivo, no se podrá interactuar dinámicamente con las entradas y las salidas. Y se puede ir un paso más allá, no se podrá interactuar con el *hardware* en general; por lo que, las interrupciones no tendrán sentido en el proyecto.

A estas clases representativas, se les deberán añadir clases para mejorar su manejo. Estas clases complementarias deberán funcionar a modo de *framework* en el proyecto. Básicamente se añadirán una clase para el manejo de representación numérica y otras utilidades, otra para el manejo de excepciones, y una de preprocesador de código. Dichas clases no son más que pasos que haría cualquier entorno de trabajo de

cualquier lenguaje, como el alumno introducirá un texto plano, es el propio simulador el encargado de realizar estas tareas.

Las clases quedarían así:

- Unidad Central de Procesamiento
 - Unidad Aritmeticológica
 - Unidad de Control
 - Registros
- Memoria
- Utilidades
- Excepciones:
- Preprocesador

4.3 Diseño de *software*

En este apartado se mostrará en la Fig. 7 un modelado UML simplificado de las clases básicas del microprocesador; expresando la modularidad, abstracción y estructura de las clases.

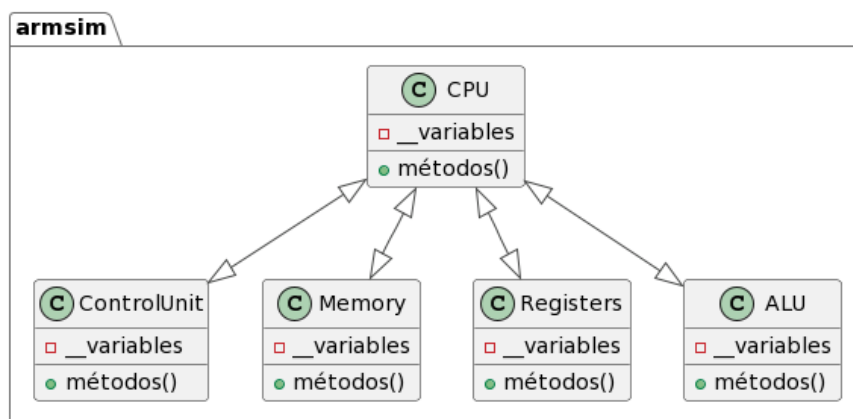


Fig. 7. Modelo simplificado UML del simulador.

4.4 Clases

A continuación, se dará información acerca de las diferentes clases del simulador, será una información de qué hace cada una y cómo, pero al no ser utilizadas por el usuario en su interacción con el simulador no se describirán sus estructuras; a excepción, de la clase CPU que es la única con la cual se utilizará para dar condiciones iniciales y tomar los datos resultantes de la simulación.

Lo primero que hay que poner sobre la mesa es que, este simulador y todas sus clases, se pueden ejecutar concurrentemente por más de un alumno. De aquí subyace una

idea importante, al existir la posibilidad de concurrencia en el uso del simulador debe ser tolerante a ello; es decir, las clases pueden ser que compartan cierta información en su hilo, pero nunca deben compartir entre hilos.

De este razonamiento, se aplicará un patrón de diseño *Singleton* en las clases para aislar cada ejecución de los posibles errores en la concurrencia. El patrón *Singleton* es un patrón de diseño que se utiliza en programación orientada a objetos con el propósito de evitar que se creen más de una instancia de una clase. Esto se logra creando el objeto deseado en una clase y recuperándolo como una instancia estática de este modo el patrón Singleton aporta robustez al código.

NOTA: En el anexo B (Documentación principal) se explica detalladamente la API (por sus siglas en inglés, “Application Programming Interface”) con los métodos y variables de cada una de las clases.

4.4.1 Preprocesador

Es la primera clase en entrar en acción, se encargará de reconocer las líneas introducidas dejarlas completamente limpias quitando las etiquetas y los comentarios dejando las líneas siguiendo el patrón:

Instrucción + [1^{er} operador] + [2^o operador] + [3^{er} operador]

Esto quiere decir que la instrucción es obligatoria y los operadores dependerán de cada una de ellas. Puede ser desde ningún operador hasta tres, esta opcionalidad se remarca poniéndolos entre corchetes. La clase estará descrita en el archivo «preprocessor.py».

Todas las instrucciones con su línea en el archivo original se introducirán a un archivo temporal con un nombre de 32 caracteres alfanuméricos y terminado con la expresión «_Instructions» y con extensión «ds». Se introducirá en una carpeta llamada «tmp», este procedimiento tiene el sentido de evitar que se creen ficheros indiscriminadamente en el servidor en la raíz de ejecución y dar una estructura. Ejemplo:

sn62l40GnO4ie0jH2SZLi4nrn4Ur7rQ0_Instructions.ds

Por otro lado, se creará otro archivo temporal con todas las etiquetas y las líneas del archivo original, y terminado con la expresión «_Labels» y con extensión «ds». También se guardará en la carpeta llamada «tmp». Ejemplo:

sn62l40GnO4ie0jH2SZLi4nrn4Ur7rQ0_Labels.ds

El propio preprocesador comprobará que este nombre aleatorio sea único en la carpeta.

4.4.2 Unidad Central de Procesamiento

La clase será el archivo «cpu.py», tendrá varias características:

1. Procesar las etiquetas para tener un índice de donde se encuentra cada una de ellas.
2. Tomar cada una de las líneas de instrucciones y mandar que se procesen.
3. Ir manteniendo el contador de programa actualizado.
4. Controlar el tiempo de máximo de ejecución de cada hilo.

Esta clase es la que debe comunicarse con el resto, por lo que el profesor, interactuará con ella a través de varios métodos que se han creado para este propósito, por un lado, hay unos métodos para establecer las condiciones iniciales del simulador. Ya sean para la ALU, los registros o la memoria. Esto permitirá en la ALU introducir valor del resultado anterior y la modificación de las banderas. Para los registros se permitirá, uno a uno, modificar de R0 a R15. Y en la memoria se podrá establecer unos límites máximos y mínimos de acceso e introducir en la posición deseada el byte que contenga antes de comenzar la simulación.

`setInitialALUValue`: Función para establecer un valor previo en la ALU antes de ejecutar el código. Posee en este orden y de manera opcional los argumentos resultado anterior, bandera negativa, bandera cero, bandera acarreo y bandera desbordamiento. Ejemplo de uso:

```
cpu.setInitialALUValue(128, 0, 0, 0, 0)
```

`setMemoryLimits`: Función para establecer los valores límite para interactuar con la memoria. Posee en este orden y de manera opcional los argumentos límite inferior de la memoria accesible y límite superior de la memoria accesible. Ejemplo de uso:

```
cpu.setMemoryLimits(0, 2048)
```

`setInitialMemoryValue`: Función para establecer un valor previo de la memoria antes de ejecutar el código. Posee como argumentos dirección de memoria y valor a introducir. Ejemplo de uso:

```
cpu.setInitialMemoryValue("0x10000004", 0xFB88233D)
```

`setInitialRegistersValue`: Función para establecer un valor previo de un registro antes de ejecutar el código.

```
cpu.setInitialRegistersValue("R1", 0x000000AB)
```

Al igual que a la entrada, para la salida también se puede formatear los datos que se devolverán. Podremos especificar los registros que se desean devolver, pueden ser uno, varios o todos. También será posible devolver la memoria, en este caso se puede elegir el rango de memoria. Existen varios modos en los que se podrá obtener la información que posee el simulador con el código de ensamble ya ejecutado, esta información será devuelta en formato *JSON* para facilitar su manejo de manera informática o podrá ser texto plano con todos los datos expuestos al lector:

`getALU`: Función para obtener el estado de la Unidad Aritmeticológica (ALU).

`getMemory`: Función para obtener el estado de la memoria.

`getSpecificRegisters`: Función para obtener el número de instrucciones específicas procesadas.

`getResponseError`: Función para obtener la información cuando se producen errores.

`getResponseSuccess`: Función para obtener el estado actual de todos los componentes.

Por último, se puede pedir el número de veces que sea ejecutado una cierta instrucción, varias o todas; además, se ha añadido la posibilidad de pedir situaciones condicionales con el número de veces que se ha ejecutado una instrucción dando una información extra para la evaluación:

`getTotalLinesProcessed`: Función para obtener el número de líneas de código que se han ejecutado.

`getSpecificProcessedInstructions`: Función para obtener el número de instrucciones específicas procesadas.

En este último método se puede preguntar al simulador para sacar la mejor información para evaluar el código de ensamble y para ello existe una serie de palabras clave, comodines y condicionales:

- Palabras clave:
 - o `ALL`: Muestra toda la tabla con el número de veces que se ejecutaron todas las instrucciones existentes en el código fuente.
 - o `,`: Símbolo para separar diferentes instrucciones.
 - o `:`: Símbolo para excluir una instrucción.
- Comodines:
 - o `?`: Cualquier carácter. Sustituye a un sólo carácter.
 - o `*`: Cualquier carácter. Sustituye a varios caracteres.
- Condicionales:
 - o `>`: Es mayor que un número de veces. Se muestra como Sí o No.
 - o `>=`: Es mayor o igual que un número de veces. Se muestra como Sí o No.

- o <: Es menor que un número de veces. Se muestra como Sí o No.
- o <=: Es menor o igual que un número de veces. Se muestra como Sí o No.
- o ==: Es igual que un número de veces. Se muestra como Sí o No.
- o !=: Es diferente que un número de veces. Se muestra como Sí o No.

Como se ha mencionado no hay entradas ni salidas en el dispositivo porque en tiempo de ejecución no se van a usar.

A continuación, se va a ejemplificar las salidas en modo texto de varios de estos métodos. Para la memoria se debe especificar un rango, puede ser en binario, octal decimal o hexadecimal.

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0x10000000	0xF0	0xC7	0xD6	0x6A	0x3D	0x23	0xB8	0xFB	0x15	0x5E	0xE5	0xEE	0x00	0x00	0x00	0x00
0x10000010	0x3C	0xB9	0xD9	0x81	0xCC	0x60	0x4A	0xB4	0xAD	0xDE	0x00	0x00	0x31	0x00	0x00	0x00
0x10000020	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x10000030	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x10000040	0xF0	0xC7	0xD6	0x6A	0xC3	0xDC	0x77	0x04	0xEB	0xA1	0x1A	0x11	0x00	0x00	0x00	0x00

Fig. 8. Ejemplo devolución de memoria.

Para los registros se pueden pedir cualquiera de los registros por cualquiera de sus nombres.

Register	Value
CONTROL	0x00000000
PRIMASK	0x00000000
PSR	0x40000000
R0	0x1000000C
R1	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13	0x20001000
R14	0xFFFFFFFF
R15	0x00000010
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000

Fig. 9. Ejemplo devolución registros.

Donde se produce el mayor control es en el número de instrucciones de cada tipo. Se pueden pedir instrucciones concretas, varias de ellas separadas por una coma o todas con la palabra especial «ALL». Pero en esta parte es donde se puede encontrar una mayor cantidad de situaciones para controlar y evaluar el código de los alumnos. Y es, por eso que, se crea la situación especial en que se pueda pedir al simulador que nos comunique si ha dado ciertas situaciones y nos responda con verdadero o falso. De este modo, se puede pedir si una instrucción se ha realizado más (o igual) de un

número de veces, menos (o igual) a un número de veces, o que se haya (o no) ejecutado exactamente un número de veces.

Command	# Uses
ADDS	3
B	3
BEQ	26076
BGE	3
CMP	4
LDR	5
RSBS	2
STR	3

Fig. 10. Ejemplo devolución instrucciones procesadas.

4.4.3 Unidad Aritmeticológica

Esta clase, contenida en el archivo «alu.py», se representa el funcionamiento de una unidad aritmeticológica. Una ALU es una unidad aritmeticológica, un componente fundamental de los procesadores y su función es realizar operaciones básicas como suma, resta, multiplicación, división, lógica booleana y comparaciones entre dos números. La ALU recibe uno o dos operandos de entrada y una señal de control que indica qué operación realizar, se realiza el cálculo y, finalmente, se devuelve el resultado de la operación con los posibles bits de acarreo, desbordamiento o cualquier otra bandera con información. La ALU es una parte esencial del camino de datos de la CPU del microcontrolador ARM Cortex M0.

Al ser una ALU programada se ha hecho uso de las operaciones que ya lleva implementadas Python 3, lo cual ha reducido mucho el trabajo a realizar y se ha utilizado el tipo *integer* (*int*) que, por defecto, tiene un tamaño de 32bits al igual que el microcontrolador ARM Cortex M0.

El uso de esta clase es simple, solo habrá que crear un objeto y llamando a la operación aritmeticológica deseada, introducir los operandos necesarios y se devolverá el resultado.

4.4.4 Unidad de Control

La unidad de control será la clase más importante del simulador. Estará contenida en el archivo «control_unit.py». Se encargará de procesar cada una de las instrucciones que se le pasen desde la unidad de control. Cabe puntualizar que en el preprocesador se filtraban que tuvieran la estructura correcta, no se validaban; aquí, se comprueba que cada instrucción se corresponda con la lógica del dispositivo y se realiza su procesado. Es decir, las instrucciones que son válidas para este microcontrolador en concreto se procesan y simulan.

4.4.5 Registros

La clase que describe los registros estará contenida en el archivo «registers.py». Son válidos todos los registros usando indistintamente la designación que se desee. Pueden usarse de R0 a R10, R11 (o FP), R12 (o IP), R13 (o MSP), R13 (o PSP o SP), R14 (o LR), R15 (PC), PSR (o APSR o IPSR o EPSR o IEPSR o IAPSR o EAPSR), PRIMASK y CONTROL. Se puede leer y escribir en cualquiera de ellos, siempre que sea posible por arquitectura.

4.4.6 Memoria

Archivo «memory.py». Representará la memoria y se puede leer o escribir en cualquier posición en modo byte, o en posiciones divisibles entre dos en modo media palabra, o en posiciones divisibles entre cuatro en modo palabra completa. Pensando en la modularidad de esta clase se permite cambiar el tipo de *endian* y poner los límites máximos y mínimos de la memoria.

4.4.7 Utilidades

Esta clase, es parte de un marco de trabajo o *framework* para facilitar ciertas funciones repetitivas en la programación del simulador. Las funciones que se contendrán serán las siguientes:

- Función para transformar una cadena de texto que represente un número binario a una cadena de texto que represente el mismo número en hexadecimal.
- Función para transformar cualquier número introducido en un número decimal. El número puede ser introducido como número o cadena de texto, y en formato binario, octal, decimal o hexadecimal.
- Función para transformar cualquier número introducido en una cadena de texto con el número representado en binario. El número puede ser introducido como número o cadena de texto.
- Función para transformar cualquier número introducido en una cadena de texto con el número representado en hexadecimal. El número puede ser introducido como número o cadena de texto.
- Función para ordenar texto de un modo natural.

Tabla 2. Orden normal frente orden natural.

Orden normal	Orden natural
Elem 1	Elem 1
Elem 10	Elem 2
Elem 2	Elem 10

4.4.8 Excepciones:

En Python, se cuenta con la característica de poder definir las excepciones que van a lanzar otras clases. En el archivo «exceptions.py» se explicitarán todas y cada una las excepciones que se usarán en el código del simulador que básicamente son una excepción por cada clase. Esta clase es parte también del *framework*.

La estructura de las excepciones viene definida por una clase padre genérica que contiene información sobre el error que ha producido la excepción en la ejecución del código, como los errores se pueden crear en una clase e irse propagando por otras tendrán un campo para mantener esa información. La estructura de las excepciones será la siguiente:

- Code: Código de error generado.
- Previous: Cuando una excepción lo que hace es propagar un error de un nivel anterior de código, la información de este error primigenio se contendrá en este campo.
- Message: Mensaje corto del error generado.
- Details: Mensaje detallado del error generado.

NOTA: En el anexo B (Documentación principal) se explica detalladamente la API con los métodos y variables de cada una de las clases.

5 Presupuesto

Al ser el Proyecto de Fin de Grado desarrollado con *software* libre y poderse desplegar servidores de la UPM, no ha se genera coste alguno.

Se puede estimar el coste que podría conllevar la creación de este proyecto en un ámbito no académico, aproximadamente, sería el siguiente:

- Desarrollo Python del Simulador (300 horas x 25€/hora): 7500€
- Despliegue de un entorno Moodle (10 horas x 50€/hora): 500€

Con todo ello, el coste total inicial hubiera sido de 8000€, impuestos excluidos. No se considera el hecho de tener que contratar un alojamiento web de coste mensual, ni las tareas propias de mantenimiento de ninguna de las dos tareas, cuyo coste se podía estimar en 20 euros mensuales. Para cada una de las tareas es necesario un profesional diferente.

6 Impacto del proyecto

En un mundo cada vez más interconectado y dependiente de la tecnología, el proyecto nace con el propósito de aumentar las características ya existentes del marco tecnológico del tándem MoodleCodeRunner, para dar una solución sostenible al aprendizaje y desarrollo en el campo de la electrónica y la programación de microcontroladores. El proyecto es un simulador diseñado para facilitar la comprensión y práctica de microcontroladores, contribuyendo así al logro de los Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030 de las Naciones Unidas.

De acuerdo con el ODS 4 sobre educación de calidad este proyecto con el simulador creado, se desarrolla un entorno accesible de manera universal que ayudará a cualquier persona sin distinción de raza o género pueda estudiar y ser evaluado sobre el microprocesador ARM Cortex M0. Al proporcionar una plataforma de aprendizaje en línea, no solo facilita el acceso a la educación, sino que también a que los estudiantes estarán adquiriendo unas competencias técnicas y profesionales contribuyendo a su preparación de para empleos relacionados con la tecnología, alineándose con el ODS 8 sobre Trabajo Decente y Crecimiento Económico.

Por otro lado, el estudiante adquiere conocimientos sobre una tecnología, que puede reutilizar multitud de veces y para usos distintos, de este modo, la tecnología deja de ser algo de usar y tirar pasando a ser algo vivo que puede ser solución a múltiples problemas. Así, a la vez que el impacto ambiental es menor y se contribuye una disminución de la basura gris se crea un consumo responsable de tecnología necesitando una menor producción disminuyendo la explotación de recursos en el planeta, apoyando así el ODS 12 sobre Producción y Consumo Responsables.

Haciendo uso de la educación en línea el proyecto fomenta la colaboración y el intercambio de conocimientos mediante la creación de una comunidad en línea. Los usuarios pueden compartir proyectos, códigos y experiencias, contribuyendo al ODS 17 sobre Alianzas para lograr los Objetivos.

Por último, destacar que la universalidad del proyecto colabora tanto el ODS 5 sobre igualdad de género y el ODS 10 sobre la reducción de las desigualdades, el software no tiene distinciones por sesgo y al ser sólo un código diseñado como un proyecto fin de grado se podría liberar para que fuese gratuito para su uso por cualquiera que lo necesitare.

7 Conclusiones

La educación en línea se ha arraigado ya en la sociedad, ya sea para cursar itinerarios educativos completos o para complementar la educación presencial tradicional. La elaboración de herramientas que se adecúen a las distintas materias para poder evaluar de la manera correcta se ha vuelto algo imprescindible.

En este proyecto se ha elaborado una de esas herramientas para mejorar la evaluación de los alumnos, en concreto, se ha creado un simulador que permita evaluar alumnos en el lenguaje de ensamble del microcontrolador ARM Cortex M0 a través del *plugin* CodeRunner de la plataforma Moodle. Será utilizado en la asignatura de Microprocesadores de la Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación de la Universidad Politécnica de Madrid.

El proyecto cumple con todos los requisitos planteados inicialmente. Durante las pruebas realizadas se ha comprobado que el simulador es capaz de ejecutar más de 30.000 instrucciones por segundo de tal modo que se evite la sobrecarga del servidor JOBE.

Se han añadido funcionalidades no planteadas inicialmente como el hecho de poder pedir explícitamente información condicional al simulador para comprobar que el alumno ha ejecutado de manera legal el código fuente para dar solución al problema propuesto.

El autor del presente proyecto considera que el mismo ha superado las expectativas iniciales y va a resultar de gran ayuda para evaluar a los alumnos en sus habilidades sobre el lenguaje de ensamble del microcontrolador ARM Cortex M0.

Tal y como se ha planteado la resolución del proyecto hace que pueda ser mantenido sin problema y ampliable por futuros alumnos. En opinión de este autor habría tres vías futuras para seguir desarrollando este proyecto, la primera sería aprovechar la modularidad del código fuente para que se usase en cualquier microcontrolador ARM de la serie Cortex M, una segunda sería darle la estructura a todo el simulador para poder empaquetarlo según las reglas de Python y así poder usarse como una librería más dentro del servidor JOBE de modo que se evitaría tener que añadirlo en cada pregunta y, por último, muy interesante sería la creación de un corrector de preguntas tomando la salida del simulador y evaluando las condiciones finales.

8 Referencias

- [1] E.T.S. de Ingenieria y Sistemas, «MICROPROCESADORES | E.T.S. de Ingenieria y Sistemas UPM,» [En línea]. Available: https://www.upm.es/comun_gauss/publico/guias/202324/2S/GA_59EC_595000021_2S_202324.pdf. [Último acceso: 23 10 5].
- [2] CodeRunner, «CodeRunner,» [En línea]. Available: <https://coderunner.org.nz/>. [Último acceso: 2021 07 05].
- [3] JOBE, «JOBE,» [En línea]. Available: <https://github.com/trampgeek/jobeblob/master/README.md>. [Último acceso: 2021 07 05].
- [4] Python, «Python,» [En línea]. Available: <https://www.python.org/>. [Último acceso: 2021 07 05].
- [5] CodeRunner, «CodeRunner GitHub,» [En línea]. Available: https://github.com/trampgeek/moodleqtype_coderunner. [Último acceso: 2021 05 07].
- [6] Python, «Python Documentation,» [En línea]. Available: <https://www.python.org/doc/>. [Último acceso: 2021 07 05].
- [7] ARM, «ARM developer,» [En línea]. Available: <https://developer.arm.com/ipproducts/processors/cortexm/cortexm0>. [Último acceso: 2021 07 05].
- [8] ARM, «CortexM0 Technical Reference Manual,» [En línea]. Available: <https://developer.arm.com/documentation/ddi0432/c/preface/aboutthisbook>. [Último acceso: 2021 07 05].
- [9] Moodle, «Plugins de Moodle,» [En línea]. Available: <https://moodle.org/plugins/>. [Último acceso: 2021 07 05].
- [10] Moodle, «Moodle,» [En línea]. Available: <https://moodle.org>. [Último acceso: 2021 07 05].

9 Anexos

A Manual de usuario.

A.1 ¿Qué es este simulador?

Este simulador es una representación de la arquitectura de von Neumann y lo que hace es interpretar línea a línea el código fuente en lenguaje de ensamble introducido por los alumnos para recrear las situaciones que tendrían los componentes del microcontrolador Cortex M0 en una ejecución real sobre él. Además, este simulador permite, por un lado, introducir condiciones iniciales para poder presentar el ejercicio adecuado, y por otro, evaluar ciertas situaciones que puedan ser convenientes para el profesor y que se basan en impedir situaciones en las que el alumno pueda realizar acciones incorrectas. Para complementar, se aporta un archivo que sirva de plantilla para realizar la personalización de las preguntas. Este archivo está preparado para ejecutarse también en un entorno de programación como VSCode, generando la misma salida de la ejecución del código para que sirvan como respuesta esperada.

A.2 Guía rápida

Tal y como está explicado en el documento que precede a este anexo; el simulador encaja dentro del *plugin* de CodeRunner para Moodle, por lo tanto, se debe tener una serie de precauciones a la hora de utilizarlo. A continuación, se verá de una manera rápida cuáles son esas consideraciones.

A.2.1 Creación de un prototipo de pregunta

Lo primero que se debe hacer es crear un prototipo de pregunta del tipo ARM en lenguaje de ensamble. Esto se hace desde la sección del «Banco de preguntas» del curso. Hay que tener en cuenta que cuando se crea un prototipo, será la base para la todas las preguntas que se deriven de este. Lo más importante está en los apartados «Personalización» y «Personalización avanzada»

En el apartado «Personalización», en el recuadro plantilla, el profesor puede poner un código que sirva de plantilla genérica para todas las preguntas que deriven de este prototipo de pregunta. El archivo adjunto «template.py» sería un ejemplo.

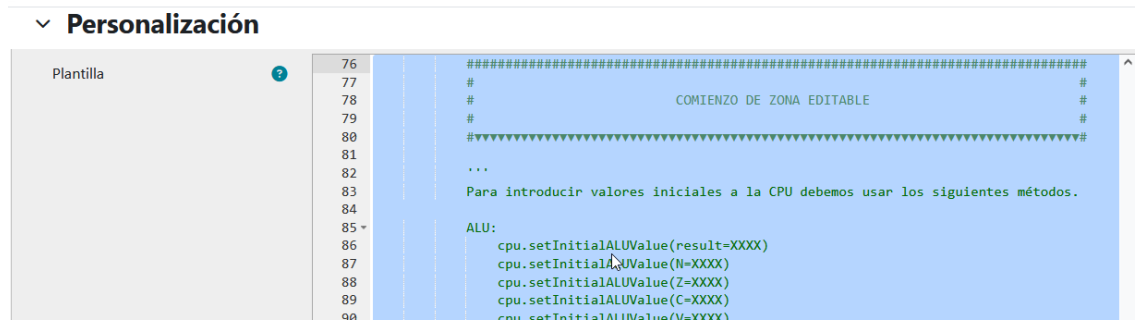


Fig. 11. Plantilla genérica para el prototipo de pregunta.

En «Personalización avanzada» es el apartado donde el profesor especificará que se está creando un nuevo prototipo definido por el usuario, en el tipo de pregunta el nombre con el que aparecerá posteriormente en las preguntas que se creen y finalmente el «Lenguaje Sandbox» que debe ser *python3* ya que todo el proyecto está creado en torno a este lenguaje.

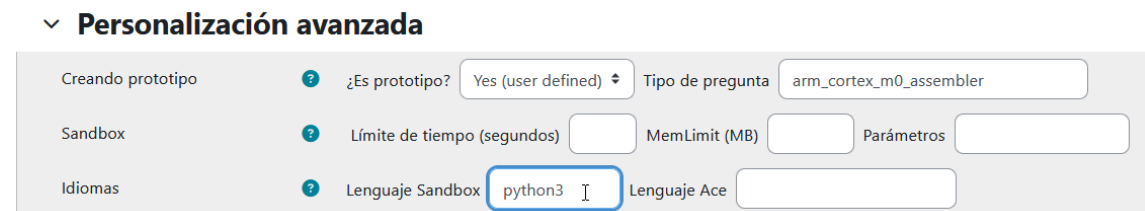


Fig. 12. Información para la creación del prototipo de pregunta.

Un campo útil podría ser el apartado *Attachments*. Se puede pensar que sería muy fácil colocar los archivos del simulador para que fuese usado por la plantilla. Pero hay un problema, la única información que se mantiene para las nuevas preguntas es el nombre para el tipo de pregunta y la propia plantilla.

A.2.2 Añadir el simulador para su uso

Como se veía el apartado anterior, el simulador no se puede adjuntar al prototipo de pregunta. Una solución muy práctica es comprimir el simulador en un archivo Zip y añadirlo como recurso oculto en el curso.

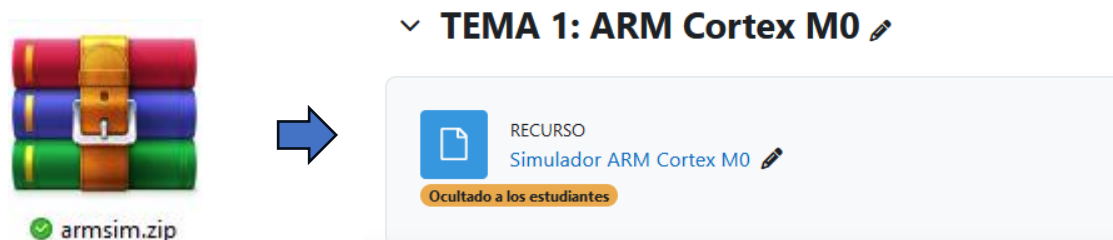


Fig. 13. El simulador como un recurso del curso.

Para la creación de este archivo Zip, se adjunta un archivo que automatice esta tarea con los archivos necesarios.

Ahora, este archivo se podrá vincular o adjuntar a las preguntas del prototipo que se han creado y sin tener que subir el archivo cada vez, ahorrando espacio. Existe una diferencia entre vincular o adjuntar, y es que la primera opción usa el fichero original mientras que la segunda opción crea un nuevo archivo replicado del original, y consecuentemente va ocupando un mayor espacio en el disco. De este modo si vinculamos siempre el archivo, cuando se modifique el original y se sobrescriba se actualizará para todas las preguntas; al contrario, si se adjunta creando una copia del original, cuando se modifique se deberán realizar los cambios en todas las preguntas correspondientes. En el apartado A.2.5 Precauciones se muestra como realizar la sobrescritura.

A.2.3 Creación de una pregunta del tipo ARM Assembler

Una vez creado el prototipo de pregunta y subido el archivo del simulador al curso, ya se pueden crear preguntas directamente para el código en lenguaje de ensamble para ARM.

En cada pregunta se debe adjuntar el simulador, como este ya es un recurso del curso, únicamente se deberá buscar como un archivo local.



Fig. 14. El simulador es parte de los recursos del curso.

A.2.4 Personalizar la pregunta

Ya es posible personalizar la pregunta como cualquier otra, añadiendo el enunciado, las respuestas, ... Simplemente es necesario modificar la plantilla genérica para personalizarla para este ejercicio concreto, añadiendo las condiciones iniciales y el tipo de respuesta a la salida.

Algo muy interesante sería la creación de un corrector de preguntas tomando la salida del simulador y evaluando las conclusiones.

A.2.5 Precauciones

A la hora de comparar las respuestas con el modo propio de CodeRunner se debe pensar en que hace una comparación literal de la respuesta esperada y la generada por el procesador. Esto se ha tenido en cuenta a la hora de generar la respuesta por parte del procesador. Por ejemplo, el simulador, a menos que se desee lo contrario, devuelve los valores de la Unidad Aritmeticológica sin el valor aleatorio para evitar que se produzca un error entre valores. Pero existen situaciones que deben estar controladas a la hora de realizar la pregunta por parte del profesor, un ejemplo se ve a continuación y es pedir el número de saltos del tipo BEQ que ha realizado la CPU, dependiendo de la capacidad de la máquina en la que se ejecute el simulador este valor variará.

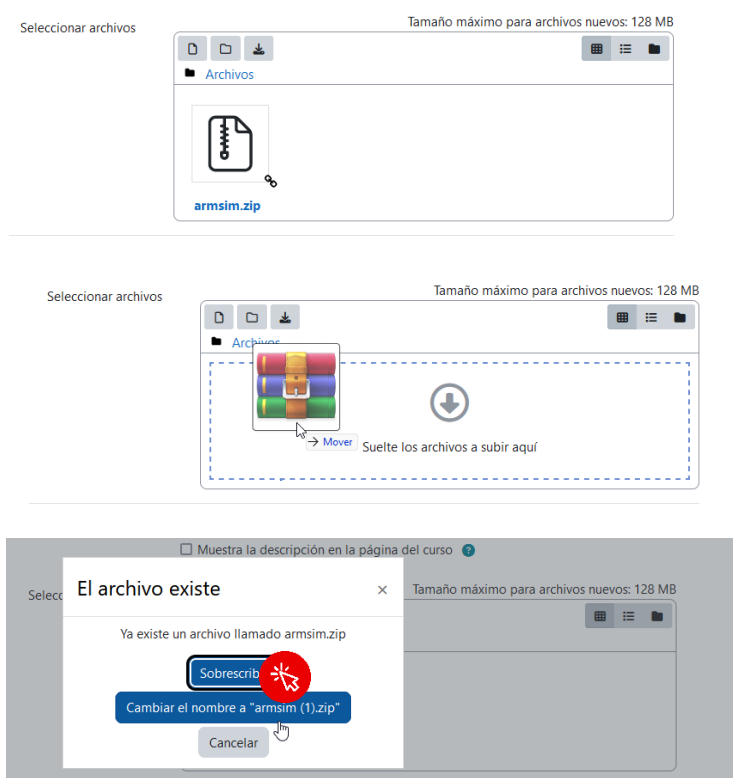
Respuesta esperada:

```
AA": "-", "B": 3, "BEQ": 32453, "BGE":
```

Respuesta obtenida:

```
AA": "-", "B": 3, "BEQ": 34421, "BGE":
```

Existe otra precaución a tener en cuenta. En el supuesto de realizar modificaciones en el código del simulador, a la hora de subirlo de nuevo a moodle es muy importante realizar una sobreescritura del archivo. El realizar esta sobreescritura viene motivado por cómo se adjunta el simulador a las preguntas, existen dos modos de hacerlo; una primera opción es incluir una copia del archivo original, de modo, que se crea un nuevo archivo con la consecuente ocupación del disco. La segunda opción es la de crear un vínculo al archivo original, de manera que, siempre se utilice este archivo. Así llegados al punto de su modificación ya sea por una corrección o una actualización, sólo será necesario modificar el original. Esto se realizará sobreescribiendo el archivo original, arrastrando y soltando el archivo nuevo con idéntico nombre. Como se muestra en esta secuencia.



A.3 Ejemplo

A.3.1 El ejercicio pretendido

Enunciado:

Escriba un programa en lenguaje de ensamble del procesador ARM CortexM0 que lea datos de 32 bits con signo (complemento a 2) ubicados consecutivamente a partir de la dirección 0x1000 0000 y almacene sus valores absolutos en memoria consecutivamente a partir de la dirección 0x1000 0040. El programa terminará cuando encuentre el primer dato de valor 0x0000 0000.

Una posible solución es:

```
LDR R0, =0x10000000
loop
LDR R1, [R0]
CMP R1, #0
forever
BEQ forever
BGE noneg
RSBS R1, R1, #0 ; R1 = R1
noneg
STR R1, [R0, #0x40] ; R1 => [R0 + 0x40]
inc
ADDS R0, #4
B loop
```

Complete la siguiente tabla con los datos en memoria a partir de la dirección 0x1000 0040 tras la ejecución del programa:

Dirección / y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x1000000y	F0	C7	D6	6A	3D	23	88	FB	15	5E	E5	EE	00	00	00	00
0x1000001y	3C	B9	D9	81	CC	60	4A	B4	AD	DE	00	00	31	00	00	00
0x1000002y	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x1000003y	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0x1000004y	F0	C7	D6	6A	C3	DC	77	04	EB	A1	1A	11	00	00	00	00

Como se ha mencionado a lo largo del Anexo, desde el prototipo de pregunta no se pueden adjuntar archivos de soporte para facilitar la creación de las preguntas. Para

esto se ha recurrido a la creación de un script que genera un único fichero ZIP, este archivo contiene el módulo preprocessor y el paquete armsim. De esta manera, se puede añadir el archivo en el curso y puesto en oculto para los alumnos y así enlazarlo a la hora de adjuntar archivos de soporte en cada pregunta.

A.3.2 Creando el prototipo

1. Crear pregunta en el banco de preguntas

Cuestionario Configuración Preguntas Resultados Banco de preguntas Más ▾

Preguntas ▾

Banco de preguntas

Seleccionar una categoría: Por defecto en uP ▾

Categoría por defecto para preguntas compartidas en el contexto uP.

No se está aplicando ningún filtro por etiquetas

Filtrar por etiquetas... ▾

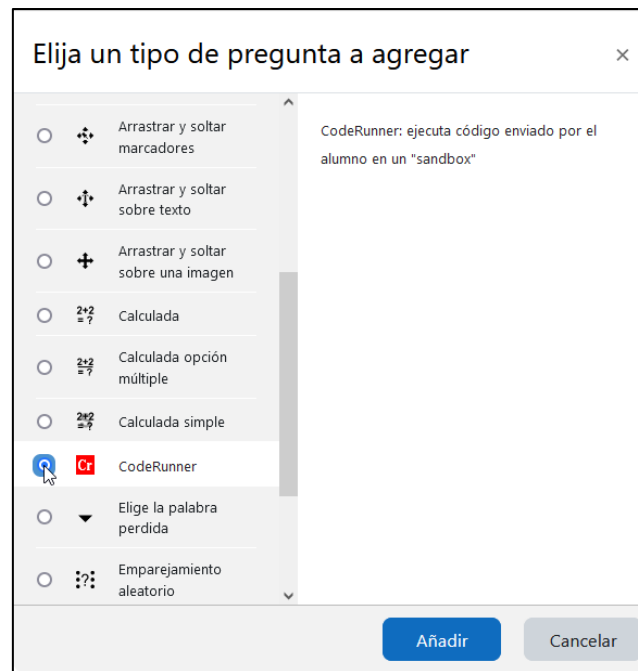
¿Mostrar el enunciado de la pregunta en la lista de preguntas? No ▾

Opciones de búsqueda ▾

- ☒ Mostrar también preguntas de las subcategorías
- ☐ Mostrar también preguntas antiguas

Crear una nueva pregunta ✖

2. Elegir pregunta CodeRunner



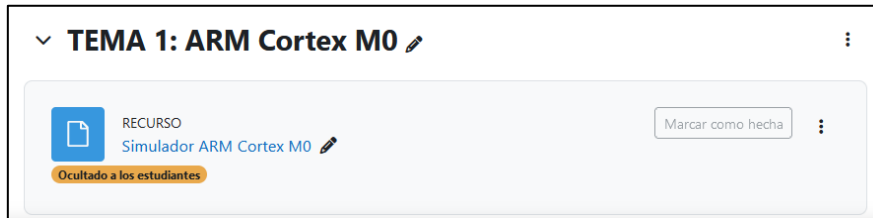
3. Elegir sobre *python3* como lenguaje base para ejecutar el simulador y señalar la casilla "Personalizar" para poder crear el prototipo

4. En «Personalización» añadir la plantilla

5. En «Personalización avanzada» se indicará que es un prototipo definido por el usuario, el nombre que deseado que aparecerá como lenguaje para las preguntas que lo hereden y el lenguaje sobre el cual trabaja este prototipo.

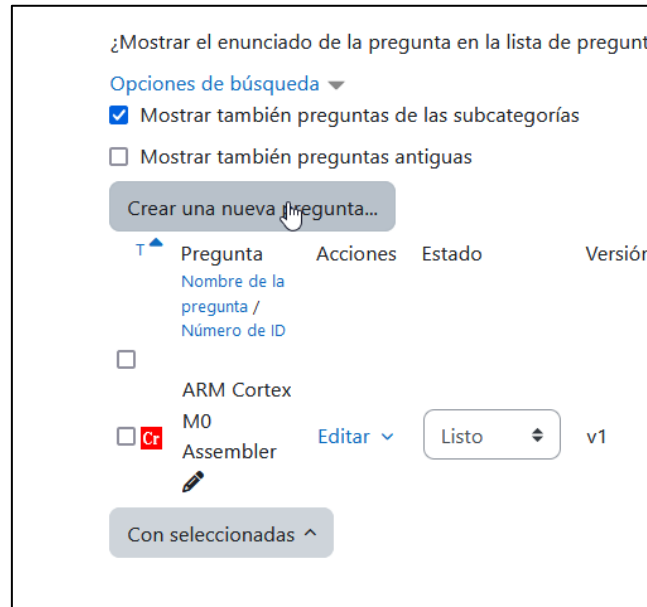
A.3.3 Añadiendo el simulador al curso

Se añade como cualquier otro archivo al curso

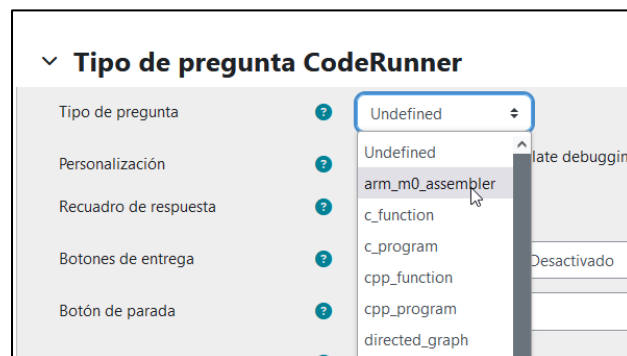


A.3.4 Creando la pregunta

1. En el “Banco de preguntas” del curso, se crea una nueva pregunta



2. Como ya se creó el prototipo, ahora ya está como elegible en la lista



3. Dejando la opción de “Personalizar”, se procede a modificar la plantilla añadiendo los valores iniciales de la memoria de la CPU

```
cpu.setInitialMemoryValue(0x10000000, 0x6AD6C7F0)
```

```
cpu.setInitialMemoryValue("0x10000004", 0xFB88233D)
```

```
cpu.setInitialMemoryValue(0x10000008, "0x5E15")
```

```
cpu.setInitialMemoryValue("0x1000000A", "0xEE5")
```

```
cpu.setInitialMemoryValue(0x10000010, 0x81D9B93C)
```

```
cpu.setInitialMemoryValue("0x10000014", 0xB44A60CC)
```

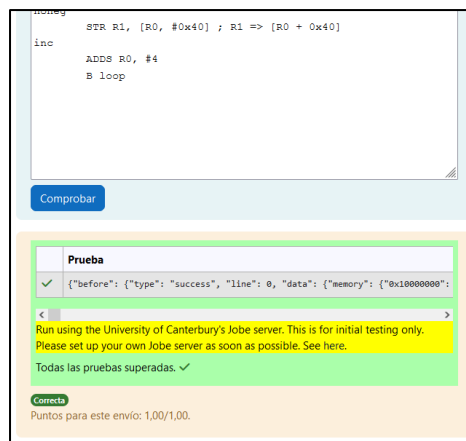
```
cpu.setInitialMemoryValue(0x10000018, "0xDEAD")
```

```
cpu.setInitialMemoryValue("0x1000001C", "0x31")
```

4. Como se puede observar a la hora de usar el método **setInitialMemoryValue** se puede pasar tanto la dirección como el valor a guardar en forma de entero (int) o de cadena de texto (str).
5. Se rellenan los valores del propio enunciado, las respuestas, los casos de prueba y demás campos que se deseen usar en la pregunta
6. Finalmente se adjunta el simulador y estaría lista la pregunta.



7. Se puede probar



B Documentación principal

La documentación expuesta a continuación está replicada en unos archivos HTML para una fácil navegación en la carpeta *Documentation*.

B.1 Módulo preprocess

Módulo Preprocesador, representa el preprocesador de un entorno de desarrollo de código ensamblador. Su principal misión es desmenuzar las instrucciones para facilitar el procesamiento por parte del microcontrolador. En este caso, ayuda al simulador.

B.1.1 Clase preprocessor

Constructor: `preprocessor (code)`

Clase que representa el preprocesador. Divide el código introducido en dos archivos.

Un primer archivo guarda la información acerca de las etiquetas. Ya sean para un salto o no. El segundo archivo, contiene únicamente las instrucciones a procesar.

Argumentos:

`code (string)`: Código fuente completo

B.1.1.1 Método analyzer

Uso: `analyzer ( )`

Función para analizar el código del alumno y crear los archivos de procesamiento.

Devuelve:

`bool`: Devuelve True si han generado correctamente los archivos.

B.2 Módulo armsim

Este es el módulo completo que simula el ARM Cortex M0.

Basados en la arquitectura de von Neumann, representada en los distintos ficheros, se procura representar lo más fielmente su funcionamiento.

B.2.1 Clase ALU

Constructor: ALU (width)

Clase que representa la ALU del microcontrolador ARM Cortex M0.

Argumentos:

- width (Opcional): ancho de palabra que se usará para los cálculos, por defecto, 32 bits.

B.2.1.1 Método AND

Uso: AND(op1, op2)

Función para realizar la operación lógica Y.

Argumentos:

- op1: Primer operador.
- op2: Segundo operador.

Devuelve:

int: Resultado de la operación.

Excepciones:

- ALUError:
 - o ALU:AND:01: Valor del primer operador no valido.
 - o ALU:AND:02: Valor del segundo operador no valido.
 - o ALU:AND:03: No se ha podido calcular la bandera N.
 - o ALU:AND:04: No se ha podido calcular la bandera Z.
 - o ALU:AND:05: La ALU no ha podido guardar el resultado de la operación.

B.2.1.2 Método ArithmeticShiftLeft

Uso: ArithmeticShiftLeft(value, n)

Función para realizar la operación lógica desplazamiento aritmético hacia la izquierda.

Argumentos:

- value: Valor numérico sobre el que realizar el desplazamiento.
- n: Número de bits a desplazar.

Devuelve:

- int: Resultado de la operación.

Excepciones:

- ALUError:
 - o ALU:ASL:01: Valor numérico sobre el que realizar el desplazamiento valido.
 - o ALU:ASL:02: Valor de número de bits a desplazar no valido.
 - o ALU:ASL:03: No se ha podido calcular la bandera N.
 - o ALU:ASL:04: No se ha podido calcular la bandera Z.
 - o ALU:ASL:05: No se ha podido calcular la bandera C.
 - o ALU:ASL:06: La ALU no ha podido guardar el resultado de la operación.

B.2.1.3 Método ArithmeticShiftRight

Uso: ArithmeticShiftRight(value, n)

Función para realizar la operación lógica desplazamiento aritmético hacia la derecha.

Argumentos:

- value: Valor numérico sobre el que realizar el desplazamiento.
- n: Número de bits a desplazar.

Devuelve:

- int: Resultado de la operación.

Excepciones:

- ALUError:
 - o ALU:ASR:01: Valor numérico sobre el que realizar el desplazamiento valido.
 - o ALU:ASR:02: Valor de número de bits a desplazar no valido.
 - o ALU:ASR:03: No se ha podido calcular la bandera N.
 - o ALU:ASR:04: No se ha podido calcular la bandera Z.
 - o ALU:ASR:05: No se ha podido calcular la bandera C.
 - o ALU:ASR:06: La ALU no ha podido guardar el resultado de la operación.

B.2.1.4 Método BitClear

Uso: BitClear(op1, op2)

Función para realizar la operación lógica AND en los bits de del primer operador con los complementos de los bits correspondientes del segundo operador.

Argumentos:

- op1: Primer operador.
- op2: Segundo operador.

Devuelve:

- int: Resultado de la operación.

Excepciones:

- ALUError:
 - o ALU:BIT_CLEAR:01: Valor del primer operador no valido.
 - o ALU:BIT_CLEAR:02: Valor del segundo operador no valido.
 - o ALU:BIT_CLEAR:03: No se ha podido calcular la bandera N.
 - o ALU:BIT_CLEAR:04: No se ha podido calcular la bandera Z.
 - o ALU:BIT_CLEAR:05: La ALU no ha podido guardar el resultado de la operación.

B.2.1.5 Método CrircularShiftLeft

Uso: CrircularShiftLeft(value, n)

Función para realizar la operación lógica desplazamiento circular hacia la izquierda.

Argumentos:

- value: Valor numérico sobre el que realizar el desplazamiento.
- n: Número de bits a desplazar.

Devuelve:

- int: Resultado de la operación.

Excepciones:

- ALUError:
 - o ALU:CSL:01: Valor numérico sobre el que realizar el desplazamiento valido.
 - o ALU:CSL:02: Valor de número de bits a desplazar no valido.
 - o ALU:CSL:03: No se ha podido calcular la bandera N.
 - o ALU:CSL:04: No se ha podido calcular la bandera Z.
 - o ALU:CSL:05: No se ha podido calcular la bandera C.
 - o ALU:CSL:06: La ALU no ha podido guardar el resultado de la operación.

B.2.1.6 Método CrircularShiftRight

Uso: CrircularShiftRight(value, n)

Función para realizar la operación lógica desplazamiento circular hacia la derecha.

Argumentos:

- value: Valor numérico sobre el que realizar el desplazamiento.
- n: Número de bits a desplazar.

Devuelve:

- int: Resultado de la operación.

Excepciones:

- ALUError:
 - o ALU:CSR:01: Valor numérico sobre el que realizar el desplazamiento valido.
 - o ALU:CSR:02: Valor de número de bits a desplazar no valido.
 - o ALU:CSR:03: No se ha podido calcular la bandera N.
 - o ALU:CSR:04: No se ha podido calcular la bandera Z.
 - o ALU:CSR:05: No se ha podido calcular la bandera C.
 - o ALU:CSR:06: La ALU no ha podido guardar el resultado de la operación.

B.2.1.7 Método LogicalShiftLeft

Uso: LogicalShiftLeft(value, n)

Función para realizar la operación lógica desplazamiento lógico hacia la izquierda.

Argumentos:

- value: Valor numérico sobre el que realizar el desplazamiento.
- n: Número de bits a desplazar.

Devuelve:

- int: Resultado de la operación.

Excepciones:

- ALUError:
 - o ALU:LSL:01: Valor numérico sobre el que realizar el desplazamiento valido.
 - o ALU:LSL:02: Valor de número de bits a desplazar no valido.
 - o ALU:LSL:03: No se ha podido calcular la bandera N.
 - o ALU:LSL:04: No se ha podido calcular la bandera Z.
 - o ALU:LSL:05: No se ha podido calcular la bandera C.
 - o ALU:LSL:06: La ALU no ha podido guardar el resultado de la operación.

B.2.1.8 Método LogicalShiftRight

Uso: LogicalShiftRight(value, n)

Función para realizar la operación lógica desplazamiento lógico hacia la derecha.

Argumentos:

- value: Valor numérico sobre el que realizar el desplazamiento.
- n: Número de bits a desplazar.

Devuelve:

- int: Resultado de la operación.

Excepciones:

- ALUError:
 - o ALU:LSR:01: Valor numérico sobre el que realizar el desplazamiento valido.
 - o ALU:LSR:02: Valor de número de bits a desplazar no valido.
 - o ALU:LSR:03: No se ha podido calcular la bandera N.
 - o ALU:LSR:04: No se ha podido calcular la bandera Z.
 - o ALU:LSR:05: No se ha podido calcular la bandera C.
 - o ALU:LSR:06: La ALU no ha podido guardar el resultado de la operación.

B.2.1.9 Método NOT

Uso: NOT(op)

Función para realizar la operación lógica NO.

Argumentos:

- op: Operador.

Devuelve:

- int: Resultado de la operación.

Excepciones:

- ALUError:
 - o ALU:NOT:01: Valor del primer operador no valido.
 - o ALU:NOT:02: Valor del segundo operador no valido.
 - o ALU:NOT:03: No se ha podido calcular la bandera Z.
 - o ALU:NOT:04: La ALU no ha podido guardar el resultado de la operación.

B.2.1.10 Método OR

Uso: OR(op1, op2)

Función para realizar la operación lógica O.

Argumentos:

- op1: Primer operador.
- op2: Segundo operador.

Devuelve:

- int: Resultado de la operación.

Excepciones:

- ALUError:
 - o ALU:OR:01: Valor del primer operador no valido.
 - o ALU:OR:02: Valor del segundo operador no valido.
 - o ALU:OR:03: No se ha podido calcular la bandera N.
 - o ALU:OR:04: No se ha podido calcular la bandera Z.
 - o ALU:OR:05: La ALU no ha podido guardar el resultado de la operación.

B.2.1.11 Método Reverse

Uso: Reverse(op, bitsTruncate)

Función para realizar la operación lógica de invertir el orden de los bits del operador.

Argumentos:

- op: Operador sobre el que realizar la operación.
- bitsTruncate: Cuantos bits a invertir.

Devuelve:

- int: Resultado de la operación.

Excepciones:

- ALUError:
 - o ALU:REVERSE:01: Valor del operador no valido.
 - o ALU:REVERSE:02: Valor del número de bits a invertir no valido.
 - o ALU:REVERSE:03: La ALU no ha podido guardar el resultado de la operación.

B.2.1.12 Método Reverse16

Uso: Reverse16(op)

Función para realizar la operación lógica de invertir el orden de los 16 bits de menor peso del operador.

Argumentos:

- op: Operador sobre el que realizar la operación.

Devuelve:

- int: Resultado de la operación.

Excepciones:

- ALUError:
 - o ALU:REVERSE16:01: La ALU no ha podido invertir los 16 bits de menor peso.

B.2.1.13 Método Reverse32

Uso: Reverse32(op)

Función para realizar la operación lógica de invertir el orden de los bits del operador.

Argumentos:

- op: Operador sobre el que realizar la operación.

Devuelve:

- int: Resultado de la operación.

Excepciones:

- ALUError:
 - o ALU:REVERSE32:01: La ALU no ha podido invertir los bits.

B.2.1.14 Método XOR

Uso: XOR(op1, op2)

Función para realizar la operación lógica O exclusiva.

Argumentos:

- op1: Primer operador.
- op2: Segundo operador.

Devuelve:

- int: Resultado de la operación.

Excepciones:

- ALUError:
 - o ALU:XOR:01: Valor del primer operador no valido.
 - o ALU:XOR:02: Valor del segundo operador no valido.
 - o ALU:XOR:03: No se ha podido calcular la bandera N.
 - o ALU:XOR:04: No se ha podido calcular la bandera Z.
 - o ALU:XOR:05: La ALU no ha podido guardar el resultado de la operación.

B.2.1.15 Método add

Uso: add(op1, op2, flags, carry)

Función para realizar la operación suma.

Argumentos:

- op1: Primer operador.
- op2: Segundo operador.
- flags (bool, opcional): Booleano que nos permite elegir si actualizar las banderas (True) o no (False). Por defecto, False.
- carry (int, opcional): Valor del acarreo que debe tenerse en cuenta. Por defecto, 0.

Devuelve:

- int: Resultado de la operación.

Excepciones:

- ALUError:
 - o ALU:ADD:01: Valor del primer operador no valido.
 - o ALU:ADD:02: No se ha podido obtener el signo del primer operador.

- o ALU:ADD:03: Valor del segundo operador no valido.
- o ALU:ADD:04: No se ha podido obtener el signo del segundo operador.
- o ALU:ADD:05: Acarreo no valido, no es un entero. Sólo puede tener valor 0 ó 1.
- o ALU:ADD:06: No se ha podido calcular la bandera N.
- o ALU:ADD:07: No se ha podido calcular la bandera Z.
- o ALU:ADD:08: No se ha podido calcular la bandera C.
- o ALU:ADD:09: No se ha podido calcular la bandera V.
- o ALU:ADD:10: La ALU no ha podido guardar el resultado de la operación.

B.2.1.16 Método addWithCarry

Uso: addWithCarry(op1, op2)

Función para realizar la operación suma con actualización de las banderas y acarreo.

Argumentos:

- op1: Primer operador.
- op2: Segundo operador.

Devuelve:

- int: Resultado de la operación.

Excepciones:

- ALUError:
 - o ALU:ADD_WITH_CARRY:01: Problema para realizar la operación suma con actualización de las banderas y acarreo.

B.2.1.17 Método adds

Uso: adds(op1, op2)

Función para realizar la operación suma con actualización de las banderas.

Argumentos:

- op1: Primer operador.
- op2: Segundo operador.

Devuelve:

- int: Resultado de la operación.

Excepciones:

- ALUError:
 - o ALU:ADD:01: Problema para realizar la operación suma con actualización de las banderas.

B.2.1.18 Método getAL

Uso: getAL()

Función ALEATORIA, genera un bit aleatorio.

Argumentos:

- Vacío

Devuelve:

- bool: Booleano con la información requerida.

B.2.1.19 Método getALU

Uso: getALU()

Función para obtener el estado de la ALU.

Argumentos:

- Vacío

Devuelve:

- str: Archivo JSON con la información completa de la ALU.

Ejemplo de salida:

- {'result':'0x0000003A', 'N':'0', 'Z':'1', ..., 'AL':'X'}

B.2.1.20 Método getC

Uso: getC()

Función para conocer el estado de la bandera de acarreo con referencia al número actual de la ALU.

Argumentos:

- Vacío

Devuelve:

- int: Valor 0 cuando NO hay acarreo y 1 cuando hay acarreo.

B.2.1.21 Método getCC

Uso: getCC()

Función MENOR QUE sin tener en cuenta el signo.

Argumentos:

- Vacío

Devuelve:

- bool: Booleano con la información requerida.

B.2.1.22 Método getCS

Uso: getCS()

Función MAYOR O IGUAL QUE sin tener en cuenta el signo.

Argumentos:

- Vacío

Devuelve:

- bool: Booleano con la información requerida.

B.2.1.23 Método getEQ

Uso: getEQ()

Función IGUAL QUE, el último resultado del cálculo de la bandera fue cero.

Argumentos:

- Vacío

Devuelve:

- bool: Booleano con la información requerida.

B.2.1.24 Método getGE

Uso: getGE()

Función MAYOR O IGUAL QUE teniendo en cuenta el signo.

Argumentos:

- Vacío

Devuelve:

- bool: Booleano con la información requerida.

B.2.1.25 Método getGT()

Uso: getGT()

Función MAYOR QUE teniendo en cuenta el signo.

Argumentos:

- Vacío

Devuelve:

- bool: Booleano con la información requerida.

B.2.1.26 Método getHI()

Uso: getHI()

Función MAYOR QUE sin tener en cuenta el signo.

Argumentos:

- Vacío

Devuelve:

- bool: Booleano con la información requerida.

B.2.1.27 Método getHS

Uso: getHS()

Función MAYOR O IGUAL QUE sin tener en cuenta el signo.

Argumentos:

- Vacío

Devuelve:

- bool: Booleano con la información requerida.

B.2.1.28 Método getLE

Uso: getLE()

Función MENOR O IGUAL QUE teniendo en cuenta el signo.

Argumentos:

- Vacío

Devuelve:

- bool: Booleano con la información requerida.

B.2.1.29 Método getLO

Uso: getLO()

Función MENOR QUE sin tener en cuenta el signo.

Argumentos:

- Vacío

Devuelve:

- bool: Booleano con la información requerida.

B.2.1.30 Método getLS

Uso: getLS()

Función MENOR O IGUAL QUE sin tener en cuenta el signo.

Argumentos:

- Vacío

Devuelve:

- bool: Booleano con la información requerida.

B.2.1.31 Método getLT

Uso: getLT()

Función MENOR QUE teniendo en cuenta el signo.

Argumentos:

- Vacío

Devuelve:

- bool: Booleano con la información requerida.

B.2.1.32 Método getMI

Uso: getMI()

Función NEGATIVO.

Argumentos:

- Vacío

Devuelve:

- bool: Booleano con la información requerida.

B.2.1.33 Método getN

Uso: getN()

Función para conocer el estado de la bandera de número negativo con referencia al número actual de la ALU.

Argumentos:

- Vacío

Devuelve:

- int: Valor 0 cuando es positivo y 1 cuando es negativo.

B.2.1.34 Método getNE

Uso: getNE()

Función NO IGUAL QUE, el último resultado del cálculo de la bandera NO fue cero.

Argumentos:

- Vacío

Devuelve:

- bool: Booleano con la información requerida.

B.2.1.35 Método getPL

Uso: getPL()

Función POSITIVO O CERO.

Argumentos:

- Vacío

Devuelve:

- bool: Booleano con la información requerida.

B.2.1.36 Método getResult

Uso: getResult()

Función para obtener el resultado de la ALU.

Argumentos:

- Vacío

Devuelve:

- int: Resultado actual de la ALU.

B.2.1.37 Método getV

Uso: getV()

Función para conocer el estado de la bandera de desbordamiento con referencia al número actual de la ALU.

Argumentos:

- Vacío

Devuelve:

- int: Valor 0 cuando NO hay desbordamiento y 1 cuando hay desbordamiento.

B.2.1.38 Método getVC

Uso: getVC()

Función NO DESBORDAMIENTO.

Argumentos:

- Vacío

Devuelve:

- bool: Booleano con la información requerida.

B.2.1.39 Método getVS

Uso: getVS()

Función DESBORDAMIENTO.

Argumentos:

- Vacío

Devuelve:

- bool: Booleano con la información requerida.

B.2.1.40 Método getZ

Uso: getZ()

Función para conocer el estado de la bandera de número cero con referencia al número actual de la ALU.

Argumentos:

- Vacío

Devuelve:

- int: Valor 0 cuando NO es cero y 1 cuando es cero.

B.2.1.41 Método mul

Uso: mul(op1, op2)

Función para realizar la operación multiplicación.

Argumentos:

- op1: Primer operador.
- op2: Segundo operador.

Devuelve:

- int: Resultado de la operación.

Excepciones:

- ALUError:
 - o ALU:MUL:01: Valor del primer operador no valido.
 - o ALU:MUL:02: Valor del segundo operador no valido.
 - o ALU:MUL:03: No se ha podido calcular la bandera N.
 - o ALU:MUL:04: No se ha podido calcular la bandera Z.
 - o ALU:MUL:05: La ALU no ha podido guardar el resultado de la operación.

B.2.1.42 Método setC

Uso: setC(value)

Función para expresar que el número actual de la ALU ha dejado un acarreo.

Argumentos:

- value: Valor que deseamos atribuir a la bandera de acarreo (C).

Devuelve:

- bool: True cuando se establece correctamente la bandera.

Excepciones:

- ALUError:
 - o ALU:SET_V:01: El valor introducido no es un valor entero valido como valor.
 - o ALU:SET_V:02: El valor no es válido, debe ser un booleano o un entero de valor 0 o 1.

B.2.1.43 Método setN

Uso: setN(value)

Función para expresar que el número actual de la ALU es un número negativo.

Argumentos:

- value: Valor que deseamos atribuir a la bandera de número negativo (N).

Devuelve:

- bool: True cuando se establece correctamente la bandera.

Excepciones:

- ALUError:
 - o ALU:SET_N:01: El valor introducido no es un valor entero valido como valor.
 - o ALU:SET_N:02: El valor no es válido, debe ser un booleano o un entero de valor 0 o 1.

B.2.1.44 Método setResult

Uso: setResult(value)

Función para establecer un resultado en la ALU.

Argumentos:

- value: Valor que deseamos atribuir al resultado.

Devuelve:

- Vacío

Excepciones:

- ALUError:
 - o ALU:SET_RESULT:01: No se ha podido actualizar el registro APSR.
 - o ALU:SET_RESULT:02: El valor introducido no es válido. Debe ser un número.

B.2.1.45 Método setV

Uso: setV(value)

Función para expresar que el número actual de la ALU ha generado un desbordamiento.

Argumentos:

- value: Valor que deseamos atribuir a la bandera de desbordamiento (V).

Devuelve:

- bool: True cuando se establece correctamente la bandera.

Excepciones:

- ALUError:
 - o ALU:SET_V:01: El valor introducido no es un valor entero valido como valor.
 - o ALU:SET_V:02: El valor no es válido, debe ser un booleano o un entero de valor 0 o 1.

B.2.1.46 Método setZ

Uso: setZ(value)

Función para expresar que el número actual de la ALU es un cero.

Argumentos:

- value: Valor que deseamos atribuir a la bandera de número cero (Z).

Devuelve:

- bool: True cuando se establece correctamente la bandera.

Excepciones:

- ALUError:
 - o ALU:SET_Z:01: El valor introducido no es un valor entero valido como valor.
 - o ALU:SET_Z:02: El valor no es válido, debe ser un booleano o un entero de valor 0 o 1.

B.2.1.47 Método sub

Uso: sub(op1, op2, flags, carry)

Función para realizar la operación resta.

Argumentos:

- op1: Primer operador.
- op2: Segundo operador.
- flags (bool, opcional): Booleano que nos permite elegir si actualizar las banderas (True) o no (False). Por defecto, False.
- carry (int, opcional): Valor del acarreo que debe tenerse en cuenta. Por defecto, 0.

Devuelve:

- int: Resultado de la operación.

Excepciones:

- ALUError:
 - o ALU:ADD:01: Problema para realizar la operación resta.

B.2.1.48 Método subWithCarry

Uso: subWithCarry(op1, op2)

Función para realizar la operación resta con actualización de las banderas y acarreo.

Argumentos:

- op1: Primer operador.
- op2: Segundo operador.

Devuelve:

- int: Resultado de la operación.

Excepciones:

- ALUError:
 - o ALU:SUB_WITH_CARRY:01: Problema para realizar la operación resta con actualización de las banderas y acarreo.

B.2.1.49 Método subs

Uso: subs(op1, op2)

Función para realizar la operación resta con actualización de las banderas.

Argumentos:

- op1: Primer operador.
- op2: Segundo operador.

Devuelve:

- int: Resultado de la operación.

Excepciones:

- ALUError:
 - o ALU:SUBS:01: Problema para realizar la operación resta con actualización de las banderas.

B.2.1.50 Método truncateAndExtend

Uso: truncateAndExtend(op, bitsTruncate, extendSign)

Función para realizar la operación lógica de truncar y extender un operador.

Argumentos:

- op: Operador sobre el que realizar la operación.
- bitsTruncate: Tamaño al que se truncará el operador.
- extendSign: Opción para extender signo o no. Por defecto, False.

Devuelve:

- int: Resultado de la operación.

Excepciones:

- ALUError:
 - o ALU:TRUNCATE:01: Valor del operador no valido.
 - o ALU:TRUNCATE:02: Tamaño al que se truncará el operador no valido.
 - o ALU:TRUNCATE:03: La ALU no ha podido guardar el resultado de la operación.

B.2.1.51 Método updateAPSR

Uso: updateAPSR()

Función para actualizar todas las banderas con respecto al número actual de la ALU.

Argumentos:

- Vacío

Devuelve:

- bool: True cuando se establece correctamente la bandera.

Excepciones:

- ALUError:
 - o ALU:UPDATE_FLAGS:01: No se ha podido actualizar el registro APSR.
 - o ALU:UPDATE_FLAGS:02: No se pudo obtener el registro APSR.

B.2.2 Clase ControlUnit

Constructor ControlUnit ()

Clase que representa la unidad de control del microcontrolador ARM Cortex M0.

Argumentos:

- Vacío

B.2.2.1 Método `getSpecificProcessedInstructions`

Uso: `getSpecificProcessedInstructions(listInstruccions)`

Función para obtener el número de instrucciones específicas procesadas.

Argumentos:

- `listInstruccions`: Lista con las instrucciones de las cuales se desea saber el número de veces que se han procesado.

Existe una serie de palabras clave, comodines y condicionales para obtener la información.

Palabras clave:

- `ALL`: Muestra toda la tabla con el número de veces que se ejecutaron todas las instrucciones existentes en el código fuente.
- `,`: Símbolo para separar diferentes instrucciones.
- `:`: Símbolo para excluir una instrucción.

Comodines:

- `?`: Cualquier carácter. Sustituye a un sólo carácter.
- `*`: Cualquier carácter. Sustituye a varios caracteres.

Condicionales:

- `>`: Es mayor que un número de veces. Se muestra como Sí o No.
- `>=`: Es mayor o igual que un número de veces. Se muestra como Sí o No.
- `<`: Es menor que un número de veces. Se muestra como Sí o No.
- `<=`: Es menor o igual que un número de veces. Se muestra como Sí o No.
- `==`: Es igual que un número de veces. Se muestra como Sí o No.
- `!=`: Es diferente que un número de veces. Se muestra como Sí o No.

Devuelve:

- `dict`: Diccionario con la información requerida.

B.2.2.2 Método `processInstruction`

Uso: `processInstruction(line)`

Función que procesa una línea del código fuente.

Argumentos:

- `line`: línea que se desea procesar.
Se compone de:
 - o `number`: número de la línea que se desea procesar.
 - o `instruction`: instrucción a procesar.

Devuelve:

- `True`, si se ha realizado correctamente la escritura.

Excepciones:

- `ControlUnitError`:
 - o `CU:xxx:01`: Alguno de los argumentos no es válido. xxx corresponde a cada instrucción.
 - o `CU:NO_VALID_LINE:01`: La línea de código está mal formada.
 - o `CU:EMPTY_LINE:01`: La línea está vacía

B.2.2.3 Método processLabel

Uso: processLabel(line)

Función que procesa las etiquetas del código fuente.

Argumentos:

- line: línea que se desea procesar.

Se compone de:

- o number: número de la línea que se desea procesar.
- o instruction: instrucción a procesar.

Devuelve:

- True, si se ha realizado correctamente la escritura.

Excepciones:

- ControlUnitError:
 - o CU:xxx:01: Alguno de los argumentos no es válido. xxx corresponde a cada instrucción.

B.2.3 Clase CPU

Constructor: CPU (file, timeout)

Clase que representa la CPU del microcontrolador ARM Cortex M0. Se encarga de procesar los archivos de código fuente y gestionar las conexiones con la memoria y demás periféricos.

Argumentos:

- file (str): Archivo de código fuente a procesar.
- timeout (int, opcional): Tiempo máximo que se procesará el código. Por defecto, 1 segundo.

B.2.3.1 Método getALU

Uso: getALU(mode, safe)

Función para obtener el estado de la Unidad Aritmeticológica (ALU).

Argumentos:

- mode (str): salida formateada en JSON o texto plano. Puede tomar los valores JSON o TXT. JSON, por defecto.
- Safe (bool, opcional): Elimina el registro AL(eatorio) de la ALU. "True", por defecto

Devuelve:

String:

- En el modo JSON:
Un archivo formateado en JSON con la información requerida.
Ejemplo de salida:
{'result':5203, 'N':0, 'Z':0, ..., 'LE':false}
- En el modo TXT
Una tabla con la información requerida.
Ejemplo de salida:

5203			
0x00001453			
0000000000000000000000001010001010011			
N:	0	Z:	0
C:	0	V:	0
EQ:	0	NE:	1
CS:	0	HS:	0
CC:	1	LO:	1
MI:	0	PL:	1
VS:	0	VC:	1
HI:	0	LS:	1
GE:	1	LT:	0
GT:	1	LE:	0

B.2.3.2 Método getMemory

Uso: getMemory(listAddress, mode)

Función para obtener el estado de la memoria.

Argumentos:

- listAddress: Lista con las posiciones de memoria requeridas.

Nota:

- : Con el símbolo se pueden introducir rangos.
- mode (str, *opcional*): salida formateada en JSON o texto plano. Puede tomar los valores JSON o TXT. JSON, por defecto.

Devuelve:

Cadena de texto (str):

En el modo "JSON":

Un archivo formateado en JSON con la información requerida.

Ejemplo de salida:

```
{'0x10000000': '0x00', '0x10000001': '0x00', '0x10000002': '0x00', ..., '0x1000001F': '0x00'}
```

En el modo "TXT":

Una tabla con la información requerida.

Ejemplo de salida:

Address	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
0x10000000	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x10000010	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

B.2.3.3 Método getResponseError

Uso: getResponseError(error, mode)

Función para obtener la información cuando se producen errores.

Argumentos:

- error (GenericError): Objeto que contiene el error producido. mode (str): salida formateada en JSON o texto plano. Puede tomar los valores JSON o TXT. JSON, por defecto.

Devuelve:

- Cadena de texto (str): Una cadena con un archivo en formato “JSON”, ya sea puro en el modo JSON o formateado para la facilidad de lectura en el modo “TXT”.

B.2.3.4 Método `getResponseSuccess`

Uso: `getResponseSuccess(listAddress, listInstruccions, listRegisters, viewALU, mode)`

Función para obtener el estado actual de todos los componentes.

Argumentos:

- listlistAddress: Lista con las posiciones de memoria requeridas.

Nota:

: Con el símbolo se pueden introducir rangos.

- listInstruccions (str): Lista con las instrucciones de las cuales se desea saber el número de veces que se han procesado.

Existe una serie de palabras clave, comodines y condicionales para obtener la información.

Palabras clave:

- ALL: Muestra toda la tabla con el número de veces que se ejecutaron todas las instrucciones existentes en el código fuente.
- ,: Símbolo para separar diferentes instrucciones.
- : Símbolo para excluir una instrucción.

Comodines:

- ?: Cualquier carácter. Sustituye a un sólo carácter.
- *: Cualquier carácter. Sustituye a varios caracteres.

Condicionales:

- >: Es mayor que un número de veces. Se muestra como Sí o No.
- >=: Es mayor o igual que un número de veces. Se muestra como Sí o No.
- <: Es menor que un número de veces. Se muestra como Sí o No.
- <=: Es menor o igual que un número de veces. Se muestra como Sí o No.
- ==: Es igual que un número de veces. Se muestra como Sí o No.
- !=: Es diferente que un número de veces. Se muestra como Sí o No.

- listRegisters: Lista con las instrucciones de las cuales se desea saber el número de veces que se han procesado.

Palabras clave:

- ALL: Muestra toda la tabla con el número de veces que se ejecutaron todas las instrucciones existentes en el código fuente.
- ,: Símbolo para separar diferentes instrucciones.
- : Símbolo para excluir una instrucción.

Comodines:

- ?: Cualquier carácter. Sustituye a un sólo carácter.
- *: Cualquier carácter. Sustituye a varios caracteres.

Condicionales:

- >: Es mayor que un número de veces. Se muestra como Sí o No.
- >=: Es mayor o igual que un número de veces. Se muestra como Sí o No.
- <: Es menor que un número de veces. Se muestra como Sí o No.
- <=: Es menor o igual que un número de veces. Se muestra como Sí o No.
- ==: Es igual que un número de veces. Se muestra como Sí o No.
- !=: Es diferente que un número de veces. Se muestra como Sí o No.

- `mode (str)`: salida formateada en JSON o texto plano. Puede tomar los valores JSON o TXT. JSON, por defecto.

Devuelve:

- `str`: Una cadena con un archivo en formato JSON, ya sea puro en el modo JSON o formateado para la facilidad de lectura en el modo TXT.

B.2.3.5 Método `getSpecificProcessedInstructions`

Uso: `getSpecificProcessedInstructions(listInstructions, mode)`

Función para obtener el número de instrucciones específicas procesadas.

Argumentos:

- `listInstructions (str)`: Lista con las instrucciones de las cuales se desea saber el número de veces que se han procesado.

Existe una serie de palabras clave, comodines y condicionales para obtener la información.

Palabras clave:

- `ALL`: Muestra toda la tabla con el número de veces que se ejecutaron todas las instrucciones existentes en el código fuente.
- `,;`: Símbolo para separar diferentes instrucciones.
- `:`: Símbolo para excluir una instrucción.

Comodines:

- `?`: Cualquier carácter. Sustituye a un sólo carácter.
- `*`: Cualquier carácter. Sustituye a varios caracteres.

Condicionales:

- `>`: Es mayor que un número de veces. Se muestra como Sí o No.
- `>=`: Es mayor o igual que un número de veces. Se muestra como Sí o No.
- `<`: Es menor que un número de veces. Se muestra como Sí o No.
- `<=`: Es menor o igual que un número de veces. Se muestra como Sí o No.
- `==`: Es igual que un número de veces. Se muestra como Sí o No.
- `!=`: Es diferente que un número de veces. Se muestra como Sí o No.
- `mode (str)`: salida formateada en JSON o texto plano. Puede tomar los valores JSON o TXT. JSON, por defecto.

Devuelve:

- Cadena de texto (`str`):

En el modo JSON:

Un archivo formateado en JSON con la información requerida.

Ejemplo de salida:

```
{"CONTROL": "0x00000000", "PRIMASK": "0x00000000", ..., "R15": "0x00000010"},
"conditionals": {"R0==0x10000007": "False"}}
```

En el modo TXT

Una tabla con la información requerida.

Ejemplo de salida:

Register	Value
CONTROL	0x00000000
PRIMASK	0x00000000
PSR	0x40000000
R0	0x10000010
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13	0x20001000
R14	0xFFFFFFFF
R15	0x00000010
R0==0x10000007	False

B.2.3.6 Método getSpecificRegisters

Uso: getSpecificRegisters(listRegisters, mode)

Función para obtener el número de instrucciones específicas procesadas.

Argumentos:

- listRegisters: Lista con las instrucciones de las cuales se desea saber el número de veces que se han procesado.

Existe una serie de palabras clave, comodines y condicionales para obtener la información.

Palabras clave:

- ALL: Muestra toda la tabla con el número de veces que se ejecutaron todas las instrucciones existentes en el código fuente.
- ,: Símbolo para separar diferentes instrucciones.
- : Símbolo para excluir una instrucción.

Comodines:

- ?: Cualquier carácter. Sustituye a un sólo carácter.
- *: Cualquier carácter. Sustituye a varios caracteres.

Condicionales:

- >: Es mayor que un número de veces. Se muestra como Sí o No.
- >=: Es mayor o igual que un número de veces. Se muestra como Sí o No.
- <: Es menor que un número de veces. Se muestra como Sí o No.
- <=: Es menor o igual que un número de veces. Se muestra como Sí o No.
- ==: Es igual que un número de veces. Se muestra como Sí o No.
- !=: Es diferente que un número de veces. Se muestra como Sí o No.
- mode (str): salida formateada en JSON o texto plano. Puede tomar los valores JSON o TXT. JSON, por defecto.

Devuelve:

- Cadena de texto (str):

En el modo JSON:

Un archivo formateado en JSON con la información requerida.

Ejemplo de salida:

```
{"instructions": {"ADDS": 4, "B": 4, "BEQ": 24439, "BGE": 4, "CMP": 5, "LDR": 6, "STR": 4}, "conditionals": {"B==8": "False", "CMP>=3": "True"}}
```

En el modo TXT

Una tabla con la información requerida.

Ejemplo de salida:

Command	# Uses
ADDS	4
B	4
BEQ	24444
BGE	4
CMP	5
LDR	6
STR	4
B==8	False
CMP>=3	True

B.2.3.7 Método getTotalLinesProcessed

Uso: getTotalLinesProcessed ()

Función para obtener el número de líneas de código que se han ejecutado.

Argumentos:

- Vacío.

Devuelve:

- int: Número de líneas procesadas.

B.2.3.8 Método process

Uso: process ()

Función para procesar y simular las líneas de código.

Argumentos:

- Vacío

Devuelve:

- int: Número de líneas procesadas.

Excepciones:

- CPUError:
 - o CPU:PC_NEXT: La dirección siguiente no se puede cargar en el SP (Stack Pointer).
 - o CPU:CU_ERROR: Error en la Unidad de Control.
 - o CPU:TIMEOUT: Se ha excedido el tiempo máximo para procesar el código.

B.2.3.9 Método setInitialALUValue

Uso: setInitialALUValue(result, N, Z, C, V)

Función para establecer un valor previo en la ALU antes de ejecutar el código.

Argumentos:

- result (int, opcional): Valor previo para el resultado de la operación anterior.
- N (int, opcional): Valor previo de la bandera Negativo.
- Z (int, opcional): Valor previo de la bandera Zero.
- C (int, opcional): Valor previo de la bandera Carry (Acarreo).
- V (int, opcional): Valor previo de la bandera Overflow (Desbordamiento).

Devuelve:

- bool: True si se ha completado con éxito el establecimiento.

Excepciones:

- CPUError:
 - o CPU:ALU_INITIAL_VALUE:01: No es posible establecer el/los valor/valores en la ALU.
 - o CPU:ALU_INITIAL_VALUE:02: El valor debe ser establecido previamente a procesar las líneas de código.

B.2.3.10 Método setInitialMemoryValue

Uso: setInitialMemoryValue(memAddress, memValue)

Función para establecer un valor previo de la memoria antes de ejecutar el código.

Argumentos:

- memAddress (str, opcional): Dirección de memoria en la que se desea establecer un valor previo.
- memValue (opcional): Valor que se desea establecer como valor previo. 1 Byte.

Devuelve:

- bool: True si se ha completado con éxito el establecimiento.

Excepciones:

- CPUError:
 - o CPU:MEMORY_INITIAL_VALUE:01: No es posible establecer el valor en la memoria.
 - o CPU:MEMORY_INITIAL_VALUE:02: El valor debe ser establecido previamente a procesar las líneas de código.

B.2.3.11 Método setInitialRegistersValue

Uso: setInitialRegistersValue(register, value)

Función para establecer un valor previo de un registro antes de ejecutar el código.

Argumentos:

- register (str, opcional): Registro en la que se desea establecer un valor previo.
- value (opcional): Valor que se desea establecer como valor previo. Hasta 4 Bytes.

Devuelve:

- bool: True si se ha completado con éxito el establecimiento.

Excepciones:

- CPUError:

- o CPU:REGISTERS_INITIAL_VALUE:01: No es posible establecer el valor en el registro.
- o CPU:REGISTERS_INITIAL_VALUE:02: El valor debe ser establecido previamente a procesar las líneas de código.

B.2.3.12 Método setMemoryLimits

Uso: setMemoryLimits(minMemoryLimit, maxMemoryLimit)

Función para establecer los valores límite para interactuar con la memoria.

Argumentos:

- minMemoryLimit (int): Límite inferior de la memoria accesible.
- maxMemoryLimit (int): Límite superior de la memoria accesible.

Devuelve:

- bool: True: Se ha completado con éxito el establecimiento.

Excepciones:

- CPUError:
 - o CPU:MEMORY_SET_LIMITS:01: No es posible establecer los límites.
 - o CPU:MEMORY_SET_LIMITS:02: Los límites deben ser establecido previamente a procesar las líneas de código.

B.2.4 Clase ALUError

Constructor: ALUError(previous, code, message, details)

Clase para representar los errores de la ALU (Unidad Aritmético Lógica).

La clase se usa como decorador. Hereda de la clase GenericError.

Clase base para representar los errores.

Argumentos:

- code (str, opcional): Código de error generado.
- previous (str, opcional): Inicializa con otro error previo.
- message (str, opcional): Mensaje corto del error generado.
- details (str, opcional): Mensaje detallado del error generado.

B.2.5 Clase CPUError

Constructor: CPUError(previous, code, message, details)

Clase para representar los errores de producidos en la CPU (Unidad Central de Procesamiento).

La clase se usa como decorador. Hereda de la clase GenericError.

Clase base para representar los errores.

Argumentos:

- code (str, opcional): Código de error generado.
- previous (str, opcional): Inicializa con otro error previo.
- message (str, opcional): Mensaje corto del error generado.
- details (str, opcional): Mensaje detallado del error generado.

B.2.6 Clase ControlUnitError

Constructor: ControlUnitError(previous, code, message, details)

Clase para representar los errores de la Unidad de Control.

La clase se usa como decorador. Hereda de la clase GenericError.

Clase base para representar los errores.

Argumentos:

- code (str, opcional): Código de error generado.
- previous (str, opcional): Inicializa con otro error previo.
- message (str, opcional): Mensaje corto del error generado.
- details (str, opcional): Mensaje detallado del error generado.

B.2.7 Clase GenericError

Constructor: GenericError(previous, code, message, details)

Clase base para representar los errores.

Argumentos:

- code (str, opcional): Código de error generado.
- previous (str, opcional): Inicializa con otro error previo.
- message (str, opcional): Mensaje corto del error generado.
- details (str, opcional): Mensaje detallado del error generado.

B.2.7.1 Método getCode

Uso: getCode()

Función que devuelve el código error.

Argumentos:

- Vacío

Devuelve:

- str: Cadena de texto con el código de error.

B.2.7.2 Método getDetails

Uso: getDetails()

Función que devuelve el mensaje detallado explicando el error.

Argumentos:

- Vacío

Devuelve:

- str: Cadena de texto con un mensaje detallado del error.

B.2.7.3 Método getMessage

Uso: getMessage()

Función que devuelve el mensaje corto explicando el error.

Argumentos:

- Vacío

Devuelve:

- str: Cadena de texto con un mensaje corto del error.

B.2.7.4 Método getPrevious

Uso: getPrevious()

Función que devuelve el error previo para encadenarlo.

De esta manera no se pierde la trazabilidad y se puede recuperar el error, previo hasta el primero, que desate los demás.

Argumentos:

- Vacío

Devuelve:

- dict: Diccionario los valores del error previo a este para no perder trazabilidad.

B.2.8 Clase MemoryError

Constructor: MemoryError(previous, code, message, details)

Clase para representar los errores de la Memoria.

La clase se usa como decorador. Hereda de la clase GenericError.

Clase base para representar los errores.

Argumentos:

- code (str, opcional): Código de error generado.
- previous (str, opcional): Inicializa con otro error previo.
- message (str, opcional): Mensaje corto del error generado.
- details (str, opcional): Mensaje detallado del error generado.

B.2.9 Clase RegistersError

Constructor: RegistersError(previous, code, message, details)

Clase para representar los errores de los Registros.

La clase se usa como decorador. Hereda de la clase GenericError.

Clase base para representar los errores.

Argumentos:

- code (str, opcional): Código de error generado.
- previous (str, opcional): Inicializa con otro error previo.

- message (str, opcional): Mensaje corto del error generado.
- details (str, opcional): Mensaje detallado del error generado.

B.2.10 Clase UtilsError

Constructor: UtilsError(previous, code, message, details)

Clase para representar los errores de producidos en la librería de ayuda Utils (Utilidades).

La clase se usa como decorador. Hereda de la clase GenericError.

Clase base para representar los errores.

Argumentos:

- code (str, opcional): Código de error generado.
- previous (str, opcional): Inicializa con otro error previo.
- message (str, opcional): Mensaje corto del error generado.
- details (str, opcional): Mensaje detallado del error generado.

B.2.11 Clase Memory

Constructor: Memory(minMem, maxMem, littleEndian)

Clase que representa la memoria de la CPU según la arquitectura de von Neumann.

Clase que representa la Memoria del microcontrolador ARM Cortex M0.

Argumentos:

- minMem (int, opcional): valor mínimo del rango de memoria.
- maxMem (int, opcional): valor máximo del rango de memoria.
- littleEndian (bool, opcional): Especifica el formato en que se guardarán los datos en la memoria.

B.2.11.1 Método getMemory

Uso: getMemory(listAddress)

Función para obtener el estado de la memoria.

Argumentos:

- listAddress (str): Lista con las posiciones de memoria requeridas, usa el símbolo , para separar direcciones. Palabra reservada "ALL" para toda la memoria.

Nota:

: Con el símbolo se pueden introducir rangos.

Devuelve:

- str: Archivo JSON con la información requerida.

Ejemplo de salida:

```
{'0x10000000': '0x3A', '0x10000001': '0x0F', '0x10000002': '0x02'}
```

B.2.11.2 Método readByte

Uso: readByte(address)

Función para leer un byte de una dirección de memoria.

Argumentos:

- address (str): Dirección de memoria que se desea leer.

Devuelve:

- int: Byte leído en formato decimal

Excepciones:

- MemoryError:
 - o MEMORY:READ_BYTE:01: La dirección no es válida no es un número valido.
 - o MEMORY:READ_BYTE:02: La dirección está fuera del rango accesible.

B.2.11.3 Método readHalfWord

Uso: readHalfWord(address)

Función para leer media palabra (2 bytes) de una dirección de memoria.

Argumentos:

- address (str): Dirección de memoria que se desea leer. NOTA: La dirección debe ser múltiplo de 2.

Devuelve:

- int: Byte leído en formato decimal

Excepciones:

- MemoryError:
 - o MEMORY:READ_HWORD:01: La dirección no es válida no es un número valido.
 - o MEMORY:READ_HWORD:02: La dirección está fuera del rango accesible.
 - o MEMORY:READ_HWORD:03: La dirección no es múltiplo de 2.

B.2.11.4 Método readWord

Uso: readWord(address)

Función para leer una palabra (4 bytes) de una dirección de memoria.

Argumentos:

- address (str): Dirección de memoria que se desea leer. NOTA: La dirección debe ser múltiplo de 4.

Devuelve:

- int: Byte leído en formato decimal

Excepciones:

- MemoryError:
 - o MEMORY:READ_WORD:01: La dirección no es válida no es un número valido.
 - o MEMORY:READ_WORD:02: La dirección está fuera del rango accesible.
 - o MEMORY:READ_WORD:03: La dirección no es múltiplo de 4.

B.2.11.5 Método writeByte

Uso: writeByte(address, value)

Función para escribir un byte en una dirección de memoria.

Argumentos:

- address (str): Dirección de memoria que se desea escribir.
- value: Valor del byte a escribir.

Devuelve:

- bool: True, si se ha realizado correctamente la escritura.

Excepciones:

- MemoryError:
 - o MEMORY:WRITE_BYTE:01: La dirección no es válida no es un número valido.
 - o MEMORY:WRITE_BYTE:02: El valor a escribir no es un número valido.
 - o MEMORY:WRITE_BYTE:03: La dirección está fuera del rango accesible.
 - o MEMORY:WRITE_BYTE:04: El valor a escribir está fuera del rango accesible.

B.2.11.6 Método writeHalfWord

Uso: writeHalfWord(address, value)

Función para escribir media palabra (2 bytes) en una dirección de memoria.

Argumentos:

- address (str): Dirección de memoria que se desea escribir. NOTA: La dirección debe ser múltiplo de 2.
- value: Valor del byte a escribir.

Devuelve:

- bool: True, si se ha realizado correctamente la escritura.

Excepciones:

- MemoryError:
 - o MEMORY:WRITE_HWORD:01: La dirección no es válida no es un número valido.
 - o MEMORY:WRITE_HWORD:02: El valor a escribir no es un número valido.
 - o MEMORY:WRITE_HWORD:03: La dirección está fuera del rango accesible.
 - o MEMORY:WRITE_HWORD:04: El valor a escribir está fuera del rango accesible.
 - o MEMORY:WRITE_HWORD:05: La dirección no es múltiplo de 2.

B.2.11.7 Método writeWord

Uso: writeWord(address, value)

Función para escribir una palabra (4 bytes) en una dirección de memoria.

Argumentos:

- address (str): Dirección de memoria que se desea escribir. NOTA: La dirección debe ser múltiplo de 4.
- value: Valor del byte a escribir.

Devuelve:

- bool: True, si se ha realizado correctamente la escritura.

Excepciones:

- MemoryError:
 - o MEMORY:WRITE_WORD:01: La dirección no es válida no es un número valido.
 - o MEMORY:WRITE_WORD:02: El valor a escribir no es un número valido.
 - o MEMORY:WRITE_WORD:03: La dirección está fuera del rango accesible.
 - o MEMORY:WRITE_WORD:04: El valor a escribir está fuera del rango accesible.
 - o MEMORY:WRITE_WORD:05: La dirección no es múltiplo de 4.

B.2.12 Clase Registers

Constructor: Registers()

Clase que representa los Registros del microcontrolador ARM Cortex M0.

B.2.12.1 Método getSpecificRegisters

Uso: getSpecificRegisters(listRegisters)

Función para obtener el número de instrucciones específicas procesadas.

Argumentos:

- listInstruccions (str): Lista con las instrucciones de las cuales se desea saber el número de veces que se han procesado.

Existe una serie de palabras clave, comodines y condicionales para obtener la información.

Palabras clave:

- ALL: Muestra toda la tabla con el número de veces que se ejecutaron todas las instrucciones existentes en el código fuente.
- ,: Símbolo para separar diferentes instrucciones.
- : Símbolo para excluir una instrucción.

Comodines:

- ?: Cualquier carácter. Sustituye a un sólo carácter.
- *: Cualquier carácter. Sustituye a varios caracteres.

Condicionales:

- >: Es mayor que un número de veces. Se muestra como Sí o No.
- >=: Es mayor o igual que un número de veces. Se muestra como Sí o No.
- <: Es menor que un número de veces. Se muestra como Sí o No.
- <=: Es menor o igual que un número de veces. Se muestra como Sí o No.
- ==: Es igual que un número de veces. Se muestra como Sí o No.
- !=: Es diferente que un número de veces. Se muestra como Sí o No.

Devuelve:

- dict: Diccionario con los registros solicitados.

B.2.12.2 Método read

Uso: read(register)

Función para leer un registro del microcontrolador.

Argumentos:

- register (str): Cadena de texto con el nombre del registro que deseamos.

Devuelve:

- int: El valor del registro solicitado.

Excepciones:

- RegistersError:
 - o REGISTERS:READ:01: El registro solicitado no es válido.
 - o REGISTERS:READ:02: El registro solicitado no es accesible.

B.2.12.3 Método write

Uso: write(register, value)

Función para escribir un registro del microcontrolador.

Argumentos:

- register (str): Cadena de texto con el nombre del registro que deseamos. value: Valor que deseamos introducir en el registro.

Devuelve:

- bool: True si se ha realizado correctamente la escritura.

Excepciones:

- RegistersError:
 - o REGISTERS:WRITE:01: Los registros IPSR y/o EPSR son de sólo lectura.
 - o REGISTERS:WRITE:02: El registro solicitado no es válido.
 - o REGISTERS:WRITE:03: El valor a escribir no es un número válido.
 - o REGISTERS:WRITE:04: El valor a escribir está fuera del rango accesible.

B.2.13 Clase Utils

Constructor: Utils()

Clase que contiene unas funciones para ayudar en tareas básicas.

B.2.13.1 Método format2Decimal

Uso: format2Decimal(number)

Función para transformar cualquier número introducido en un número decimal. El número puede ser en introducido como número o cadena de texto, y en formato binario, octal, decimal o hexadecimal.

Argumentos:

- number: Número que se desea transformar en decimal.

Devuelve:

- int: Un número en decimal.

Excepciones:

- UtilsError:
 - o UTILS:F2D_x:01: Se ha introducido un valor en un formato binario, octal, decimal o hexadecimal; que no es en realidad de ese formato. x puede valer B, O, D o H según el número introducido.

- o UTILS:F2D:02: El número introducido no es numérico.

B.2.13.2 Método naturalKey

Uso: naturalKey(key)

Función para ordenar texto de un modo natural.

Orden normal	Orden natural
a1	a1
a10	a2
a11	a3
a2	a10
a3	a11

Argumentos:

key: Claves para ordenar.

Devuelve:

Las claves ordenadas.

B.2.13.3 Método stringB2H

Uso: stringB2H(number)

Función para transformar una cadena de texto que represente un número binario a una cadena de texto que represente el mismo número en hexadecimal.

Argumentos:

- number: Cadena de texto con un número representado en binario.

Devuelve:

- str: El mismo número que el introducido en representación hexadecimal.

Excepciones:

- UtilsError:
 - o UTILS:STR_B2H:01: El valor binario introducido no es en realidad número binario.

B.2.13.4 Método toBin

Uso: toBin(x, sizeBits)

Función para transformar cualquier número introducido en una cadena de texto con el número representado en binario. El número puede ser introducido como número o cadena de texto.

Argumentos:

- x: Número que se desea transformar en binario.
- sizeBits (opcional): el tamaño de bits para nuestro número binario.

Devuelve:

- str: Cadena de texto con el número representado en binario.

Excepciones:

- UtilsError:
 - o UTILS:TO_BIN:01: El argumento sizeBits no es numérico.
 - o UTILS:TO_BIN:02: El argumento sizeBits no es numérico.
 - o UTILS:TO_BIN:03: El argumento x no es numérico.

B.2.13.5 Método toHex

Uso: toHex(x, sizeBytes)

Función para transformar cualquier número introducido en una cadena de texto con el número representado en hexadecimal. El número puede ser introducido como número o cadena de texto.

Argumentos:

- x: Número que se desea transformar en hexadecimal.
- sizeBytes (opcional): el tamaño de bytes para nuestro número hexadecimal.

Devuelve:

- str: Cadena de texto con el número representado en hexadecimal.

Excepciones:

- UtilsError:
 - o UTILS:TO_HEX:01: El argumento sizeBytes no es numérico.
 - o UTILS:TO_HEX:02: El argumento sizeBytes no es numérico.
 - o UTILS:TO_HEX:03: El argumento x no es numérico.