Taller de Sistemas Embebidos Microcontroladores

Información relevante

Taller de Sistemas Embebidos

Asignatura correspondiente a la **actualización 2023** del Plan de Estudios 2020 y resoluciones modificatorias, de Ingeniería Electrónica de FIUBA

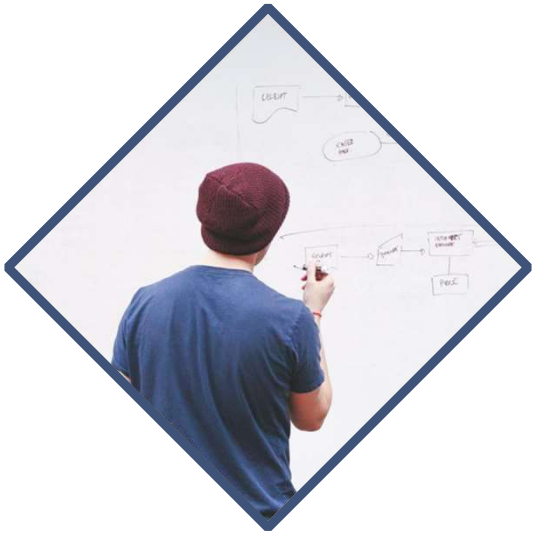
Estructura Curricular de la Carrera

El **Proyecto Intermedio** se desarrolla en la asignatura **Taller de Sistemas Embebidos**, la cual tiene un enfoque centrado en la **práctica propia de la carrera** más que en el desarrollo teórico disciplinar, con eje en la **participación de las y los estudiantes**

Más información . . .

. . . sobre la **actualización 2023** . . . https://www.fi.uba.ar/grado/carreras/ingenieria-electronica/plan-de-estudios

. . . sobre el **Taller de Sistemas Embebidos** . . . https://campusgrado.fi.uba.ar/course/view.php?id=1217



¡Hola!

Soy Juan Manuel Cruz
Taller de Sistemas Embebidos
Consultas a: jcruz@fi.uba.ar

1

Introducción

Actualización 2023 del Plan de Estudios 2020 y resoluciones . . .

Conceptos básicos

- Referencia:
  - Fundamentals of System-on-Chip Design on Arm Cortex-M Microcontrollers - René Beuchat (Author), Florian Depraz (Author), Andrea Guerrieri (Author), Sahand Kashani (Author)
  - Chapter 1: A Memory-centric System Model
    - En este primer capítulo, revisaremos los conceptos básicos de un sistema informático. Los elementos principales son los mismos para todos, desde una computadora central (mainframe) hasta un sistema de control de bajo costo. La mayoría de las diferencias dependen de la escala del sistema, como su tamaño, consumo de energía y potencia informática.

Conceptos básicos

- Este capítulo ofrece una descripción general de estas **arquitecturas** universales, que también encontramos en un sistema en un chip (a veces **system-on-chip**), abreviado **SoC**. Una descripción general de la familia de **procesadores Arm** ayudará a explicar las diferentes terminologías en relación con la línea **Cortex-M**. El capítulo abordará las siguientes preguntas y más.
  - 1. Where can we find microprocessors in the world today and in the near future?
  - 2. What are the basic elements involved in every computer architecture?
  - 3. What is an embedded system?
  - 4. What is a SoC?

Conceptos básicos

- 5. What are the elements of a SoC?
- 6. What is the difference between a processor and a microprocessor?
- 7. What are the main elements found in a processor?
- 8. What is a microcontroller?
- 9. How do we start a new design for an embedded system?
- 10. What elements must be taken into account to realize such a design?
- 11. Where are the instructions to execute?
- 12. What kind of memories are available for an embedded system design?

Conceptos básicos

- 13. What is the internal architecture of SRAM memory?
- 14. What are the main functions of a memory management unit (MMU)?
- 15. What are the main principles around the interrupt controller, the source of interrupts, and programmable interfaces?
- 16. For the core software, do we want to use a kernel, an operating system or nothing (bare metal)?
- 17. What is the CMSIS standard library and why should we use it?
- 18. Do we want to build everything from scratch ourselves?
- 19. Does high-level programming exist for this kind of processor?

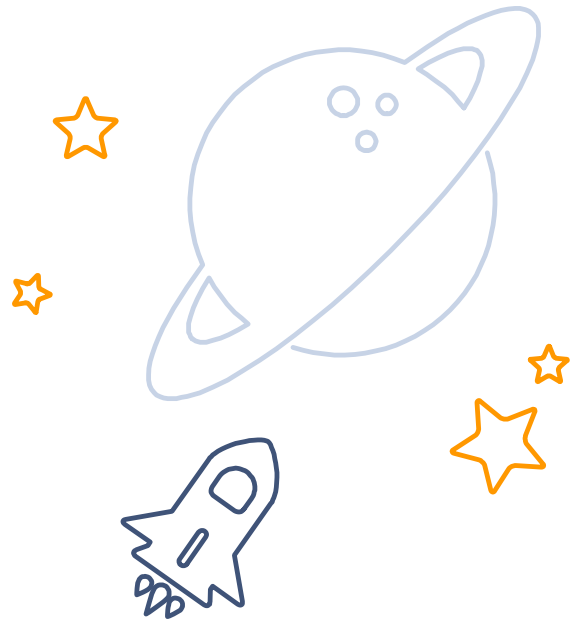
Conceptos básicos

- Desde mediados de los años 1970, las computadoras se han introducido en el uso personal y en los hogares gracias a la invención del microprocesador, empezando por el Intel 4004.
- La ciencia microelectrónica ha progresado hasta alcanzar capacidades casi increíbles en términos de la cantidad de transistores que podrían estar disponibles en un chip, y su tamaño ha disminuido a sólo unos pocos nanómetros en la actualidad.
- A principios de la década de 1980, todos los componentes básicos de una computadora estaban integrados en un solo chip que contenía un microprocesador, memoria e interfaces programables para comunicarse con el mundo exterior.

Conceptos básicos

- Un sistema en un chip (system-on-chip) es básicamente todo lo necesario para implementar una computadora incorporada en un solo chip.
- En este capítulo se describe la jerarquía de un entorno informático moderno, desde los sistemas globales que abarcan todo el mundo hasta los pequeños equipos individuales que se encuentran en un teléfono inteligente, una computadora personal o una unidad del llamado Internet de las cosas (IoT).
- Todo se puede interconectar . . .

Solución Adecuada



. . . lo más simple posible, previa determinación del objetivo de excelencia a cumplir, obviamente contando con la documentación debida y recurriendo a la metodología de trabajo adecuada

2

Documentación debida

1er Cuatrimestre de 2024, dictado por primera vez . . .

“

Fundamentals of System-on-Chip Design on Arm Cortex-M Microcontrollers - René Beuchat (Author), Florian Depraz (Author), Andrea Guerrieri (Author), Sahand Kashani (Author)

Chapter 1: A Memory-centric System Model

Indice

Indice

1.1 Global Computing Systems

- Hoy en día, la **informática** está **omnipresente** en todo el mundo, con wide area networks (WAN), cloud computing, global servers, portable computers, tablets y smartphones en todas partes (ver Figura 1.1).
- La expansión del **Internet de las Cosas** es una de las nuevas revoluciones tecnológicas, así como la forma en que las vidas humanas están cada vez más integradas con las capacidades informáticas.
- La evolución también se está produciendo rápidamente en los campos de la **robótica**, los **vehículos autónomos** y la **inteligencia artificial**, con muchas áreas de investigación y numerosos productos industriales nuevos.

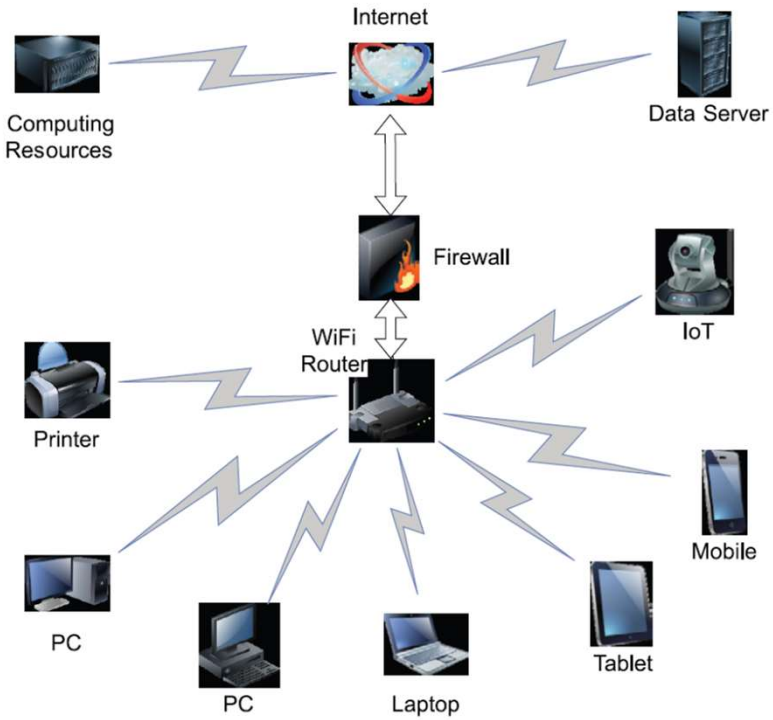


Figure 1.1 High-level component view of a global computing system

1.1 Global Computing Systems

- Sin embargo, la mayor parte de esta tecnología está oculta al usuario humano promedio.
- Los servidores grandes están alojados en edificios exclusivos con sistemas de refrigeración y pueden colocarse bajo tierra o incluso bajo el mar.
- Normalmente, tendrán miles de unidades de procesamiento en forma de microprocesadores junto con enormes cantidades de memoria y aceleradores específicos, como unidades de procesamiento de gráficos (GPU) o field-programmable gate arrays (FPGA) para operaciones especializadas.

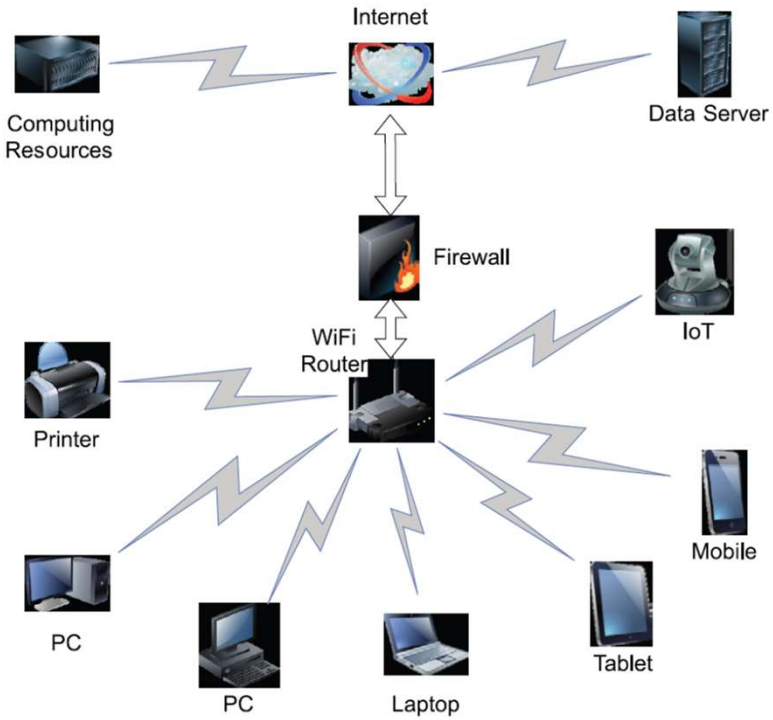


Figure 1.1 High-level component view of a global computing system

1.2 General Computer Architecture

- ¿Qué es una **arquitectura** de **computadora**?
  - Se trata, a nivel de **hardware**, de un sistema compuesto por tres elementos básicos:
    - **Procesador**(es) o **microprocesador**(es) para gestión de datos y ejecución de programas.
    - **Memorias** para almacenamiento de códigos, almacenamiento de **datos** y almacenamiento **operacional** (por ejemplo, stack, heap).
    - Dispositivos de E/S – **I/O**, para la comunicación entre las partes internas y el mundo **exterior** a través de **interfaces** programables.

1.2 General Computer Architecture

- Las conexiones entre estos elementos se realizan a través de **tres buses** principales:
  - El **bus** de **direcciones** deriva de la unidad **solicitante**, como **procesador**, y se utiliza para especificar a qué unidad se solicita acceder y a qué elemento dentro de esta unidad.
  - El **bus** de **datos** se utiliza para **transferir** información (**datos**) entre una unidad **solicitante** y una unidad con que se completará la **transferencia**.
  - El **bus** de **control** se utiliza para especificar **qué hacer**: la **dirección** de las **transferencias** de **datos** (**lectura** o **escritura**) desde el punto de vista de la unidad **solicitante** y **cuándo hacerlo** en términos de tiempo.
- El sistema de **bus** de **hardware** resultante también requiere otras funciones y señales obligatorias, incluido un **reloj** para **sincronizar** las **transferencias** de **datos**, un **reset** para **inicializar** el **sistema**, líneas de **alimentación** y líneas de **interrupción**.

1.2 General Computer Architecture

- La primera generación de computadoras se basaba en **tubos** de **vacío**. Se necesitaban grandes habitaciones para acomodarlos y sus suministros de energía eran enormes; también era necesario reemplazar los tubos con frecuencia.
- Tras la invención del **transistor**, fue posible una nueva generación de computadoras, más **confiables** y **rápidas**.
- Estos fueron posteriormente sustituidos incorporando muchos transistores en un solo circuito: los **circuitos integrados** de los años **60**.
- Esto dio origen a una unidad informática conocida como **procesador**, que utilizaba muchos componentes, que podían tomar la forma de **placas** completas o incluso ocupar habitaciones enteras.
- Con el aumento de la densidad de los transistores, todas estas piezas estuvieron disponibles en un **único chip** de **silicio** o "**die**": el **microprocesador**.

1.2 General Computer Architecture

- La Figura 1.2 es una vista general de la **arquitectura** de una computadora moderna, mostrando sus elementos básicos.
  - Para las **interfaces** programables, se ilustran tres ejemplos simples, que proporcionan una interfaz para dispositivos de **visualización** de datos o diferentes **sensores/actuadores**, una interfaz de **red** (por ejemplo, Ethernet) y el control de un **motor**.

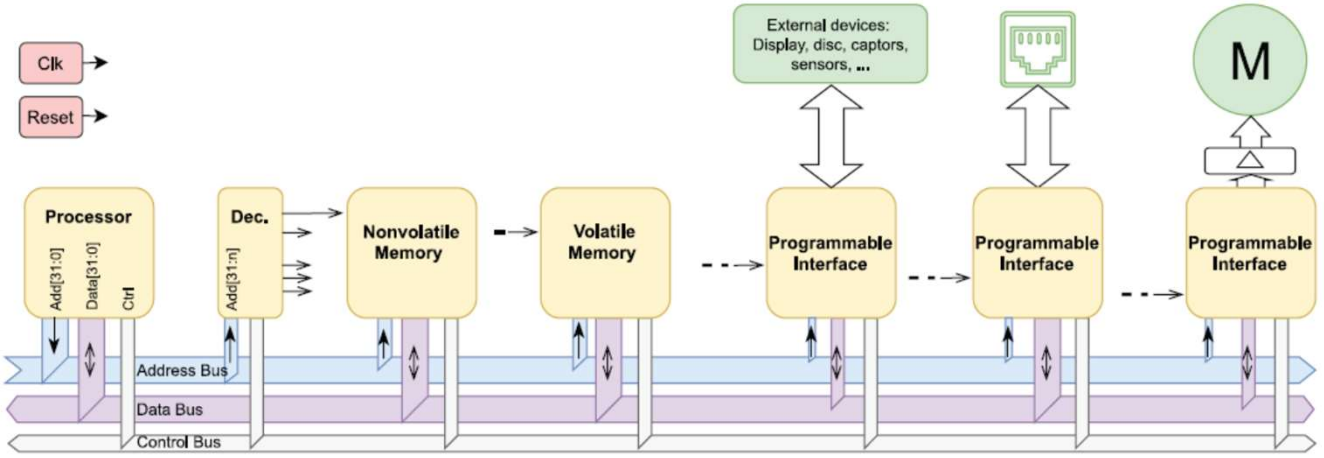


Figure 1.2 General computer architecture with minimum of elements (global hardware view; Dec. = Decoder)

1.2 General Computer Architecture

- El **procesador** proporciona una **dirección** para **seleccionar** un **dispositivo** e indica una **posición** de **memoria** o un **registro** interno de **interfaz** programable.
- El **bus** de **control** se utiliza para proporcionar la **dirección** del **acceso**, ya sea de **lectura** o de **escritura**.
- El **dispositivo seleccionado** aceptará los **datos** proporcionados por el **procesador** (en un ciclo de **escritura**) o proporcionará los **datos** solicitados en el **bus** de **datos** (para un acceso de **lectura**).

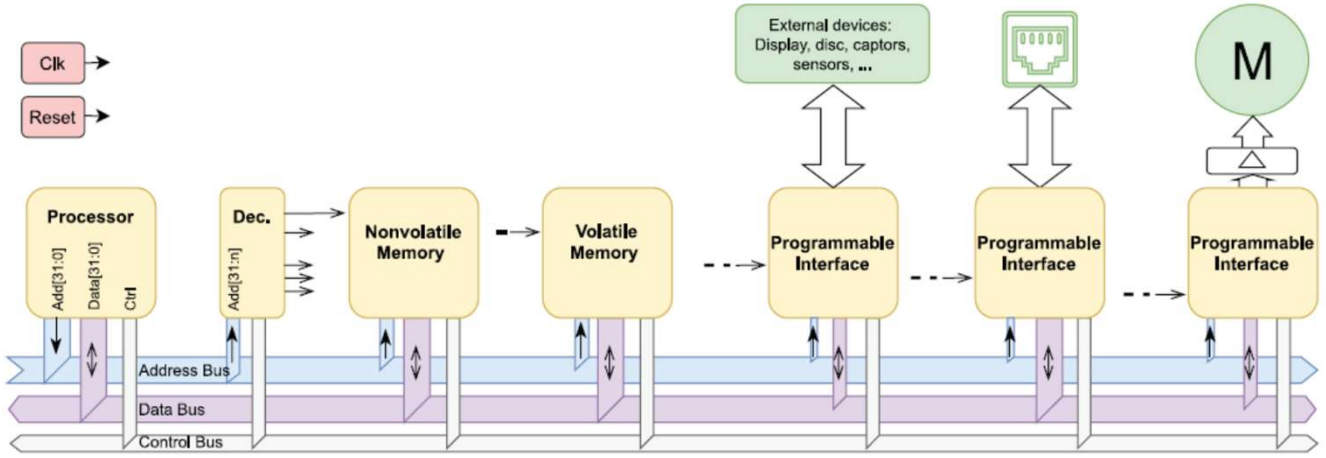


Figure 1.2 General computer architecture with minimum of elements (global hardware view; Dec. = Decoder)

1.2 General Computer Architecture

- La **dirección** de **escritura** o **lectura** siempre se toma desde el punto de vista del **solicitante**, en este caso el **procesador**.
- El **decodificador** recibe la **parte superior** de la **dirección** y puede **seleccionar** una **unidad** de memoria **específica** o una **interfaz** programable **para acceder**; genera una **señal** llamada **Chip Select**) para permitir el **acceso** exclusivo a **un dispositivo** a la vez.

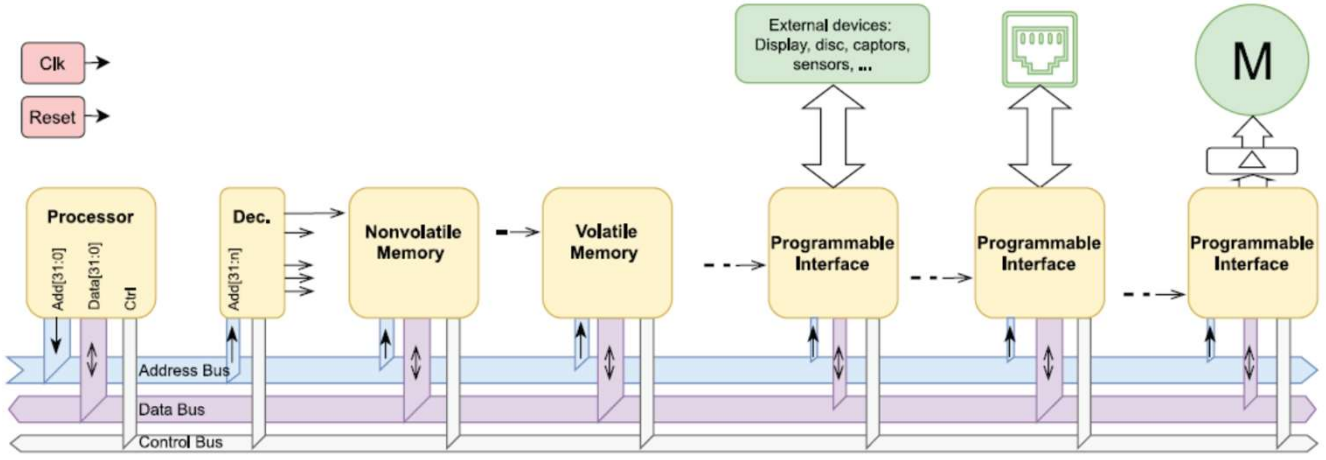


Figure 1.2 General computer architecture with minimum of elements (global hardware view; Dec. = Decoder)

1.2 General Computer Architecture

- La vista de la Figura 1.2 carece de detalles de las señales utilizadas o de las conexiones de los pines.
  - Estos elementos son muy específicos de los dispositivos reales involucrados. Físicamente, los buses constan de muchos cables. **Transportan información** en momentos concretos y, en general, de forma **síncrona** mediante señales de **reloj**. Están definidos por especificaciones y todos los elementos del bus deben respetar los **protocolos** definidos en términos de **niveles lógicos** (**tensiones**) y **temporización** utilizados en el **bus**.

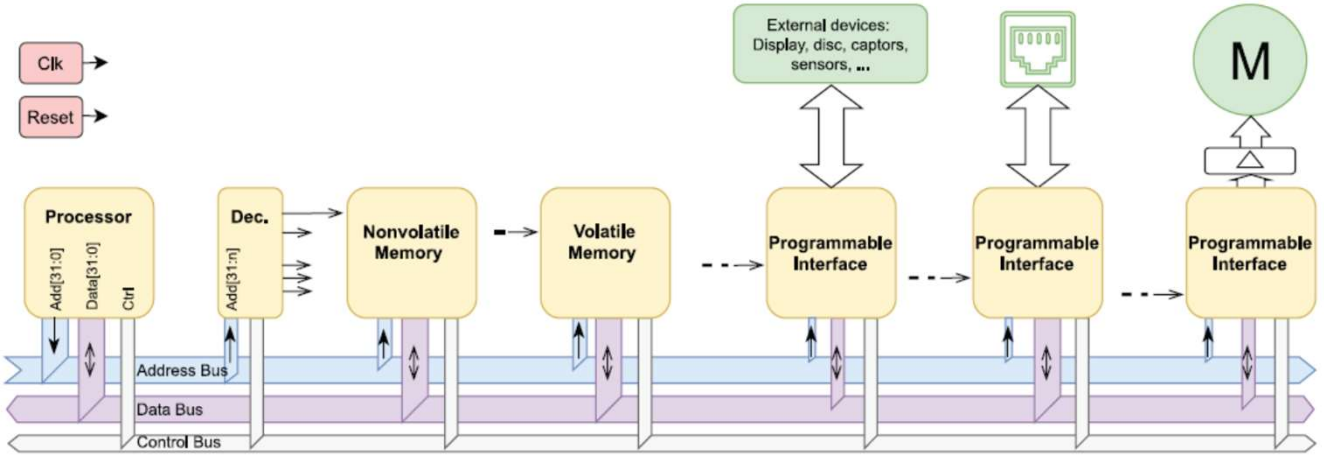


Figure 1.2 General computer architecture with minimum of elements (global hardware view; Dec. = Decoder)

1.2 General Computer Architecture

- En una computadora general, los elementos electrónicos son específicos y están separados en diferentes circuitos integrados. Sin embargo, la creciente miniaturización significa que la mayoría, si no todos, ahora pueden estar disponibles en un solo chip: un **microcontrolador**.
- **Arm's Cortex-M** es un **microprocesador** que constituye la **base** de muchos **microcontroladores** del mercado actual. No es el único y la mayoría de los fabricantes de procesadores ofrecen ahora diferentes familias de productos relacionados, más o menos especializados. Estas arquitecturas de procesador incluyen x86, AVR, PIC, RISC-V, ARC y MIPS32, entre las más conocidas.
- **Arm** ha desarrollado sus **microprocesadores** como **arquitectura IP** (**propiedad intelectual**) y los vende a cientos de fabricantes, quienes los utilizan para desarrollar sus propios **microcontroladores** con **interfaces** y **memorias** programables **específicas**. Pueden aprovechar la disponibilidad de una amplia gama de **recursos compartidos**, como **herramientas** de **desarrollo**, **lenguajes** y sus **compiladores**, **sistemas operativos**, **depuradores**, etc.

1.3 Embedded System Architectures

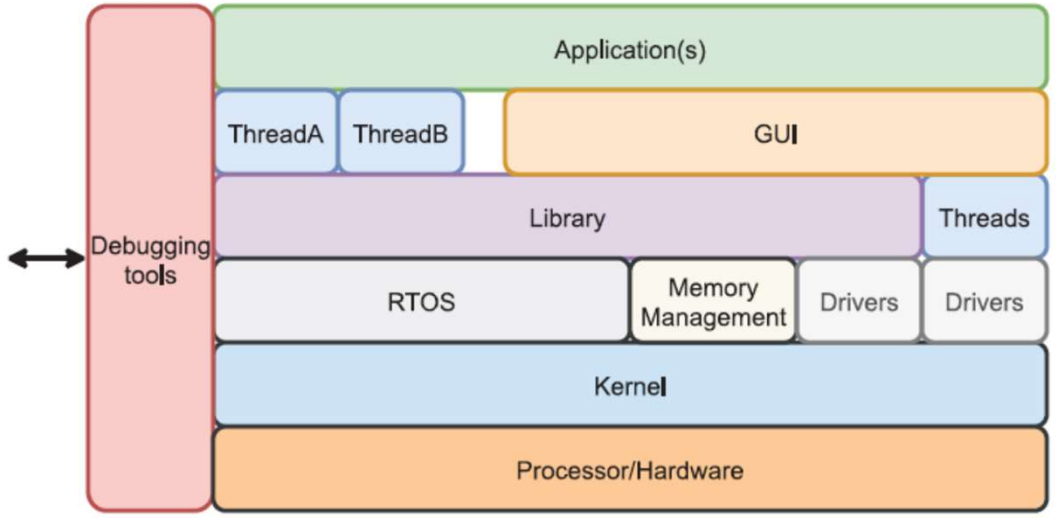
- Un **sistema embebido** es una computadora especializada diseñada para una **aplicación específica**. Estos sistemas suelen diseñarse teniendo en mente un gran mercado para reducir el **costo** de producción, y están diseñados desde el nivel de **hardware** hasta el nivel de **software**. Se producen con o sin sistema operativo o kernel de tiempo real y tienen la misma arquitectura general que se ve en la Figura 1.2. Sin embargo, si bien la visión global es similar, los detalles son diferentes.
- Un caso similar es el de un sistema especializado que necesita una potencia de cálculo muy alta y está sujeto a restricciones en el tiempo de ejecución, teniendo una ventana limitada para responder a los eventos que requieren cómputo. Sin embargo, todos se basan en una **arquitectura común**, pero con **diferentes** escalas en términos de **consumo** de **energía**, **capacidades** de **cálculo**, **capacidades** de **memoria** y **disponibilidad** de **interfaces**. A partir de 2020, el número de procesadores interconectados sigue aumentando y pueden ubicarse en un solo chip (**multinúcleos**), en una placa (**multiprocesadores**) o distribuirse **localmente** o por todo el **mundo**.

1.3 Embedded System Architectures

- Se pueden encontrar ejemplos de sistemas embebidos a nuestro alrededor:
  - Cafeteras con paneles para el control de la máquina
  - Sistemas de frenos antibloqueo (ABS) en automóviles
  - Relojes deportivos con sensores de salud
  - Cámaras
  - Sistemas de calefacción automatizados con sensores y reguladores de temperatura.
  - Unidades de disco duro
  - Robots móviles (o estáticos)

1.3 Embedded System Architectures

Sin embargo, el **hardware** por sí solo no es suficiente. Además, debemos tener una aplicación de **software** basada en un kernel y/o sistema operativo simplificado (a menudo llamado sistema operativo de tiempo real o **RTOS**) que gestione todos los recursos, el **acceso** a los **periféricos**, el **acceso** y **protección** de la **memoria**, **gestión** del **tiempo** y **planificación** de **tareas**. Por encima de este **software** de bajo nivel pueden encontrarse **bibliotecas** de funciones útiles, como interfaces gráficas de usuario (**GUI**) para la visualización de información y la **interacción** del **usuario**, implementaciones de **protocolos** de **comunicación** de red y **procesamiento** de **señales**.



s in an embedded system

1.3 Embedded System Architectures

En el nivel superior están las **aplicaciones** mismas, como se muestra en la Figura 1.3; la **jerarquía** permite que cualquier nivel llame a un nivel inferior según sea necesario.

Una parte final e importante de la **arquitectura** es el **soporte** para la **depuración** y el **control interactivo** con la ayuda de **hardware** especializado y **herramientas** de desarrollo/depuración.

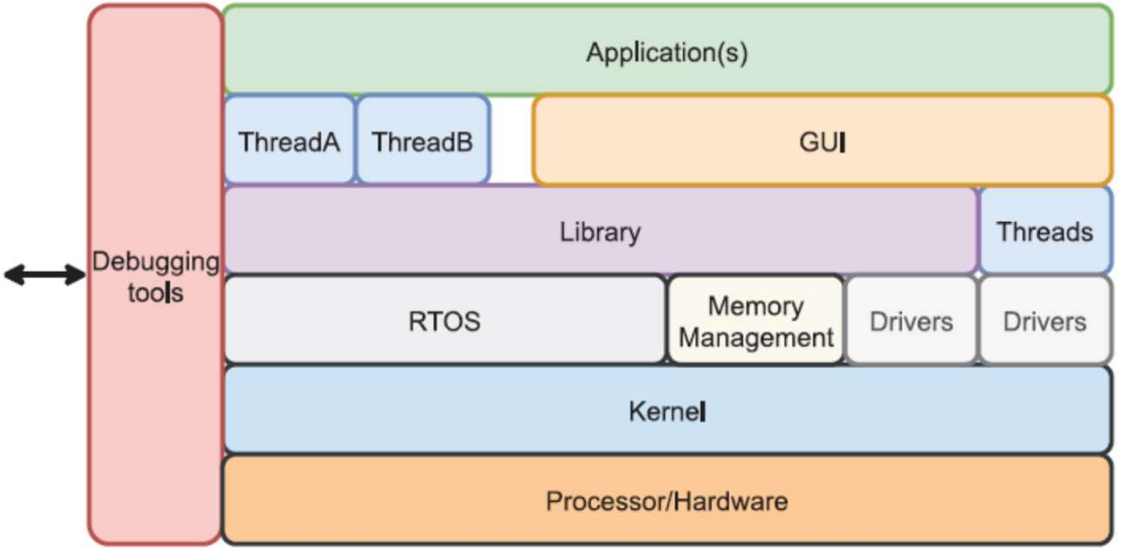


Figure 1.3 Software levels in an embedded system

1.4 Embedded System-on-Chip Architectures

- Un sistema en un chip es simplemente todo elemento de una computadora en un único circuito integrado.
- Sin embargo, en general, los SoC no pueden alojar suficiente memoria para un sistema operativo completo como Linux o Android y, por lo tanto, a menudo, se debe proporcionar algo de memoria externa, al propio SoC. De esta forma se suelen añadir unos pocos gigabytes.
- Actualmente, esta técnica se utiliza en la familia de procesadores Cortex-A, pero ¿por qué no también en la familia Cortex-M en un futuro próximo?
- Los SoC de alta gama pueden incluir unidades de procesamiento de gráficos (GPU), módulos inalámbricos para Wi-Fi, Bluetooth y LoRa, y sensores médicos y ambientales.

1.4 Embedded System-on-Chip Architectures

- Los **microcontroladores** comenzaron con el **TMS1802NC** (**1971**) de **Texas Instruments** de la familia TMS 0100, y más tarde con el famoso **i8048** (**1976**) de **Intel**, seguido del **i8051** (**1980**), todavía disponible hoy como microcontrolador de 8 bits y líder durante muchos años en esta área de electrónica.
- Ahora hay muchos actores en el mercado. Muchos de ellos proporcionan su **propia arquitectura patentada** para sus procesadores y/o las interfaces programables que los acompañan.
- Pero casi todos también ofrecen la **arquitectura Arm** en su cartera. ¿Por qué?
- Bueno, consideremos la perspectiva de un usuario típico: el diseñador o arquitecto de un nuevo producto.
  - El o ella es parte de una **gran empresa** con una **inversión existente** en **herramientas** de desarrollo y **conocimiento** que no puede darse el lujo de cambiar para cada proyecto.
  - Una vez seleccionada una **familia** de **procesadores**, ésta se **utilizará** durante **muchos años**.
  - Si la **empresa** ofrece diferentes **arquitecturas**, pero mantiene la **compatibilidad** a nivel de **procesador**, será más **fácil adaptarse** y **evolucionar** si se pueden aplicar los **mismos conocimientos** básicos y **herramientas** de desarrollo.

1.4 Embedded System-on-Chip Architectures

- Muchos **fabricantes** ofrecen **herramientas** que se pueden **utilizar** para **todos** sus diferentes **procesadores**, tengan o no la misma arquitectura de procesador: el **usuario** está **contento** y puede moverse entre ellos para utilizar el más compatible con una aplicación específica. De manera similar, muchos **fabricantes** ofrecen **interfaces** programables **iguales** o muy similares con muchos procesadores y configuraciones de memoria diferentes.
- Así, tenemos diferentes capas de **usuarios**:
  - Los **usuarios** de **aplicaciones** quieren un **producto terminado**, potente en términos de lo que quieren hacer con él. En general, salvo unos pocos verdaderos entusiastas (**geeks**), no les importa la tecnología, aunque si se trata de un smartphone, ¡muchos quieren lo último!
  - Se espera que los diseñadores de productos realicen un producto, basándose en lo que está disponible en el mercado, como un circuito integrado. Pueden elegir entre miles de dispositivos de cientos de fabricantes. Los niveles de rendimiento e integración dependen de lo que esté disponible. Este tipo de **usuario** trabaja al **nivel** de la **placa** de circuito impreso (**PCB**).

1.4 Embedded System-on-Chip Architectures

- Diseñadores del propio SoC, que pueden trabajar en muchas empresas diferentes, dependiendo de la empresa que fabrica los microcontroladores. Reciben una solicitud de un nuevo producto y deben realizarlo según diferentes criterios. Estos podrían incluir la elección del procesador, el tipo y cantidad de memoria, las interfaces programables que deben agregarse, los aceleradores necesarios, etc. Pueden ser ingenieros que se especializan en el diseño de circuitos integrados y que deben diseñar un circuito en particular, como un sensor específico con capacidades de procesamiento y comunicación.
- Diseñadores IP de la arquitectura del procesador que necesitan crear una nueva generación del mismo o desarrollar módulos especializados. Los diseños resultantes se venderán a los integradores existentes de los microprocesadores (los diseñadores de productos y SoC).
- En el nivel más bajo de integración, tenemos a las empresas de silicio, quienes deben proporcionar las mejores capacidades en términos del (mayor) número de transistores y/o el (menor) tamaño asociado con el menor precio y consumo de energía.

1.4 Embedded System-on-Chip Architectures

- Hoy en día, uno de los actores principales en IP de microprocesadores es Arm, con sus familias Cortex, A, R y M.
- La última de ellas es la que se considera específicamente en este libro. Muchas versiones de la arquitectura están disponibles y Arm las proporciona como IP (propiedad intelectual) lista para fabricar, con todos los elementos de una computadora disponibles en un solo chip.
- Los fabricantes de microcontroladores deben agregar las interfaces programables y seleccionar las opciones adecuadas entre las que se ofrecen.
- Para proporcionar enfoques adicionales al diseño, hay disponibles en el mercado módulos con conectores estándar con muchas características y componentes ya incorporados; estos pueden denominarse CoM (computers on modules) o SoM (systems on modules).
- Algunos ejemplos conocidos incluyen Arduino, Raspberry Pi, BeagleBoard, Toradex e iWave, y muchos fabricantes proporcionan kits de desarrollo que permiten a los usuarios comenzar a usar sus microcontroladores antes de seleccionar un microprocesador específico y su PCB.

1.5 Elements of SoC Solutions

- Cuando un ingeniero tiene que diseñar una nueva solución SoC y ha considerado las cuestiones relevantes destacadas en la sección anterior según lo que se requiere que haga el producto, debe pensar en los elementos clave que se utilizarán y/o desarrollarán.
- En primer lugar, suele ser bueno establecer un primer sistema funcional como prototipo para verificar la funcionalidad del dispositivo esperado.
- A esto a veces se le denomina producto mínimo viable (MVP). Para ello, una primera versión creada con un kit de diseño de software (SDK), en lugar de cualquier hardware, puede resultar muy útil y se recomienda.
  - Esto puede permitir la verificación de la cantidad de memoria necesaria para el proyecto, el tiempo de respuesta del sistema y mucha otra información útil.
  - El producto final no dependerá de un SDK, pero un SDK ayudará en su diseño.

1.5 Elements of SoC Solutions

- Si es demasiado trabajo diseñar una placa de circuito impreso (PCB) específica para el producto, es posible optar por un sistema basado en CoM o SoM.
- Para comenzar el diseño o la integración de un nuevo sistema, un ingeniero debe hacer preguntas sobre:
  - potencia informática necesaria, que dependerá en gran medida de los algoritmos utilizados en la aplicación y de cualquier unidad/dispositivo adicional requerido
  - energía eléctrica necesaria, que dependerá de la aplicación, dónde se ubicará el sistema y qué fuentes de energía estarán disponibles
  - uso de energía de la batería o de la red eléctrica/alimentación externa
  - ancho de banda de los datos a gestionar y/o transferir, y el número de operaciones por segundo involucradas

1.5 Elements of SoC Solutions

- comunicaciones necesarias:
  - inalámbrico/RF (por ejemplo, Bluetooth, Wi-Fi, LoRa, SigFox, ZigBee, GSM, 3G, 4G, 5G, etc.)
  - cableado (por ejemplo, Ethernet 10/100/1G, bus CAN, USBxx, etc.)
- Tipo y tamaño de memoria que se necesitará o se podría esperar
- conectores de extensión:
  - estándar (PCIe, VME, etc.)
  - Serie (UART, USB, etc.)
  - propietarios
  - para depuración (JTAG, SWD, etc.)

1.5 Elements of SoC Solutions

- dispositivos a incluir, como acelerómetros, cámaras, sensores, monitores de temperatura, monitores de corriente, etc.
- ayudas para probar/monitorear hardware:
  - puntos de prueba para osciloscopios y analizadores lógicos
  - Formas de ver el consumo de energía, los niveles de tensión y ripple (tensiones/corrientes)
  - LED
  - Interfaces serie (UART, SPI, i2c, etc.)
  - botón para Reset

1.5 Elements of SoC Solutions

- Obviamente hay muchas preguntas y este libro pretende ayudarle a responder muchas de ellas.
- Como mínimo, será **consciente** de los **problemas potenciales** y estará bien posicionado para intentar encontrar **soluciones adecuadas**.
- Hacer la **pregunta correcta** ayuda a encontrar una **respuesta inicial** y, finalmente, la **respuesta correcta**.
- Recuerde, debe **leer** atentamente la **documentación completa** y las **hojas** de **datos** para ver cómo funcionan realmente los componentes.
- Ocasionalmente, la **documentación** tiene **errores** pero, igualmente, ¡**a veces** las características simplemente **han cambiado**!

1.5.1 Processor Cores

- Hay muchas **familias** de **microprocesadores** disponibles en el mercado. Uno de los principales criterios de selección es el **tamaño** del **bus** de **datos**, en términos de 4, 8, 16, 32 o 64 bits de **data path**. Otro es el **tamaño** del **bus** de **direcciones** necesario para acceder al volumen de memoria que se necesitará.
- Muchas **arquitecturas** conocidas se han desarrollado desde la década de **1970**, hace ya medio siglo. Como ingeniero o futuro ingeniero, es bueno estar familiarizado con al menos algunos de ellos. Pueden tener una arquitectura de propósito general o especializada:
  - Familia **i86** (por ejemplo, muchas/la mayoría de las PC), Familia **Arm** (por ejemplo, Raspberry Pi), **PIC**, **AVR** (por ejemplo, Arduino), basado en **i8051**, **RISC V**, **MSP430**, **Xtensa** (por ejemplo, ESP32), **DSP**, conjunto de instrucciones extensible.

1.5.1 Processor Cores

- También existen extensiones especializadas, como:
  - unidades de punto flotante (FPU), unidades de procesamiento de gráficos (GPU), unidades de gestión de memoria (MMU), extensiones de criptografía (por ejemplo, AES)
- Es probable que haya otros en un futuro próximo; por ejemplo, aceleradores de inteligencia artificial.
- En términos de diseño de SoC, también existen extensiones en forma de interfaces programables de muchos tipos y extensiones para FPGA y dispositivos lógicos programables complejos (CPLD).

1.5.1 Processor Cores

- ¿Cuál es la cantidad mínima de **unidades** dentro de un **procesador**? La Figura 1.4 es un **diagrama** de **bloques** de una **arquitectura** de **procesador** muy simple. Casi todos los **procesadores** tienen esto como mínimo. Consideremos sus elementos con más detalle.
- El **trabajo** de un **procesador** es **manipular datos**. Para ello, **ejecuta** un **programa**, que es simplemente una **lista** de **instrucciones** básicas. Muchas de estas **instrucciones** están **disponibles** en todos los **procesadores**, en forma de **operaciones lógicas** y **aritméticas** simples. La unidad que las **ejecuta** es la unidad aritmética y lógica (**ALU**) y las operaciones son las clásicas **SUMA**, **RESTA**, a veces **MULtiplicación** y **DIVisión**, **AND**, **OR**, **EOR** lógico (OR exclusivo, a veces denominado **XOR**), y **NOT**, junto con **SHIFTing**. y **ROTación**. Estos dos últimos pueden funcionar en **un** solo **ciclo** o en **múltiples ciclos** mediante un **barrel shifter**.

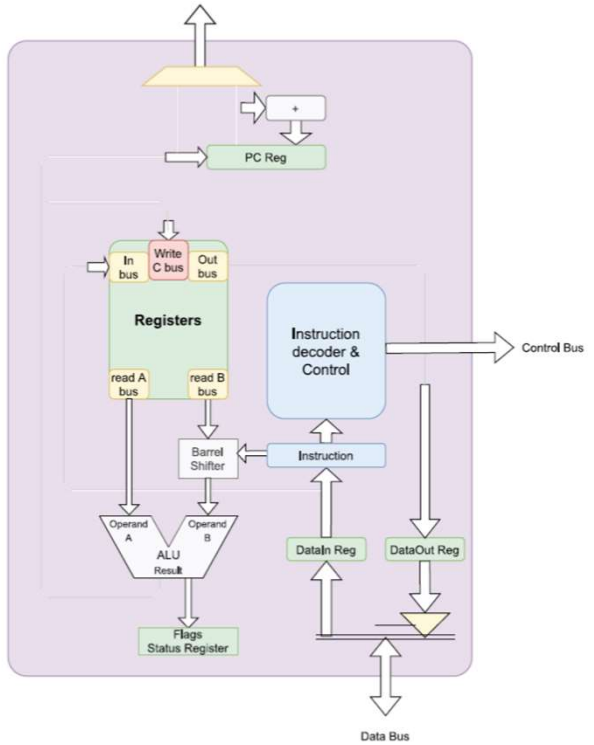


Figure 1.4 Simple processor architecture

1.5.2 Registers and the Control Unit

- Para **guardar** y **ejecutar** las **operaciones**, se utilizan **registros** como **fuentes** y **destinos** de **datos**.
- Estos podían ser **simples** y **únicos**, como fue el caso de los primeros procesadores, en los que se denominaban **acumuladores**.
- En los **procesadores modernos** puede haber **16**, 32 o incluso más (ver, por ejemplo, Figura 1.5). De hecho, dependiendo del modo del procesador, pueden existir muchos **bancos** de **registros**.
- Para **escribir** o **leer** el contenido de estos **registros**, el **acceso** se realiza a través de uno o más **buses** de **datos** internos.

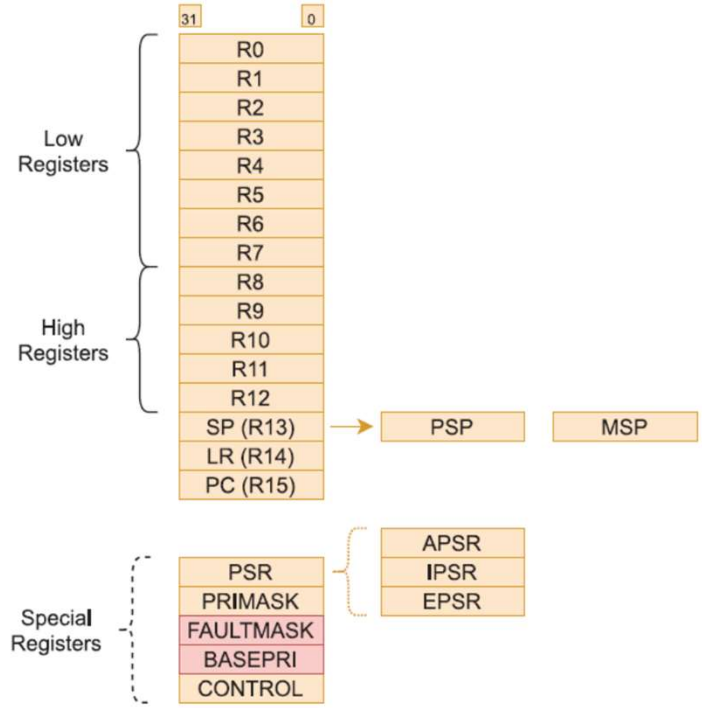


Figure 1.5 View of a processor's internal registers, here the Cortex M0

1.5.2 Registers and the Control Unit

- Un registro particular es el **registro** de **flags** o registro de **estado** (ver, por ejemplo, la Figura 1.6). Este retiene el estado de algunas de las operaciones de la **ALU**, principalmente las **aritméticas**.
- En general, las **operaciones con signo** se realizan en números con signo utilizando operaciones **en complemento a dos** (consulte el Anexo 1 para un repaso sobre la numeración binaria).
- Casi todos los procesadores utilizan los siguientes indicadores de estado:

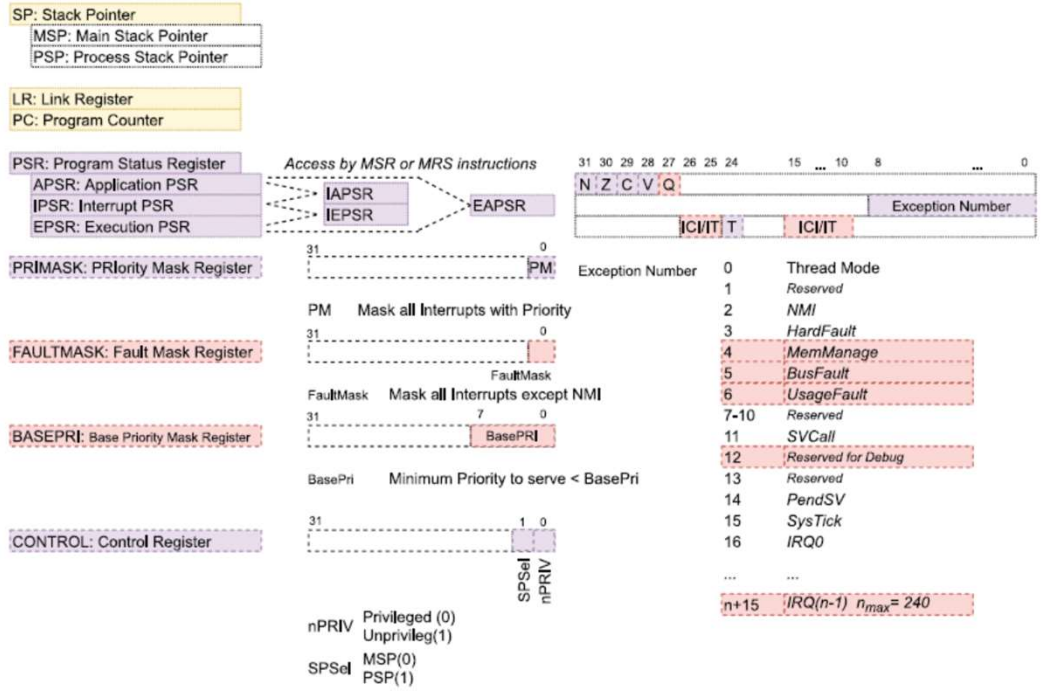


Figure 1.6 Status register and some specific registers

1.5.2 Registers and the Control Unit

- N para Negative. Este es el bit más significativo del resultado. Si es igual a 1, significa que el número es negativo; de lo contrario, es positivo y se codifica en números con signo en complemento a dos.
- Z por Zero. Esto significa que el resultado tiene todos los bits en 0 y el valor del registro es igual a 0.
- C por Carry. Cuando se suman o restan dos números binarios, podemos tener un acarreo. Es como tener un bit más en el registro de resultados. Este bit de resultado suplementario se convierte en el flag de acarreo.
- V para oVerflow. Cuando una operación con signo involucra dos números con signo, si se suman dos números positivos y el resultado es negativo, es un error y se activa el flag V. De manera similar,

1.5.2 Registers and the Control Unit

- cuando se suman dos números negativos y el resultado es positivo. ¿Extraño? Sí, en el mundo decimal con dígitos ilimitados esto no ocurre. Pero en un procesador, el número de bits de un número siempre está limitado (por ejemplo, a 8, 16, 32 o 64).
- Q es un **flag** introducida con **aritmética saturada**. En lugar de tener un overflow, la operación proporciona como resultado el mayor número positivo o el mínimo negativo. El resultado no es correcto, pero es mejor que el que se puede obtener simplemente con el flag de overflow. El procesador **Cortex-M3** tiene este flag.

Es **responsabilidad** del **programador comprobar** si se ha producido un desbordamiento o un acarreo y **validar** o **invalidar** el **resultado** en consecuencia.

1.5.2 Registers and the Control Unit

- Las **instrucciones** se **codifican** en la **memoria** como **código binario**.
- Tienen un **formato** específico y dependen de la familia de **procesadores**.
- Un **código binario determinado** sólo se puede **ejecutar** en una **arquitectura** de **procesador específica**. En general, cada nueva familia de procesadores **agrega** algunas **instrucciones** o **cambia** la **codificación**, o incluso el conjunto de instrucciones completo.
- A esto nos referimos como Instruction Set Architecture (**ISA**). Suele ser **propiedad intelectual** de la empresa que lo definió.

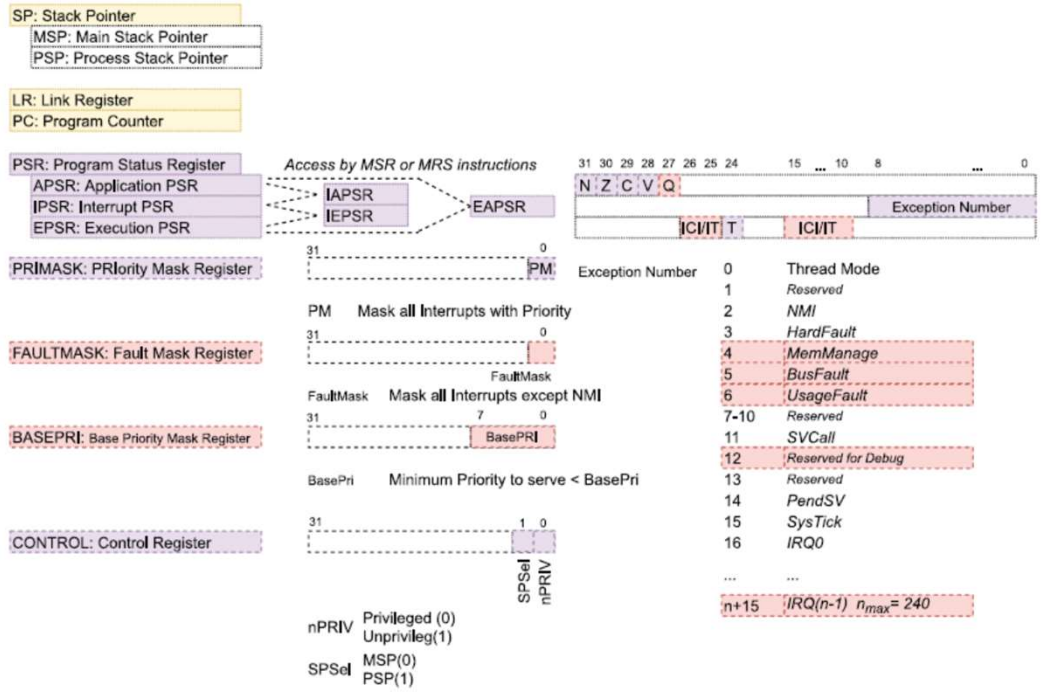


Figure 1.6 Status register and some specific registers

1.5.2 Registers and the Control Unit

- El **código** se **almacena** en una **memoria** externa al propio procesador, pero puede estar en el mismo chip.
- Para **acceder** al **código actual** es necesario saber dónde está la **siguiente instrucción**, **leerla**, **decodificarla** y **ejecutarla**. Se utiliza un **registro** específico para **proporcionar** esta **dirección** y se denomina contador de programa (**PC**).
- Su **contenido** se pasa al **bus** de **direcciones** externo para **seleccionar** la **instrucción** de la **memoria**. Normalmente, la **siguiente instrucción** se encuentra en la **siguiente dirección**. El **PC** se **incrementa automáticamente** según el **tamaño** de la **instrucción**.

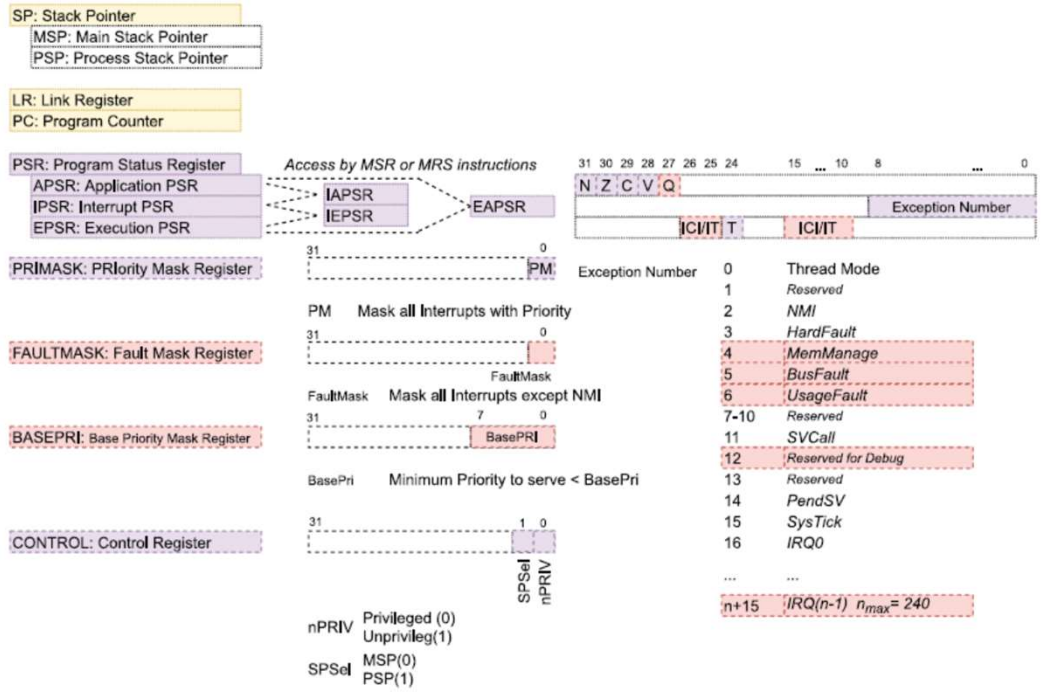


Figure 1.6 Status register and some specific registers

1.5.2 Registers and the Control Unit

- Cuando el programa tiene una bifurcación (**branch** o **jump**), la **nueva dirección** debe cargarse desde el **código** de la **instrucción** o el contenido de un **registro global** o mediante alguna **otra operación**.
  - El **PC** recibe un **nuevo valor**, que es la **dirección** desde la cual leerá la **siguiente instrucción**.
- Los **datos** recuperados de la memoria se **pasan** al **registro** de **instrucciones** para que el **decodificador** y el **controlador** de **instrucciones** los **decodifiquen**.
  - Esta parte muy importante, también denominada **unidad** de **control**, decide qué **hacer**, qué registros **acceder**, **controla** el bus de control externo y todas las secuencias internas, y **especifica** la operación de **ALU** a **ejecutar**.
  - En este esquema simple, la **lectura** y **escritura** de la **memoria** o la **interfaz** programable se **realiza** en la etapa de **ejecución**.

1.5.2 Registers and the Control Unit

- En un **procesador** muy **simple**, cada **paso** de una **instrucción** se **completa** antes de iniciar una nueva, como se muestra en la Figura 1.7. Sin embargo, esto no es muy eficiente en términos de uso de recursos internos.

In a very simple processor, every step of an instruction is completed before a new one is started, as shown in Figure 1.7. However, this is not very efficient in terms of its use of internal resources.

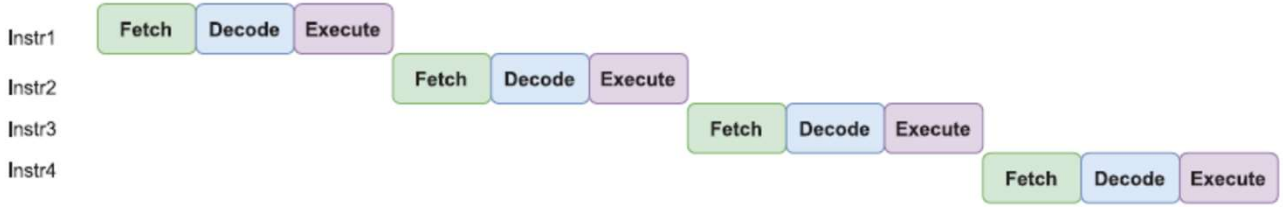


Figure 1.7 A sequence of decoded instructions from loading (fetch) to execution

1.5.2 Registers and the Control Unit

- Un mejor enfoque es hacer lo que se llama **pipelining**; tan pronto como se ha **cargado** una **instrucción** en el **registro** de **instrucciones** interno, se puede recuperar una **nueva instrucción mientras** se **decodifica** la primera.
- Tan pronto como esta **decodificación** haya **terminado**, comenzará su **ejecución** y se podrá iniciar la **decodificación** de la nueva **instrucción**, como se representa en la Figura 1.8.
- Se puede **cargar** una nueva **instrucción** en cada **ciclo** y su **resultado** estará disponible **dos ciclos después**, pero la frecuencia de los ciclos será efectivamente mayor, ya que se pueden completar cuatro instrucciones en el tiempo que antes se necesitaba para completar dos.

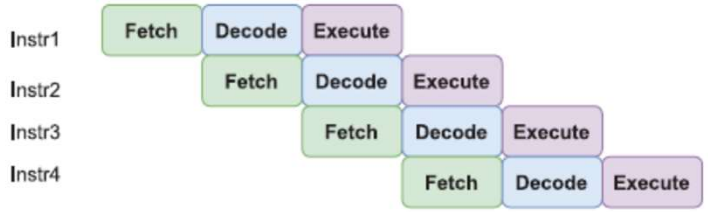


Figure 1.8 Pipelining of fetch, decoding and execution instructions

1.5.2 Registers and the Control Unit

- Sin embargo, este tipo de **paralelismo** deja algunos **problemas** por resolver.
  - Por ejemplo, si se debe ejecutar una instrucción de bifurcación de programa (**Branch** o **Jump**), la **dirección** de la **siguiente instrucción** sólo se conocerá cuando se haya **completado** la **ejecución** de la anterior; en este caso, es necesario **retrasar** la siguiente **recuperación**, **interrumpiendo** el **pipeline** o **cancelando** la **secuencia**, como se ilustra en la Figura 1.9.

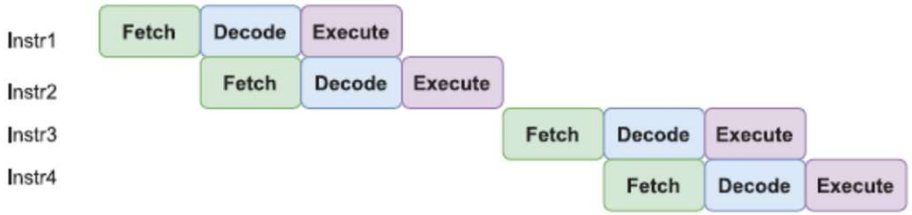


Figure 1.9 Instr2 involves a program jump and breaks the pipeline

- Sin embargo, el beneficio del **pipelining** es positivo porque los programas tienen muchas más instrucciones consecutivas que de bifurcación.
  - Existen algunos métodos que pueden evitar romper completamente el **pipeline**, pero por ahora solo describiremos las características estándar.

1.5.2 Registers and the Control Unit

- Para **transferir datos** dentro y fuera del **procesador**, **dos buffers** específicos acceden al **bus** de **datos externo**. A veces se utiliza un **buffer bidireccional**, con un **buffer tri-state** que permite diferenciar la **dirección** de **transferencia**. Los **datos** pueden ser una **instrucción**, un **acceso** a una **variable** en **memoria** o un **acceso** a una **interfaz** programable; cualquier cosa que pueda **proporcionar** o **recibir datos**.
- Cuando tanto las **instrucciones** como los **datos** están disponibles a través del **mismo** par de **bus** de **direcciones** y **bus** de **datos**, la **arquitectura** se describe como **von Neumann** (llamado así en honor a un matemático que, en 1945, fue vocero de un comité sobre arquitectura de computadoras que propuso los conceptos principales de esta arquitectura: una ALU, una unidad de control, memoria y mecanismos de entrada/salida). Esta arquitectura utiliza un **espacio** de **memoria unificado** que contiene **instrucciones** y **datos**. Se **comparte** el mismo **bus** para leer **instrucciones** y acceder a **datos**. El uso de un bus común reduce las líneas físicas y permite que el bus compartido sea amplio, pero las **transferencias** de instrucciones y datos deben ser **secuenciales**, lo que crea un cuello de botella y **reduce** el rendimiento.

1.5.2 Registers and the Control Unit

- A la vez, las figuras 1.4 y 1.10 representan dicha arquitectura de **von Neumann**.
  - Físicamente, las **memorias** pueden ser **diferentes** pero los mismos **buses** físicos se utilizan de forma **uniforme**.

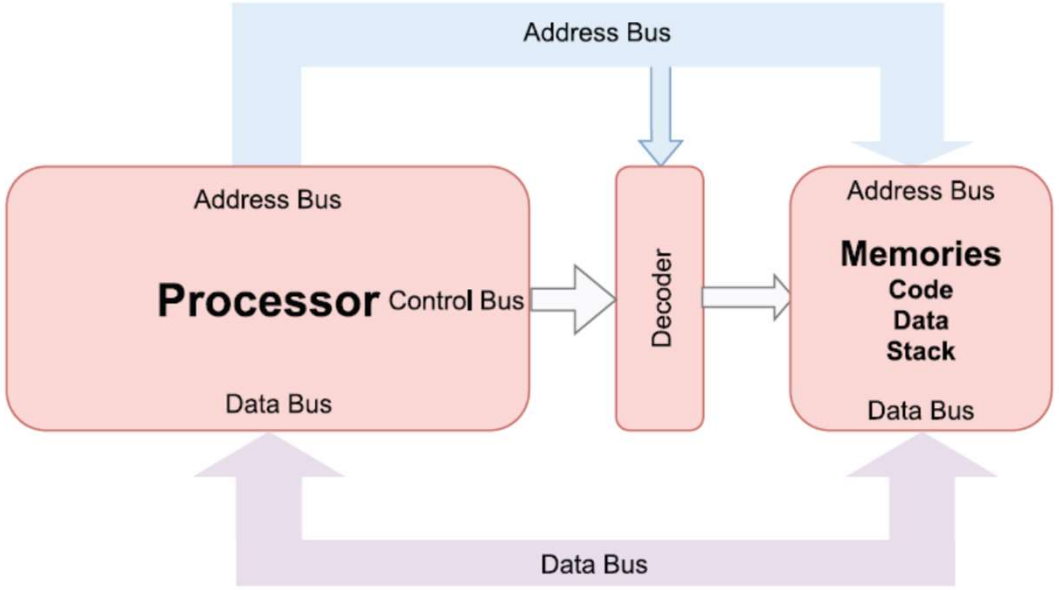


Figure 1.10 Organization of memory and buses in a von Neumann architecture

1.5.2 Registers and the Control Unit

- Una **arquitectura** alternativa, más potente en términos de eficiencia informática, es la **arquitectura Harvard**, representada en la Figura 1.11.
  - Aquí, los **buses** de **direcciones** y **datos** se **duplican** para **separar** el acceso a **instrucciones** y **datos**, lo que tiene un costo en términos de complejidad y líneas físicas.
  - Sin embargo, permite realizar la **lectura** de una nueva **instrucción** al **mismo tiempo** que una **transferencia** de **datos**.
  - En funcionamiento, el **bus** de **instrucciones** es **unidireccional**, desde la memoria al procesador. Esto significa que es necesario encontrar una manera de cargar el código en la memoria si es memoria no volátil, pero esto es un detalle de implementación.

1.5.2 Registers and the Control Unit

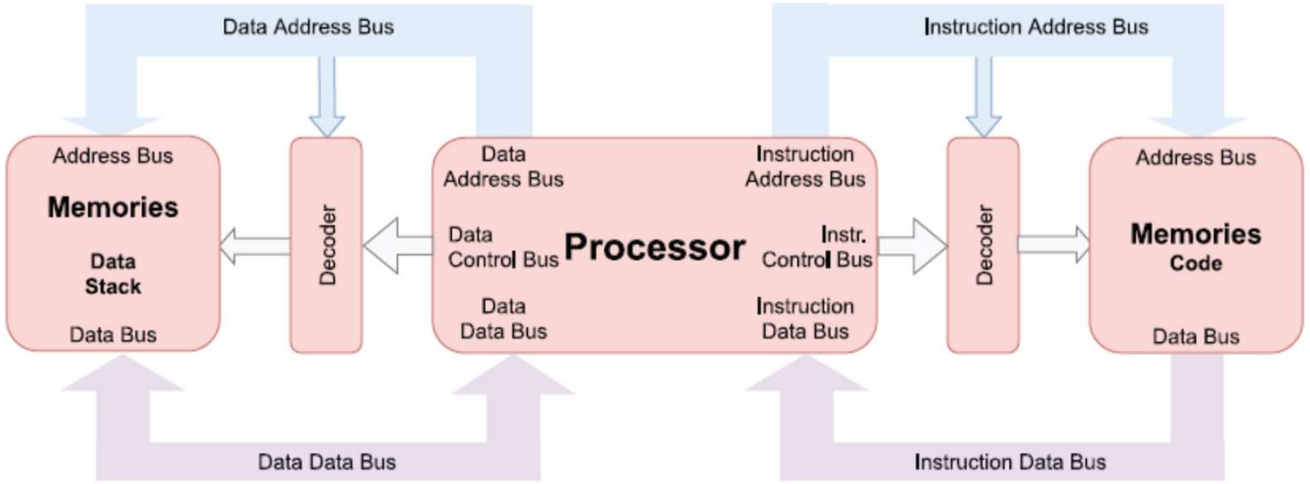


Figure 1.11 Organization of memory and buses in a Harvard architecture

- Hoy en día, ambas **arquitecturas** son de **uso común**, como veremos en el Capítulo 3. Sin embargo, antes de esto, se requiere más descripción de los elementos básicos de un sistema de procesamiento.

1.5.3 Memory

- Como vimos en los apartados anteriores, un procesador necesita tener **memorias** tanto para el **código** como para los **datos**.
- Otra área específica se utiliza para una pila (**stack**), una forma especial de memoria de datos en la que el procesador puede hacer **PUSH** datos para almacenarlos y **POP** datos para recuperarlos.
  - Consideraremos esto con más detalle más adelante, pero primero necesitamos tener un modelo de memoria, comenzando con un modelo de alto nivel antes de pasar al nivel del transistor.

1.5.3 Memory

- Una **memoria** puede verse como una gran cantidad de **cajones** separados, cada uno con su propio número de **identificación**, comenzando desde **0** y **aumentando** a partir de entonces.
  - Así, una posición en la memoria puede verse como un cajón específico y dentro de este un conjunto de cuadros representan los bits de información, un cuadro vacío representa un 0 binario y un cuadro lleno un 1 binario.
  - Si todos los cajones están alojados en un solo **gabinete**, esto representa el **chip** de memoria.
  - Se selecciona mediante una señal, el chip select (**CS**), que puede verse como la **llave** para bloquear o desbloquear la **puerta** del gabinete.

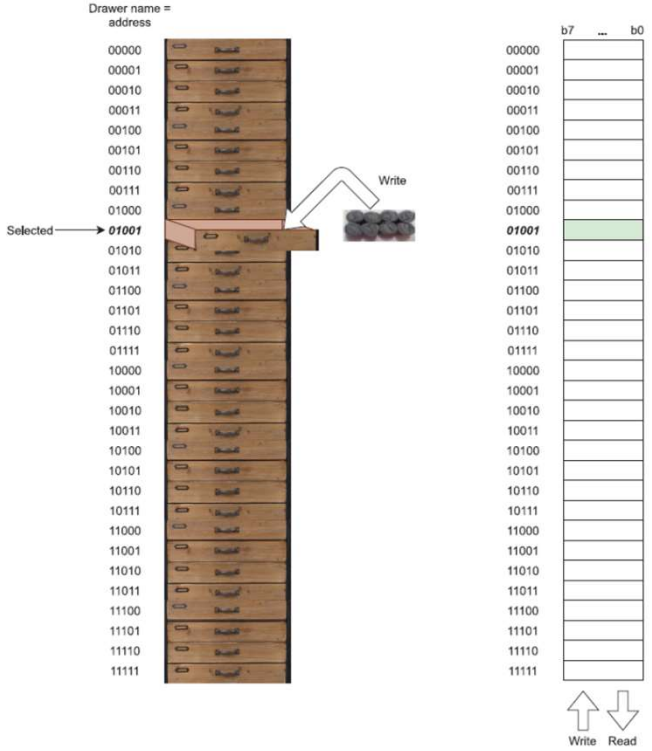


Figure 1.12 Memory model with drawers in a cabinet

1.5.3 Memory

- En la Figura 1.12 se presenta un modelo simple de un gabinete con cajones.
  - Por convención, un acceso de **escritura** lo realiza una unidad **solicitante**, que va del **procesador** a la **memoria**, y un acceso de **lectura** lee el **contenido** de una posición en la **memoria**.
  - Para la mayor parte de la **memoria**, la **lectura no** es **destructiva**, lo que significa que los **datos** se **retienen** en esa posición de la memoria después de haber sido leídos.
  - En la ilustración, se accede al cajón “01001” (en binario) y se coloca algo en su interior.
- El siguiente paso es crear lo que se llama un **mapa** de **memoria**, un **modelo** de las **memorias disponibles** en un sistema.
  - Independientemente de si la **memoria** es **interna** o **externa** al microcontrolador, el **modelo** de mapa de memoria será el **mismo**.

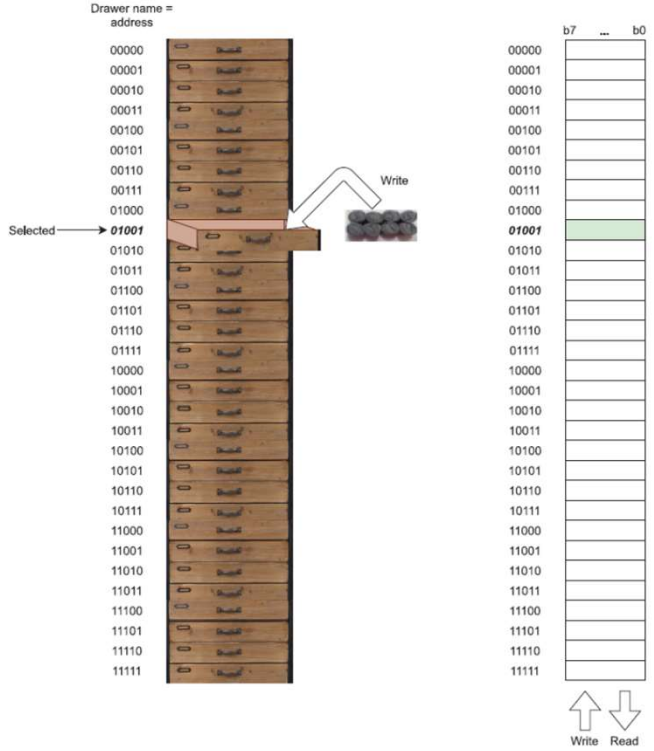


Figure 1.12 Memory model with drawers in a cabinet

1.5.3 Memory

- Para diseñar un SoC o entender cómo está diseñado, debemos determinar qué información se requiere en términos de acceso a la memoria disponible:
  - Primero, ¿cuál es el espacio de direcciones disponible (el rango de direcciones de memoria a las que se puede acceder)?
    - Esto dependerá de la cantidad de líneas de dirección proporcionadas por el procesador. Por tanto, 24 líneas de dirección proporcionan un máximo de 2^{24} posiciones de memoria, o 16 mebibytes (MiB) de espacio de memoria; para un espacio de direcciones de 32 bits, hay 2^{32} posiciones de memoria, 4 gibibytes (GiB) de espacio de memoria, según la noción de que una dirección representa un byte en la memoria.
  - ¿Cuál es el ancho del bus de datos?
    - Los tamaños comunes son 8, 16 o 32 (o incluso más) bits. Al día de hoy, la mayoría de procesadores modernos, incluido el Cortex-M, utilizan 16 o 32 bits.
  - ¿Qué tipo de memoria se debe utilizar? ¿Volátil o no volátil? ¿Estático o dinámico? ¿Asíncrono o sincrónico?

1.5.3 Memory

- ¿Qué **señales** de **control** se necesitan?
- ¿Se requerirá que el procesador comience a **ejecutar código** después de un **Reset**? Si es así, se necesitará memoria no volátil.

- Una vez respondidas estas preguntas, podemos iniciar el **mapeo** de la **memoria**, de la siguiente manera:
  - Cuando el procesador comienza a funcionar al encenderse o después de un **Reset**, necesita **ejecutar código** para iniciar el sistema. Este código debe estar disponible de inmediato y se denomina **boot** (**strap**) **code**.
  - Dependiendo del procesador utilizado, el **vector** de **Reset**, la dirección en la que iniciar un programa al **encender** (**Power Up**) o **reiniciar** (**Reset**) y la tabla de **vectores** de **excepción** (para manejar cuestiones de **inicio**) comenzará en la dirección **0x0000 0000**; por tanto, se necesita memoria no volátil.

1.5.3 Memory

- Para algunos otros procesadores, esta dirección estará en una ubicación diferente. Tomando la Figura 1.13 como ejemplo, seleccionemos una memoria no volátil (flash) de 4 MiB en un bus de datos de 16 bits.
- A continuación, el sistema de este ejemplo debe tener unos pocos megabytes de memoria estática (SRAM) que se puedan leer y escribir. Seleccionemos 8 MiB de SRAM, lo que da un espacio de memoria de 0x80 0000 bytes.
- Supondremos que este microcontrolador hipotético tiene un bus de memoria de 16 bits de ancho, con acceso separado al byte inferior (LB; d7..d0) y al byte superior (UB; d15..d8).
  - Así, una memoria no volátil de 4 MiB (suponiendo un byte de 8 bits) está representada por dos memorias de 2 MiB cada una organizadas en un bus de datos de 8 bits, o un único dispositivo de 2 mebi-palabras de 16 bits de ancho.

1.5.3 Memory

- Otra cosa que necesitamos saber, y que depende del **fabricante** del **microcontrolador**, es dónde se asignarán las **interfaces** programables: ¿en el espacio de **memoria** o en un espacio de entrada/salida (**I/O**) específico? ¿El procesador tiene líneas de hardware específicas e instrucciones específicas para su uso?
  - En el caso del **Arm Cortex-M**, este mapeo está en el **mismo espacio** de memoria y, generalmente, en el rango de **direcciones superior**.
- También necesitamos saber si la arquitectura externa se ajusta a **von Neumann** o **Harvard**.
- Para el acceso a la memoria externa, normalmente es **von Neumann**, incluso si utiliza una **arquitectura Harvard** internamente.
- Para que la **decodificación** sea más sencilla, la dirección inicial de una memoria es un múltiplo de su tamaño.
- Una **dirección** se trata como una **dirección** de **byte** en casi todos los sistemas. Si el ancho del bus es de 16 bits, la dirección **bit0** es igual a **0** para una dirección de memoria **alineada**.

1.5.3 Memory

- Si tomamos el ejemplo que se muestra en la Figura 1.13, podemos ver que la **memoria flash** (no volátil) comienza en la **dirección 0** (0x00 0000); la dirección más baja está en la **parte superior** y la dirección **aumenta** hacia **abajo**.
  - El bus de datos tiene 16 bits de ancho, el incremento de dirección es de dos bytes a la vez de una fila de memoria a la siguiente y, en esta organización, está en formato **little-endian**: la **dirección** más **baja** está en la parte **inferior** del **bus**.
  - Así, en la Figura 1.13, en la misma fila la dirección par está en el lado inferior derecho del bus (bits d7..d0), mientras que el desplazamiento (+1) está en el lado izquierdo (bits d15. .d8).

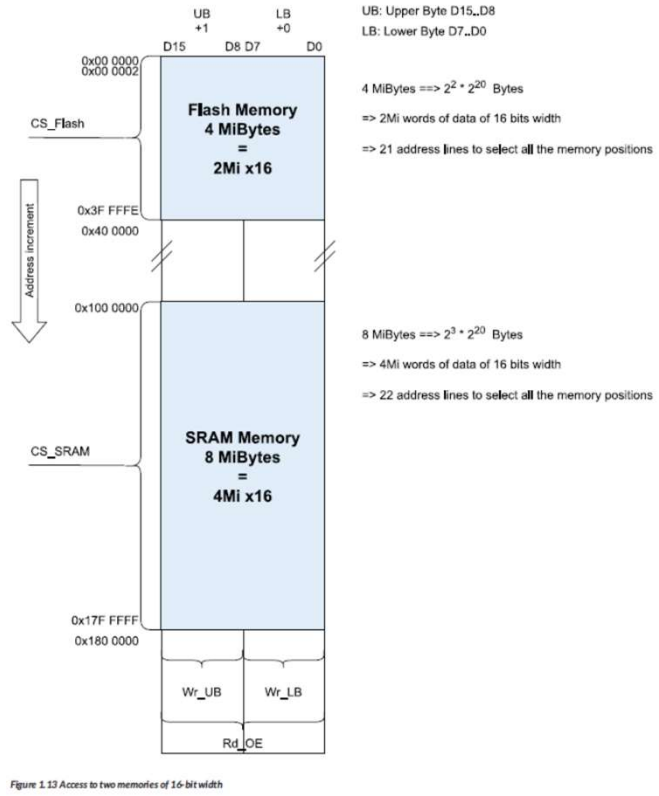


Figure 1.13 Access to two memories of 16-bit width

1.5.3 Memory

- Las señales de chip select (CS) dependen de la parte más alta del bus de direcciones.
  - Para muchos microcontroladores, esto es simplemente una interfaz programable para inicializar la dirección base correcta para todas las memorias disponibles.
- Para el acceso de lectura, incluso si la memoria necesita leerse un byte a la vez, el acceso sólo está disponible para todo el ancho del bus de datos: el procesador simplemente debe ignorar cualquier dato innecesario proporcionado.
- Sin embargo, para el acceso de escritura es obligatorio seleccionar solo la parte específica de la memoria (y del bus de datos) en la que se escribirá, es decir, el byte superior o el byte inferior.
  - De lo contrario, es probable que se escriba información incorrecta en una parte no deseada de la memoria.

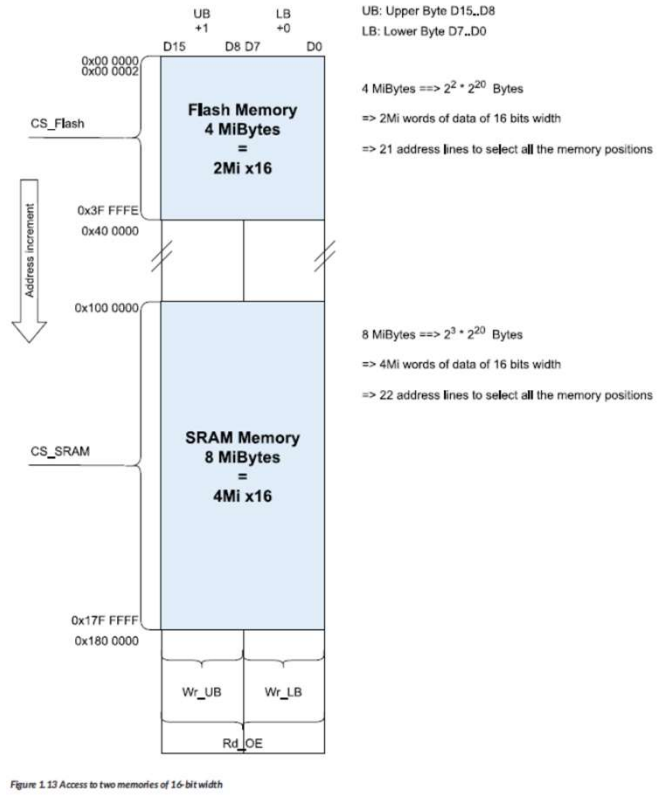


Figure 1.13 Access to two memories of 16-bit width

1.5.3 Memory

- El procesador proporciona señales especiales (habilitación de bytes) para controlar esto.

- Para un bus de datos de **32 bits**, se necesitan **cuatro** señales de habilitación de **bytes**: BE3 para b31–b24; BE2 para b23-b16; BE1 para b15-b8; BE0 para b7–b0, suponiendo una convención **little-endian**.
- Para un procesador **big-endian**, el orden se invertirá: BE0 para b31–b24; BE1 para b23-b16; BE2 para b15-b8; BE3 para b7–b0.
- En el Capítulo 8 se exploran más características de diseño en relación con la memoria.

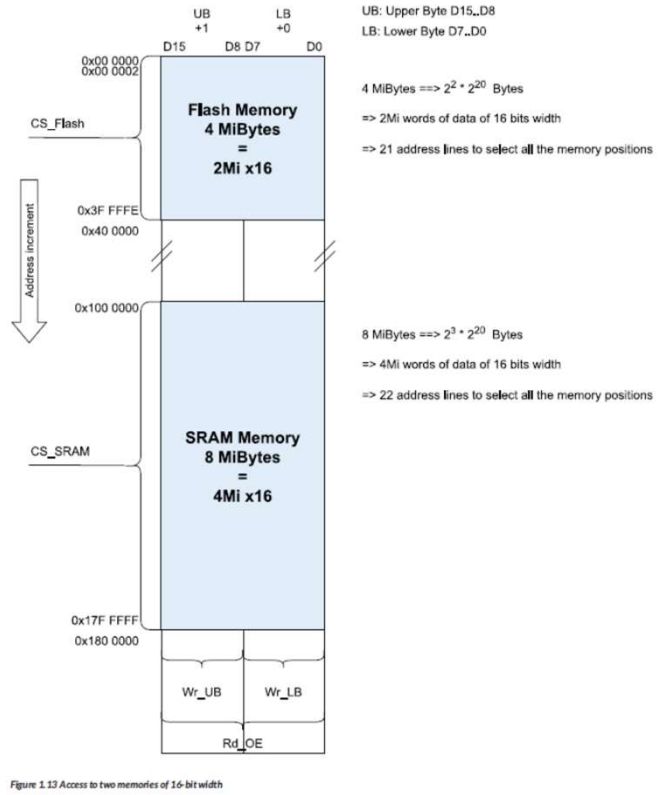


Figure 1.13 Access to two memories of 16-bit width

1.5.4 Memory Classes

- Cuando se necesita memoria, existen muchos modelos diferentes para elegir. Se necesita una clasificación inicial de alto nivel, comenzando por si la **memoria** es **volátil** o **no volátil** (ver Figura 1.14):
- Memoria **no volátil**. Cuando se **corta** la **energía**, el **contenido** de la memoria **no** se **pierde** y está disponible nuevamente al encenderse. Esto es obligatorio para el código de inicio utilizado cuando se inicia o reinicia un procesador.
  - **ROM**: Memoria de sólo lectura; el contenido de esta memoria se define durante la fabricación del dispositivo. En caso de error, el chip queda efectivamente inutilizable.
  - **PROM**: ROM programable; el contenido se puede programar con una pieza de hardware específica llamada programador de memoria. El contenido sólo se puede programar una vez.
  - **EPROM**: PROM borrable; al igual que una PROM, se podía programar a través de un hardware programador específico, pero se instaló una ventana de cuarzo en la parte superior del dispositivo y también era posible borrar la memoria mediante luz ultravioleta, aunque era necesario retirar el

1.5.4 Memory Classes

- dispositivo de su placa para borrarlo y reprogramación (que necesitaba 12,5, 21 o 25 V según el dispositivo específico).Una variación específica fue una carcasa de plástico sin capacidad de borrar la memoria, llamada OTP (One-Time Programmable).
- EEPROM (o E2PROM): PROM borrable eléctricamente; como avance en EPROM, ahora se podría programar la memoria directamente en la placa y reescribir su contenido mediante borrado eléctrico byte a byte. Baja densidad y costoso, pero todavía se usa hoy en día dentro de algunos microcontroladores para memoria de tamaño limitado.
- Memoria Flash: Basada en EEPROM y que representa un gran paso adelante en el mundo de los sistemas embebidos, la memoria flash de tipo NAND puede borrarse en bloques (la de tipo NOR funciona a nivel de palabra) y reescribirse in situ utilizando un protocolo y un estándar adecuados. Voltaje. Comúnmente disponible en tamaños de muchos gigabytes, se usa habitualmente en memorias USB y tarjetas SD para cámaras, y puede usarse en casi todos los microcontroladores.
- FRAM: Memoria de acceso aleatorio ferroeléctrica; una tecnología prometedora anunciada hace varios años, que se esperaba que reemplazara a la SRAM y la memoria flash, pero las presiones

1.5.4 Memory Classes

- competitivas de esta última han obstaculizado el desarrollo de FRAM y aún no está disponible a bajo costo y alta densidad, aunque está disponible en algunos microcontroladores para modificaciones fáciles. memoria no volátil.
- MRAM: RAM magnetorresistiva; como ocurre con FRAM, una tecnología prometedora que, hasta la fecha, ha perdido terreno frente al floreciente mercado de la memoria flash y no se utiliza ni está disponible ampliamente.
- PRAM: RAM de cambio de fase; Utilizando cambios en la resistencia eléctrica en medios tipo CD/DVD, PRAM promete mayores capacidades que FRAM o MRAM, y ha estado disponible para algunos microcontroladores.
- ReRAM: RAM resistiva; similar a PRAM en el uso de cambios de resistencia eléctrica, aquí en un dieléctrico.
- FeFET: memoria de transistor de efecto de campo ferroeléctrico; una nueva esperanza para una memoria no volátil rápida y de alta densidad basada en la polarización permanente del campo eléctrico. Vale la pena seguir su evolución.

1.5.4 Memory Classes

- Memoria **volátil**. Representada por la memoria de acceso aleatorio (**RAM**) tradicional (pero tenga en cuenta que las formas de memoria no volátil ahora también ofrecen acceso aleatorio [**NVRAM**], como se describió anteriormente). Cuando se **corta** la **energía**, el **contenido** de la memoria volátil se **pierde**. Puede ser **estático** o **dinámico**. En este último caso, es necesario **actualizar periódicamente** el **contenido** de la memoria.
  - **SRAM**: RAM estática; esta es la forma básica de RAM volátil, disponible hasta unos pocos megabytes. Se necesitan cuatro transistores para memorizar un poco de información, más dos más para acceder al contenido durante el acceso de lectura o escritura.
  - **SSRAM**: SRAM síncrona, en la que el acceso se sincroniza mediante un reloj.
  - **DRAM**: RAM dinámica; La idea principal detrás de la DRAM es hacer uso de un elemento de memoria basado en un solo transistor para memorizar y acceder a un poco de información. La ganancia de rendimiento es enorme (aproximadamente cuatro veces en comparación con SRAM), pero es necesario actualizar la memoria completa cada pocos milisegundos. El acceso se realiza en dos pasos: selección de una fila de bits y luego de la columna de la fila seleccionada. Los datos están organizados en un formato similar a una matriz.

1.5.4 Memory Classes

- SDRAM: DRAM síncrona; Utiliza el mismo principio que DRAM pero con acceso síncrono controlado por reloj.
- SDRAM DDR: SDRAM de velocidad de datos dual (DDR[1], DDR2, DDR3, DDR4, DDR5); mejora el rendimiento de la SDRAM al acceder a la memoria en ambos flancos (ascendente y descendente) de la señal del reloj. Para la misma frecuencia de reloj, la velocidad de transferencia de datos se duplica.
- GDDR (SDRAM): DDR gráfico; muy similar a DDR, pero con funciones especiales para mejorar el acceso local y el rendimiento de las GPU.
- LPDDR: DDR de bajo consumo; una versión de DDR (SDRAM) con un consumo de energía significativamente menor, dirigida a computadoras móviles (tablets y smartphones).
- QDR (SDRAM): velocidad de datos cuádruple; utiliza dos relojes y buses de lectura y escritura separados para lograr velocidades de transferencia más rápidas que DDR, y está dirigido a comunicaciones y redes de alta velocidad.

1.5.4 Memory Classes

- La memoria **volátil** se puede clasificar además según si el acceso a los datos es **síncrono** o **asíncrono**:
  - **Asíncrono**: no se utiliza ningún **reloj**. La familia RAM tiene algunos dispositivos sin reloj; Realizan acceso asincrónico.
  - **Síncrono**: el **acceso** a la memoria se **basa** en el **reloj**; Las soluciones RAM más recientes utilizan un reloj para sincronizar las transferencias de datos.
- El **acceso** a los datos para transferir direcciones y datos se puede realizar en **paralelo** o en **serie**:
  - Acceso a datos **paralelos**: en un bus paralelo, todas las líneas de direcciones y datos están disponibles, lo que permite al procesador ejecutar código y acceder a datos directamente con una velocidad de transferencia muy rápida entre la memoria y el procesador; menos de diez nanosegundos para la transferencia de una palabra completa (de varios bits).
    Tenga en cuenta que algunos dispositivos multiplexan estas líneas.

1.5.4 Memory Classes

- Acceso a datos en **serie**: cuando el **número** de **líneas** utilizadas para transferir direcciones y datos debe ser **limitado**, el acceso en serie a la memoria (un bit a la vez) es una opción sensata. Las memorias utilizadas en tarjetas **SD** y memorias **USB** utilizan este enfoque. La transferencia de información se realiza con un bus serie en forma de controlador **SPI** (Interfaz periférica serie) o **QSPI** (SPI en cola). Los protocolos alternativos incluyen **USB** (rápido) e **I2C** (protocolo de baja velocidad para múltiples periféricos).

El direccionamiento también se puede abordar de dos maneras diferentes:

- Direccionamiento **paralelo** para **acceso directo**: Para acceder a una posición de memoria, la **dirección completa** se transfiere en paralelo (para cubrir muchos bits de la dirección). A veces, la dirección se proporciona en dos pasos como para la DRAM (ver arriba): primero para especificar la dirección de la fila, luego la dirección de la columna.
- Acceso de **transmisión**: se transfiere la **primera dirección** a la que se accede en la memoria, después de lo cual la dirección se incrementa automáticamente y los datos se transfieren hacia/desde la siguiente dirección contigua, y así sucesivamente.

1.5.4 Memory Classes

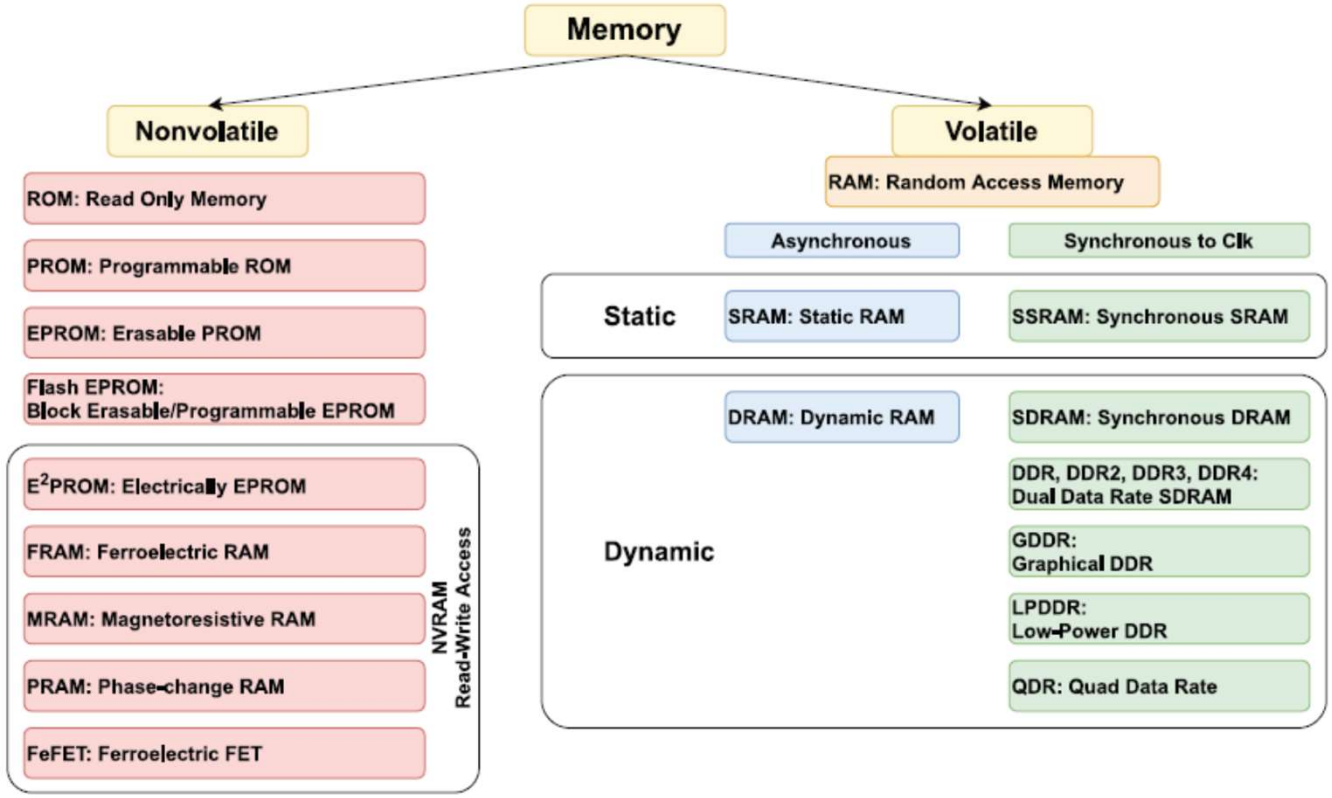


Figure 1.14 Classification of memory as volatile/nonvolatile

1.5.5 Internal Memory Organization

- Para comprender mejor la función interna de la memoria, consideraremos un chip SRAM, interno a un microprocesador, como se ilustra en la Figura 1.15.
- La **celda** de memoria está representada por dos **inversores** (compuestos por dos transistores cada uno) en **loop**. Se utilizan dos **transistores** como **gate** para transferir el **bit** de datos de dos líneas verticales, **D** y **/D**. Durante la **lectura**, estos dos **transistores** de paso son **seleccionados** en una **fila** por un **decodificador**, controlado por la **dirección** de la posición de memoria a la que **acceder**. Para un acceso de **lectura**, los valores directo y complementario se conectan a un amplificador diferencial para proporcionar la señal **DataOut**. La **salida** está **disponible** en el caso de **un ciclo** de **lectura**.

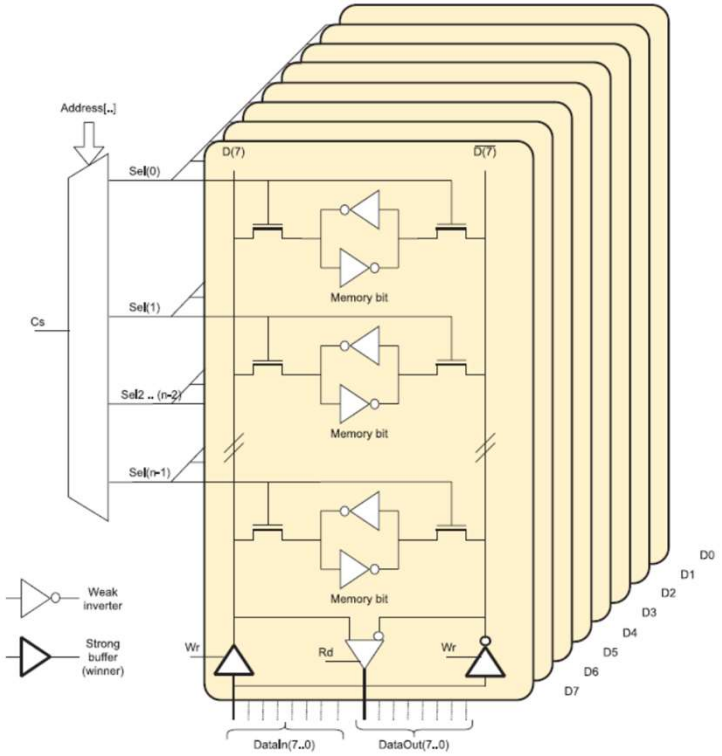


Figure 1.15 Organization of an internal SRAM

1.5.5 Internal Memory Organization

- Para un ciclo de escritura, ocurre el mismo proceso pero a la inversa. La señal DataIn se activa dentro de las líneas verticales mediante fuertes buffers. El par inversor recibe el nuevo valor D y /D para escribir. Habrá un cortocircuito si el valor a escribir y el que ya está en la celda tienen niveles opuestos.
- Sin embargo, los transistores están diseñados para soportar esto. El valor a escribir siempre gana gracias a los transistores más potentes. Cuando ya no se selecciona el chip, los transistores que pasan se cierran y el par inversor mantiene el valor memorizado.

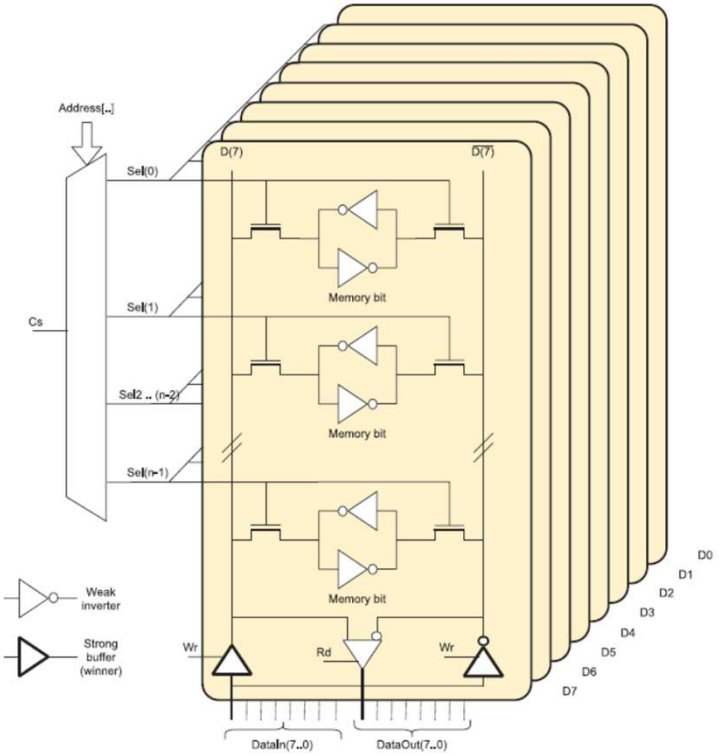


Figure 1.15 Organization of an internal SRAM

1.5.5 Internal Memory Organization

- La misma **arquitectura** está disponible para cada bit de datos de la memoria, seleccionado por una dirección única. Si el bus de datos externo no tiene líneas físicas **DataIn** y **DataOut** separadas, los bits se conectan en un bus de datos bidireccional.
- Para obtener un bus de datos con un ancho mayor que 1 bit, se agregan múltiples áreas idénticas con la misma línea de selección; en la Figura 1.15, el bus de datos tiene un ancho de 8 bits.

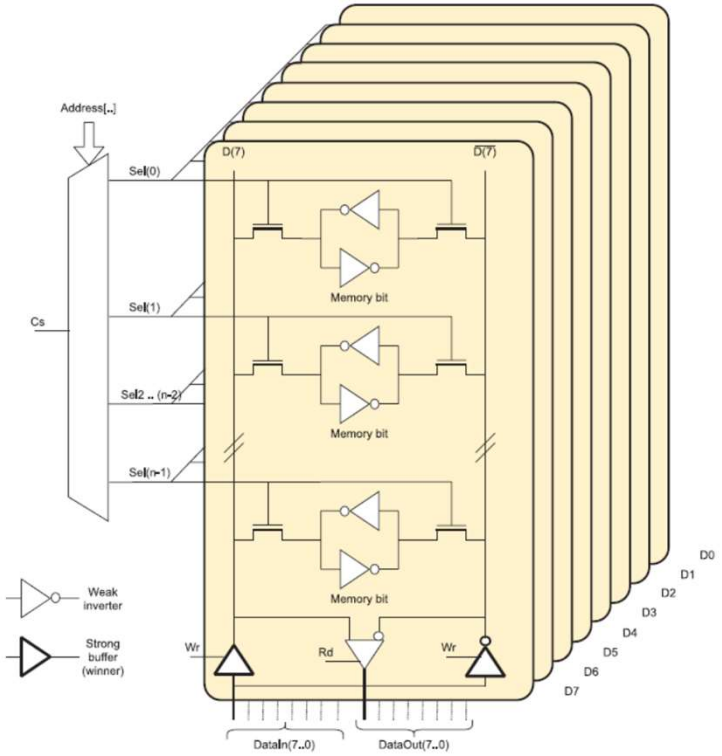


Figure 1.15 Organization of an internal SRAM

1.5.6 Interrupt Controllers

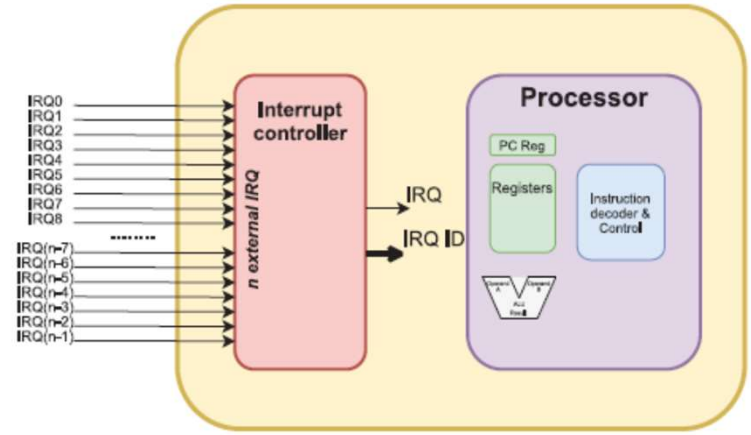
- Volvemos ahora al **procesador**. **Ejecuta** las **instrucciones** que se proporcionan desde la memoria y lee **secuencialmente** desde la dirección proporcionada por el **registro PC**.
- La dirección de instrucción proporcionada por el **registro PC** se **incrementa automáticamente** según el tamaño del código de instrucción (comúnmente se usan 16 bits y 32 bits, pero esto puede variar y dependerá del procesador específico).
  - Algunas instrucciones proporcionan **bifurcaciones** o **direccionamientos indirectos** en la ejecución del código.
- Sin embargo, ¿qué sucede si tenemos instrucciones específicas que deben ejecutarse como resultado de una **solicitud externa** o un **error** en la **ejecución** del **código**, o si necesitamos **responder** a un **evento** externo?
- Estas solicitudes se denominan **excepciones** o **interrupciones**.
  - El término **excepción** se utiliza generalmente para cualquier tipo de solicitud especial (de **hardware** o **software**); el término **interrupción** generalmente se refiere una solicitud que se origina en el **hardware**.

1.5.6 Interrupt Controllers

- Las fuentes de posibles **interrupciones** de hardware son todas las **interfaces** programables a las que sirve el procesador; por ejemplo, una **interfaz serie** que está lista para enviar o recibir un dato, un cambio a un nivel de pin de entrada/salida de propósito general paralelo (**GPIO**), un paquete de datos que se recibe en una red, un temporizador que detecta un valor de tiempo específico, etc.
  - Un **error** detectado en una **interfaz** programable también puede generar una **interrupción**.
- En general, el **procesador** necesita **detectar** e **identificar** la **fuente** de la **interrupción** para ir a una pieza de **software** adecuada para **manejar** la **solicitud** y **hacer algo** con los datos asociados.

1.5.6 Interrupt Controllers

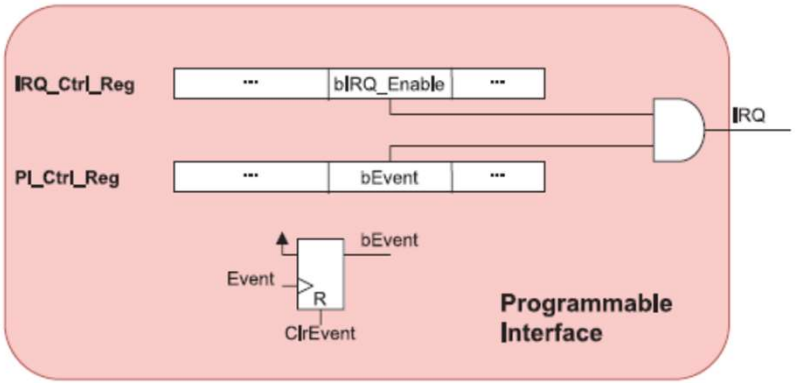
- Un **controlador** de **interrupciones** es una unidad que recibe **interrupciones** externas y genera una solicitud de interrupción (**IRQ**) al **procesador**. La Figura 1.16 ilustra una conexión simplificada entre un controlador de interrupciones y el procesador. En el caso de interrupciones de hardware, se pueden generar **n solicitudes** de **interrupción** externas.
- El **controlador** de **interrupciones** está asociado con un mecanismo que permite descubrir la solicitud de **mayor prioridad**, en forma de **ID** o identificador de solicitud de **interrupción**. Esta información a menudo se denomina **vector**. A veces, el **controlador** de **interrupciones** se trata como un elemento **interno** del propio **procesador**; en otras arquitecturas puede ser una **interfaz** programable **bajo** el **control** del **procesador**.



pt controller

1.5.6 Interrupt Controllers

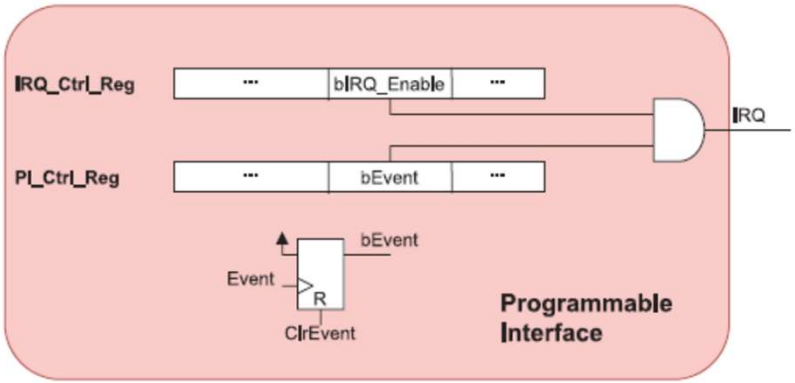
- La Figura 1.17 muestra la configuración mínima requerida para administrar una IRQ dentro de una interfaz programable.
  - Esta configuración se utiliza en la mayoría de las interfaces programables para generar una solicitud de interrupción física a un procesador.
  - Los nombres pueden cambiar, pero la funcionalidad sigue siendo la misma.



from a programmable interface

1.5.6 Interrupt Controllers

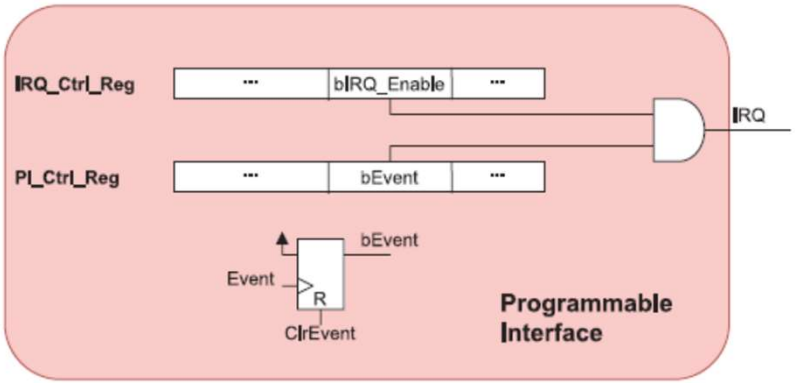
- Así, tenemos **dos registros** internos con **un bit** en cada uno:
  - Un **evento** en un registro de control, aquí llamado **PI_Ctrl_Reg**, de la interfaz programable se **activa** cuando ocurre un **evento** de **hardware**. El procesador puede leer este bit y, mediante **polling**, puede ver si se ha producido el evento asociado y cuándo.
  - Un segundo registro, aquí llamado **IRQ_Ctrl_Reg**, tiene un bit (**bIRQ_Enable**) que se usa como **máscara** para permitir la activación física de una **IRQ** al procesador o a un controlador de interrupción intermedio si **bEvent** está activado.



from a programmable interface

1.5.6 Interrupt Controllers

- Para reconocer la **interrupción**, el **software** debe realizar una **acción**, generalmente en la **función** de **manejo** de interrupciones del programa (firmware/controlador). Esto también dependerá de la interfaz programable para activar la señal **ClrEvent**, como se muestra en la Figura 1.17. Se utilizan dos enfoques principales:
  - **bEvent** se borra mediante una escritura directa en **PI_Ctrl_Reg**. Lea atentamente la documentación específica porque para algunas interfaces programables el bit se borra escribiendo un '0' y en otras se debe escribir un '1'.
  - Escribiendo directamente al registro (**R**) que generó el evento. Nuevamente se debe consultar la documentación específica.



from a programmable interface

1.5.6 Interrupt Controllers

- La salida **IRQ** de la interfaz programable está **conectada** al **procesador** (dentro del **microcontrolador**) a través de una unidad específica: el **controlador** de **interrupciones**, como se muestra en las Figuras 1.16 y 1.18.
- El **controlador** de **interrupciones** puede tener **diferentes capacidades** según la familia de procesadores. El Capítulo 3 entrará en más detalles sobre la familia **Cortex-M**.

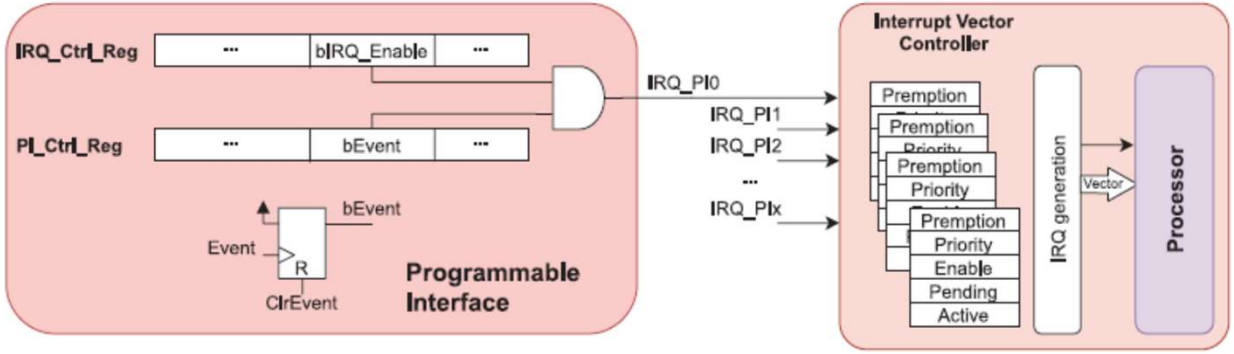


Figure 1.18 *Interruption from programmable interface to processor via interrupt vector controller*

1.5.6 Interrupt Controllers

- La idea detrás de esta unidad es poder recibir **muchas** fuentes de **excepciones** generadas por **hardware** o **software**.
  - Si se proporcionan **varios al mismo tiempo**, es necesario el **arbitraje**. Esto lo podría hacer el **procesador**, pero es más eficiente hacerlo a través del **hardware**.
  - Dependerá de una serie de **parámetros**, como si ya se está ejecutando una interrupción de mayor **prioridad**, la selección de la solicitud de mayor **prioridad** en espera, etc.

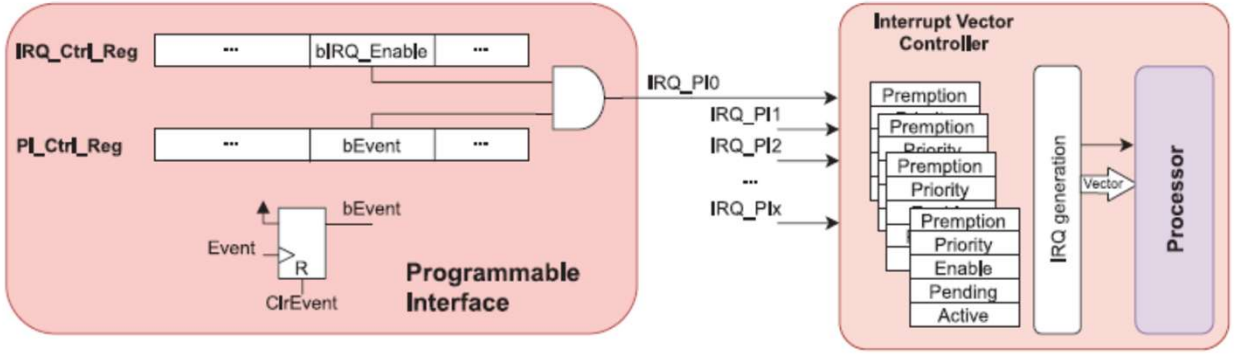


Figure 1.18 *Interruption from programmable interface to processor via interrupt vector controller*

1.5.6 Interrupt Controllers

- Si una nueva **interrupción** es **válida**, se solicita al núcleo del **procesador** que la atienda y recibe un **vector asociado** con la solicitud. Este vector **indica dónde** debe **ir** el **procesador** en el programa: la dirección del controlador de **IRQ**.

- En todos los **procesadores** existe una manera de **enmascarar todas** las **interrupciones** (o casi todas) con una única máscara global. Este es **PRIMASK**, **bit0** en el procesador **Cortex-M**.

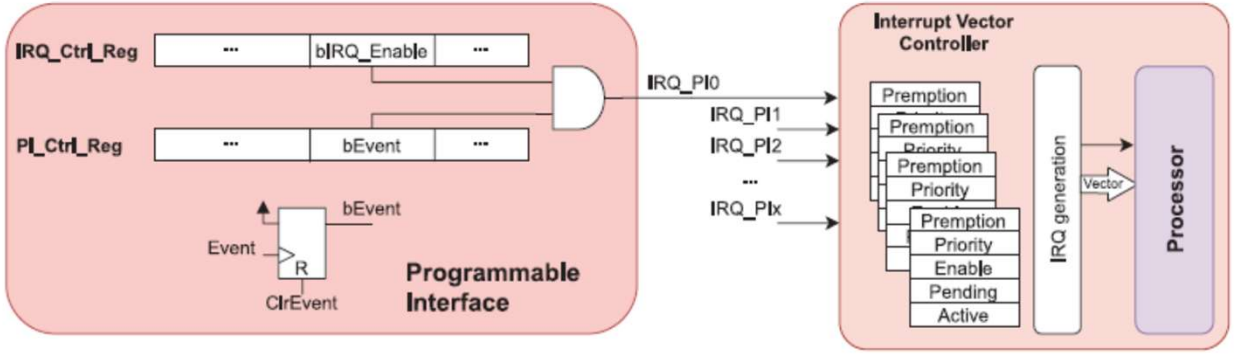
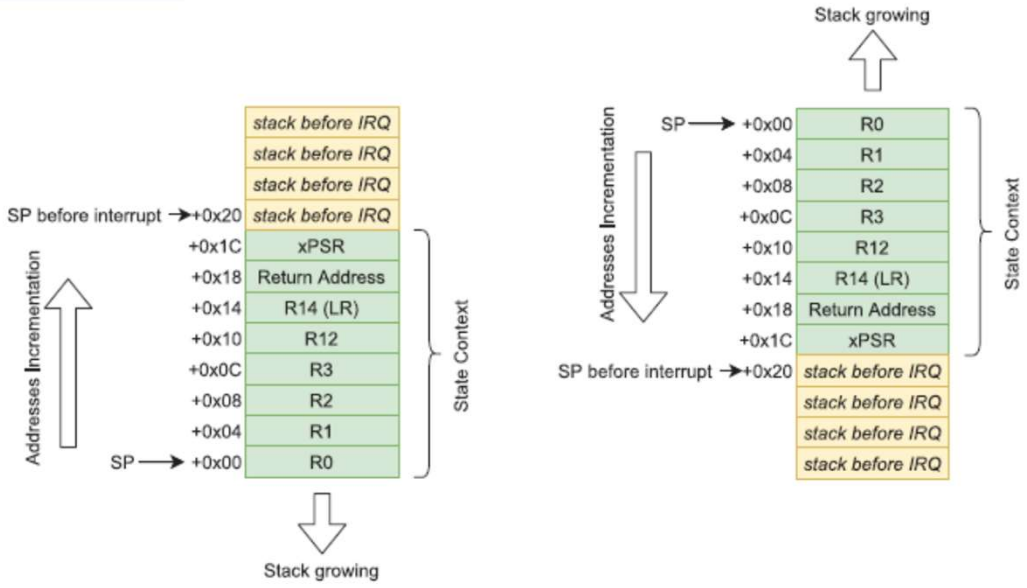


Figure 1.18 *Interruption from programmable interface to processor via interrupt vector controller*

1.5.6 Interrupt Controllers

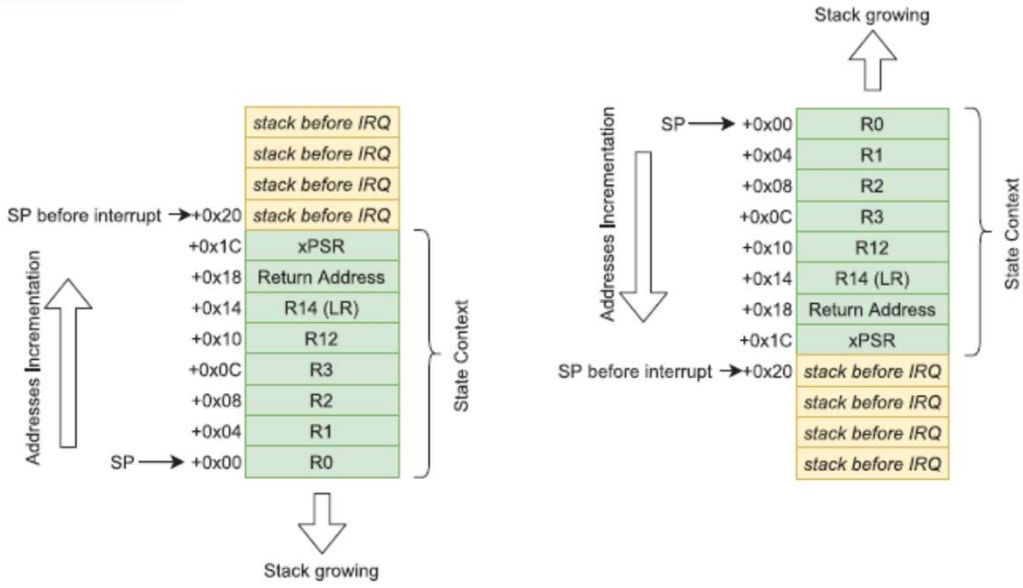
- En el procesador **Cortex-M**, se **guarda automáticamente** algunos **registros** cuando se **acepta** una solicitud de **interrupción**.
  - La Figura 1.19 muestra los registros guardados en el **stack**.
  - Necesitamos revisar cuidadosamente la documentación de un procesador para ver cómo se llena su **stack**.
  - El lado izquierdo de la figura muestra un **stack** con direcciones que se incrementan desde abajo hacia arriba.



ıving automatically on the stack after an IRQ on a Cortex-M processor

1.5.6 Interrupt Controllers

- El **stack** en sí crece en la dirección opuesta, desde la dirección más alta a la más baja, una convención utilizada en la mayoría de los procesadores.
- La vista de la derecha representa exactamente el mismo proceso, pero las direcciones se incrementan de arriba a abajo.
- Tenga en cuenta que esto es solo una representación visual de la dirección del incremento de la dirección de la memoria y no refleja los datos que llenan la memoria en un orden diferente.



ıving automatically on the stack after an IRQ on a Cortex-M processor

1.5.7 Memory Protection Unit (MPU)/Memory Management Unit (MMU)

- Cuando un procesador ejecuta un programa, puede hacerlo con o sin sistema operativo. Puede hacerlo como un programa único (secuencial) o mediante muchas tareas pseudo-paralelas, a menudo llamadas subprocesos. Esta es una forma de asignar una pequeña porción del tiempo del procesador a muchas piezas de código casi al mismo tiempo: a cada hilo se le da la impresión de ejecutarse de forma aislada.
- Cuando un sistema es simple, el enfoque básico de tener todo, como recursos, memorias, servicio de interrupción, etc., en un solo espacio es la forma más fácil de trabajar. A medida que un sistema se vuelve más complejo, con múltiples tareas y diferentes espacios de tareas que deben protegerse entre sí, se hace necesario un hardware específico.
- Un ejemplo sencillo es la unidad de protección de memoria (MPU), que de hecho protege tanto la memoria como las interfaces programables porque utilizan el mismo mecanismo de acceso. La MPU separa el mapa de direcciones de memoria en zonas y puede limitar el acceso a estas zonas en consecuencia. La MPU se explica con más detalle en el Capítulo 3.

1.5.7 Memory Protection Unit (MPU)/Memory Management Unit (MMU)

Una versión más sofisticada de la MPU es la unidad de gestión de memoria (MMU). Su función principal es recibir una dirección virtual del procesador y generar una dirección física. Utiliza tablas internas para convertir la dirección y proporcionar la real, que se pasa al decodificador, memorias e interfaces programables. La Figura 1.20 sitúa a la MMU en el contexto de un sistema global.

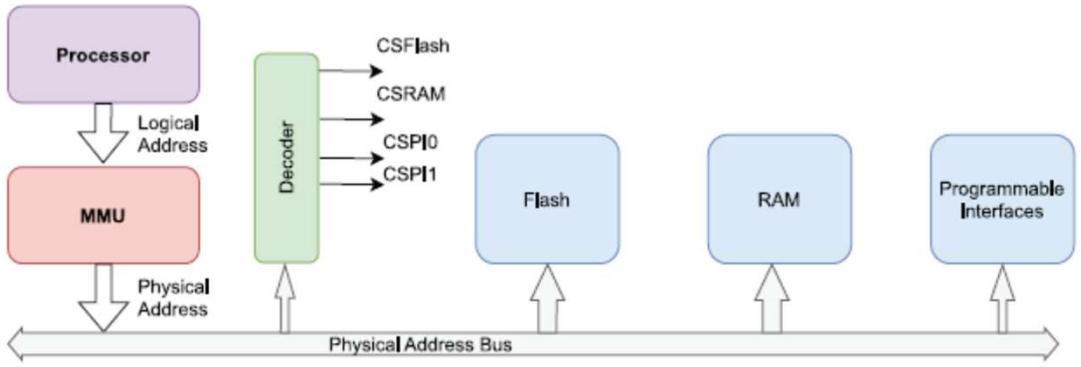


Figure 1.20 *Place of an MMU in a processor system*

1.5.7 Memory Protection Unit (MPU)/Memory Management Unit (MMU)

- En esta vista, el procesador genera y manipula una dirección lógica.
  - Esta dirección es la entrada a la **MMU**. A través de tablas asociadas a la gestión de tareas, la **MMU** genera la **dirección física** utilizada por el resto del circuito. Como ya se indicó, este enfoque se utiliza para **sistemas** más **complejos**, basados en **sistemas operativos** como **Linux**.
  - Las **MMU** suelen estar disponibles en computadoras más grandes y en la familia **Arm Cortex-A**, pero actualmente no se utilizan en la familia **Cortex-M**.

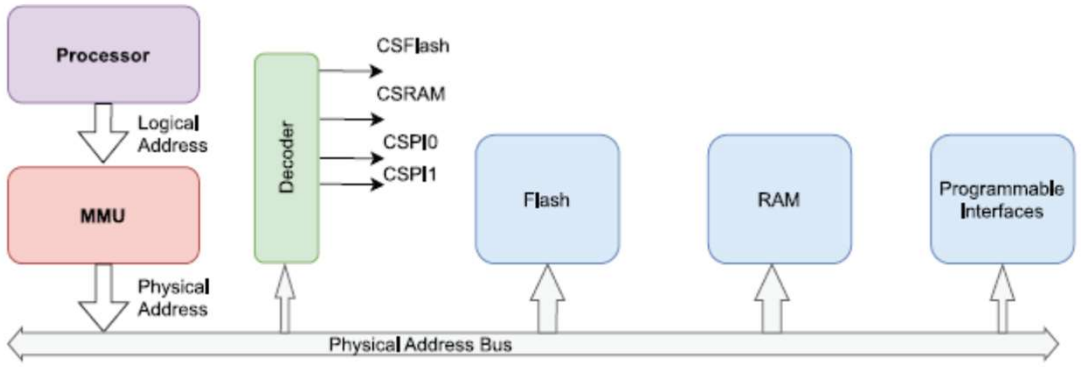


Figure 1.20 *Place of an MMU in a processor system*

1.5.7 Memory Protection Unit (MPU)/Memory Management Unit (MMU)

- La segunda tarea realizada por una MMU es la protección de diferentes áreas de memoria e interfaces programables dependiendo de la tarea en ejecución y los privilegios del procesador en ese momento. Algunas áreas de memoria pueden ser accedidas por algunas partes del código y otras no. Esta es la misma funcionalidad que está disponible en la MPU (como se describe arriba); este último es un elemento opcional en algunos procesadores Cortex-M y depende de la implementación según el fabricante. La tercera y última capacidad de una MMU es proporcionar memorias virtuales, lo cual es un medio para ofrecer más espacio de direcciones lógicas que el espacio de memoria físicamente disponible.

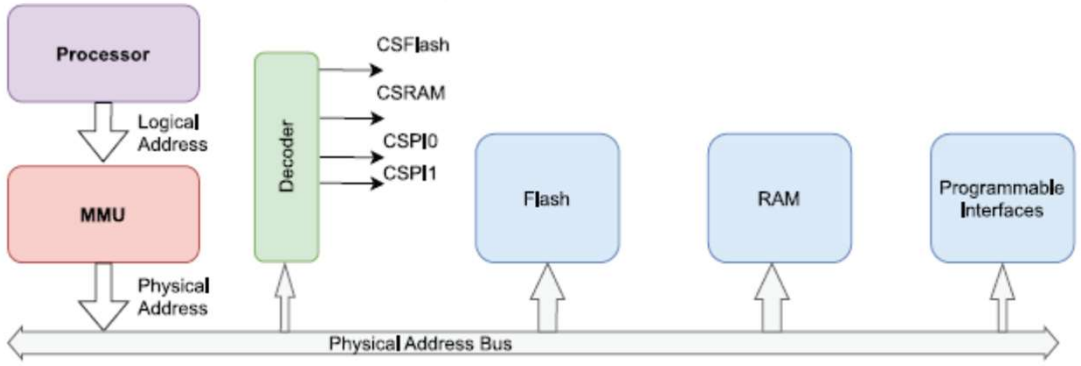


Figure 1.20 *Place of an MMU in a processor system*

1.5.7 Memory Protection Unit (MPU)/Memory Management Unit (MMU)

¿Cómo funciona esto? Consideremos la representación de MMU en la Figura 1.21, en la que tenemos un área de memoria utilizada para el acceso al programa a través del registro del Contador de Programa (PC).

- Podemos ver que tenemos un espacio de memoria lógica que es mayor que la memoria física disponible.
- Cuando el PC accede a una dirección en la región del espacio de memoria lógica marcada con (1), la MMU traduce esta dirección lógica (LA) a una dirección física (PA) en la memoria física. Si bien la PC solo accede a esta región, todo está bien.
- Sin embargo, tan pronto como la PC se aventura en la siguiente región del espacio de memoria lógica (2), no hay memoria física correspondiente disponible en el otro lado de la MMU.

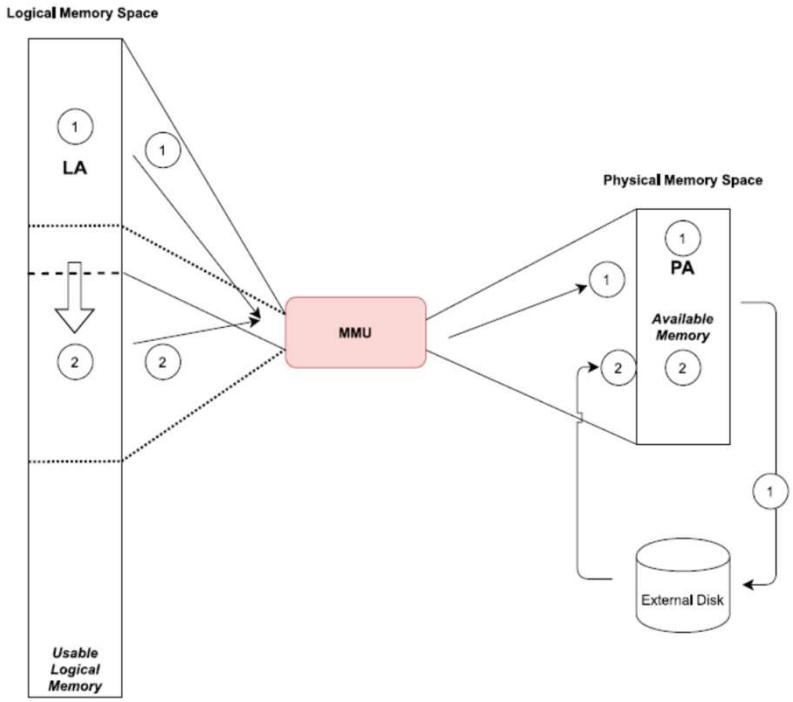


Figure 1.21 MMU with virtual memory

1.5.7 Memory Protection Unit (MPU)/Memory Management Unit (MMU)

- En esta situación, la MMU genera una excepción, la instrucción defectuosa se suspende y se ejecuta un software de bajo nivel para guardar el estado actual de la memoria física en un dispositivo externo (podría ser una unidad de disco, una memoria flash o algún otro almacenamiento de memoria de alta capacidad), y reemplácelo con una imagen de la memoria física correspondiente a (2) cargada desde el almacén de memoria externa.
- El mapa de memoria de la MMU se actualiza para reflejar el nuevo mapeo de LA -> PA y se debe volver a ejecutar la instrucción que provocó la excepción.

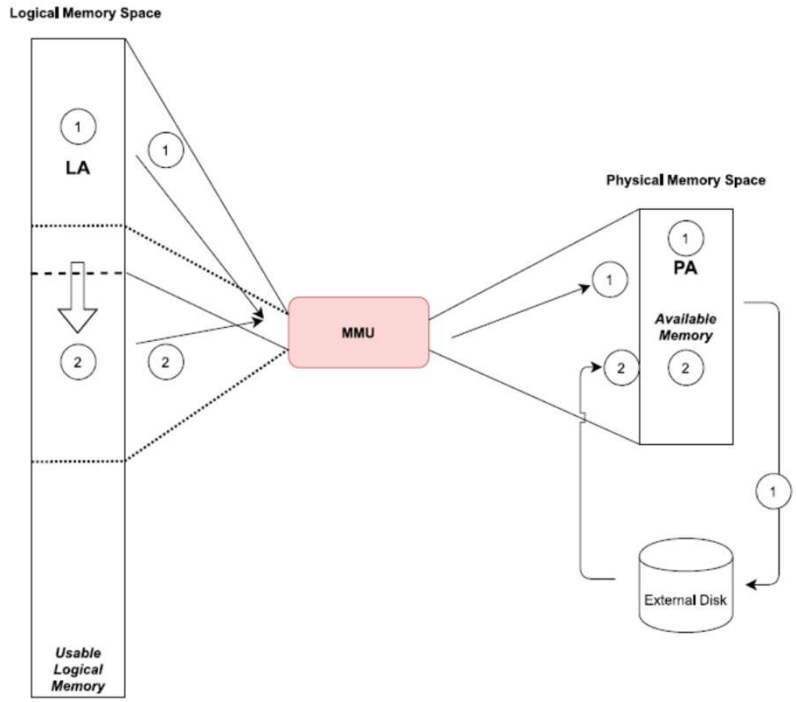


Figure 1.21 MMU with virtual memory

1.5.7 Memory Protection Unit (MPU)/Memory Management Unit (MMU)

- Estas operaciones de **intercambio** las realiza el procesador en una **memoria** que debe estar siempre **disponible**.
  - Es una característica interesante pero **tiene** un **costo**: el tiempo para intercambiar los datos entre la memoria física y la memoria externa.
  - En general, estos intercambios no se realizan en grandes **áreas** de **memoria** sino que utilizan páginas **pequeñas** de unos pocos kilobytes.
  - ¡**No** es adecuado para **sistemas** en **tiempo real**!

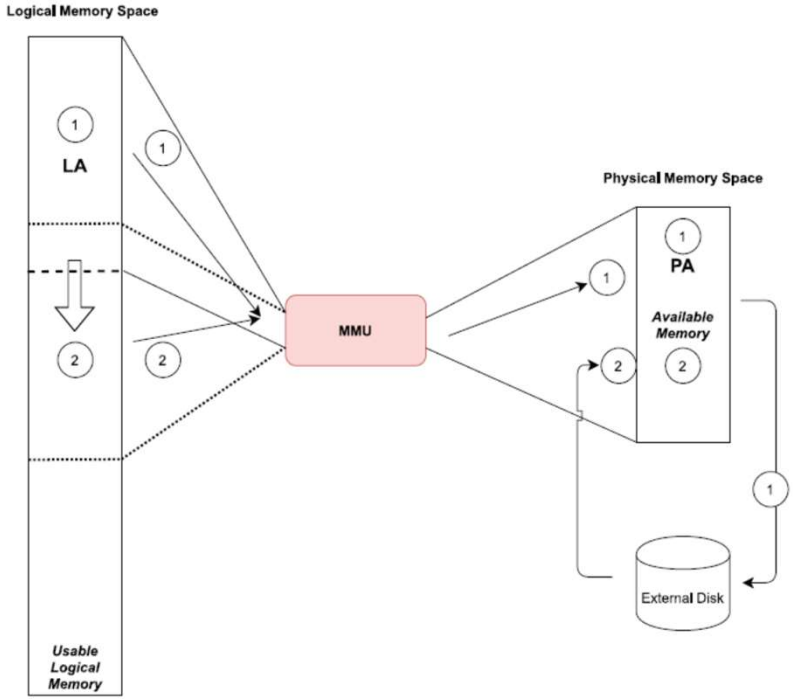


Figure 1.21 *MMU with virtual memory*

1.5.7 Memory Protection Unit (MPU)/Memory Management Unit (MMU)

- En resumen, una MMU tiene la capacidad de:
  - 1. traducir direcciones lógicas en direcciones físicas.
  - 2. habilitar la protección de áreas de memoria
  - 3. permitir una memoria virtual de mayor tamaño que la memoria física disponible.
- Debido a que la primera y la última de estas funciones no son útiles para los microcontroladores, no se implementan en este tipo de circuito integrado.
  - Sin embargo, los procesadores más potentes y que consumen mucha energía, que normalmente funcionan a una frecuencia más alta, sí implementan una MMU.
  - Sólo la segunda de estas funciones es útil debido a la complejidad de la clase de microcontrolador, y está disponible en la forma más limitada de una MPU.

1.5.8 Interconnects

- Dentro y fuera del **microcontrolador** se necesitan muchas **interconexiones** para transferir **direcciones** y **datos**, así como **señales** de **control** para sincronizar las transferencias.
  - El Capítulo 4 está dedicado a explicar los principios fundamentales de estas interconexiones, y el Capítulo 5 cubre los buses específicos (**AMBA AHB**, **APB** y **AXI**) utilizados en la familia **Cortex-M**.
- Todos estos buses son **buses paralelos** que involucran muchos cables. Un **bus** de **datos** clásico tiene 16 o 32 bits de ancho, pero podría tener hasta 128 bits o incluso más. Un bus **más ancho** significa una **mayor** tasa de **transferencia** de datos para cualquier frecuencia de operación determinada.
- Ya hemos visto que hay dos arquitecturas de procesador principales disponibles: **von Neumann** y **Harvard**.
  - El primero tiene un **espacio** de memoria **unificado** para instrucciones y transferencias de datos, mientras que el segundo tiene funcionalidades **separadas** con buses de datos y direcciones separados.

1.5.8 Interconnects

- Cuando los **buses** de instrucciones y datos son **comunes**, se utilizan **menos cables** pero es necesario realizar **transferencias secuenciales** para leer una instrucción y acceder a la memoria de datos.
  - Con una **arquitectura Harvard**, se pueden hacer ambas cosas al mismo tiempo, lo que ofrece un mejor rendimiento pero un mayor coste en términos de área de matriz.
  - Es responsabilidad del diseñador establecer el equilibrio correcto entre los recursos necesarios y el rendimiento esperado, es decir, seleccionar la arquitectura de procesador adecuada.
- Considerar la **interconectividad** significa garantizar que tenemos una visión general de todos los elementos de un sistema, el **ancho** de **banda** que se **requerirá** y el ancho de banda que estará **disponible** dentro de nuestro diseño.
- En sistemas más potentes, otra unidad que aún no se ha analizado es el controlador de acceso directo a memoria (**DMA**).
  - Esta unidad puede funcionar en **paralelo** con el **procesador** para **acceder** a la **memoria** de datos y **manejar** algunas **transferencias** y **operaciones** de **datos**.

1.5.8 Interconnects

- En dispositivos de alta gama, a menudo se incorpora un controlador **DMA** en **interfaces** programables específicas.
- En otras implementaciones, puede ser una **unidad genérica** que maneja muchas fuentes/destinos de datos: **memorias** e **interfaces** programables.
- Como unidad **solicitante**, tendrá muchos cables para todos los **buses** de **dirección/datos** y **control** disponibles.

Si se utiliza este tipo de unidad, se deben tomar algunas decisiones, principalmente en cuanto a si las **transferencias** de datos para el **procesador** se pueden realizar al **mismo tiempo** que el controlador **DMA** está **operando** o si los dos deben **actuar secuencialmente**.

- Esto puede tener un gran impacto en el rendimiento general y en el área del circuito integrado necesaria.

1.5.8 Interconnects

- La responsabilidad de tales **decisiones** arquitectónicas es principalmente del **fabricante** del **microcontrolador**.
- Debe proponer lo que cree que es la **mejor arquitectura** para obtener el **mejor rendimiento** y el **menor precio** y **consumo** de **energía**.
- Para los propósitos de este libro sobre **procesadores Arm Cortex-M**, la familia de bus **AMBA** (Advanced Microcontroller Bus Architecture) es la que se utiliza.
  - Las especificaciones **AMBA** están libres de regalías, son independientes de la plataforma y se pueden utilizar con cualquier arquitectura de procesador.
  - El uso de **AMBA** facilita el desarrollo correcto desde el primer momento de diseños multiprocesador con una gran cantidad de controladores y periféricos. La adopción generalizada significa que **AMBA** tiene un ecosistema global sólido de socios que garantiza la compatibilidad y escalabilidad entre componentes de propiedad intelectual de diferentes equipos de diseño y proveedores.

1.5.8 Interconnects

- Existen muchos otros diseños de buses que utilizan principios similares y están disponibles en el mercado, por ejemplo:
  - Espoleta (OpenCores)
  - Avalon en FPGA (Intel)
  - Protocolo de núcleo abierto (Accellera)
  - CoreConnect (IBM)

- El bus AMBA ha pasado por varias generaciones diferentes (AMBA 2, 3, 4, 5) y variaciones, incluidas AHB (AMBA High-Performance Bus), APB (Advanced Peripheral Bus), AXI (Advanced eXtensible Interface) y CHI (Coherent Hub). Interface), que se utiliza en sistemas multinúcleo totalmente coherentes y de alto rendimiento.
  - Antes de AMBA 2, no era posible que dos administradores tuvieran acceso simultáneo al mismo tiempo.

1.5.8 Interconnects

- Un **árbitro** proporcionó acceso exclusivo al bus a una unidad de **administrador** a la vez. Sin embargo, a través de la organización matricial del **bus administrador**, muchos **administradores** ahora pueden usar las arquitecturas de bus **AHB** o **AXI** para realizar transferencias **simultáneas** siempre que accedan a diferentes subordinados, lo que significa que pueden ejecutarse a **frecuencias** más **altas**.
- La idea detrás de todos los buses mencionados anteriormente es la **interconexión** más **sencilla** de **interfaces**, **procesadores**, **aceleradores** y **memorias**.
  - Esto permite a los diseñadores (empresas) desarrollar módulos certificados para uno o más de estos buses.
  - Los integradores de arquitecturas de bus específicas pueden comprar estos módulos y montar su propio SoC.
  - Estos estándares hacen que sea mucho más fácil construir un sistema plug-and-play que funcione al primer intento.

1.5.8 Interconnects

- Estos buses están diseñados para conexiones locales entre decenas de unidades, con muy pocos administradores.
  - Para los sistemas multinúcleo, se están desarrollando nuevas tecnologías como la network-on-a-chip (NoC), cuya idea es tener conexiones en serie de alta velocidad dentro de un circuito integrado, como las tendríamos a mayor escala con muchas computadoras conectadas entre sí.
  - Esto no es algo que ofrezca actualmente la familia Cortex-M, pero miremos de nuevo dentro de unos años...

1.6 System Architecture and Complexity

- ¡Divide y conquistaras! Después de una descripción general de las diferentes partes de un núcleo de procesador desde el punto de vista del sistema, ahora veremos los dos componentes Cortex-M más simples, M0 y M0+, y sus diferentes niveles de integración.
  - Como hemos visto, diferentes fabricantes pueden diseñar microcontroladores con su propia arquitectura para el procesador y las interfaces programables, o pueden adoptar una arquitectura estandarizada proporcionada por empresas especializadas como Arm y utilizar un procesador como el Cortex-M, el Cortex-A más potente, o el Cortex-R más rápido para aplicaciones de tiempo real.
- Una vez seleccionada la familia de procesadores principal, es necesario realizar elecciones más detalladas en cuanto al consumo de energía, la potencia informática y, por supuesto, el precio.
  - Si tomamos el ejemplo de un procesador Cortex-M0 o Cortex-M0+ simple de bajo consumo, hay muchas opciones disponibles para la configuración general de los componentes.

1.6 System Architecture and Complexity

- Así, la Figura 1.22 representa una vista general de un **microcontrolador/SoC** basado en un microprocesador **Cortex-M0**.
- La configuración general la realiza un fabricante, la arquitectura principal la proporciona **Arm**.
- Hay dos áreas principales que contienen varios elementos de nivel inferior:
  - El componente **Cortex-M0**, que incluye:
    - Procesador **Cortex-M0**, compuesto por:
      - **Cortex-M0 processor core**
      - **nested vector interrupt controller**

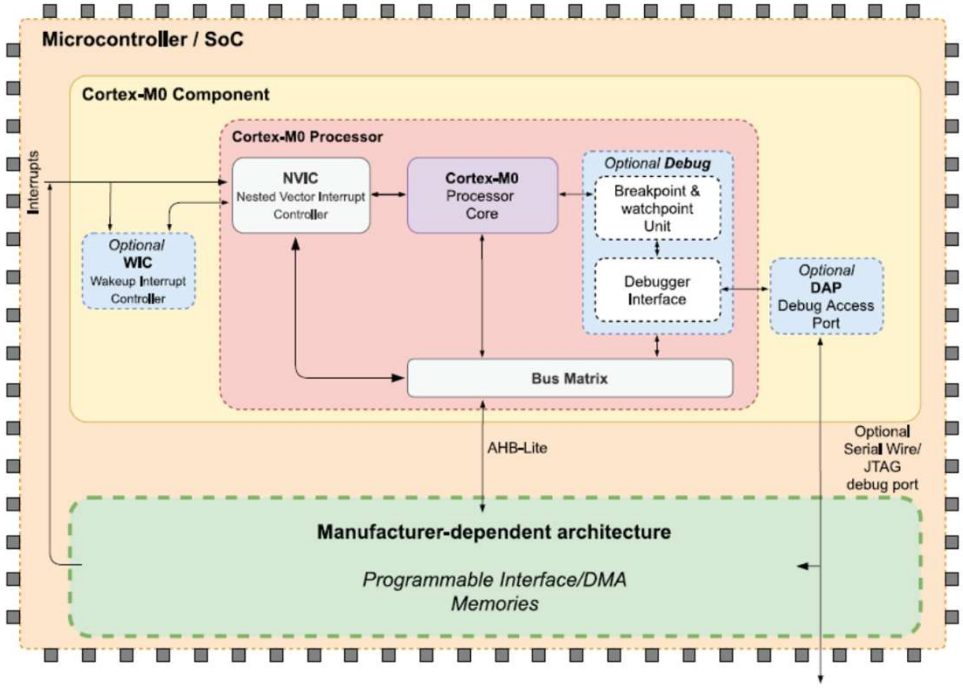


Figure 1.22 Microcontroller based on Cortex-M0

1.6 System Architecture and Complexity

- unidades de **debug** opcionales:
  - unidad de **breakpoint** and **watchpoint**
  - interfaz del **debugger**
  - **bus matrix**
- elementos opcionales, que incluyen:
  - **wakeup interrupt controller**
  - **debug access port** (Serial Wire or JTAG)

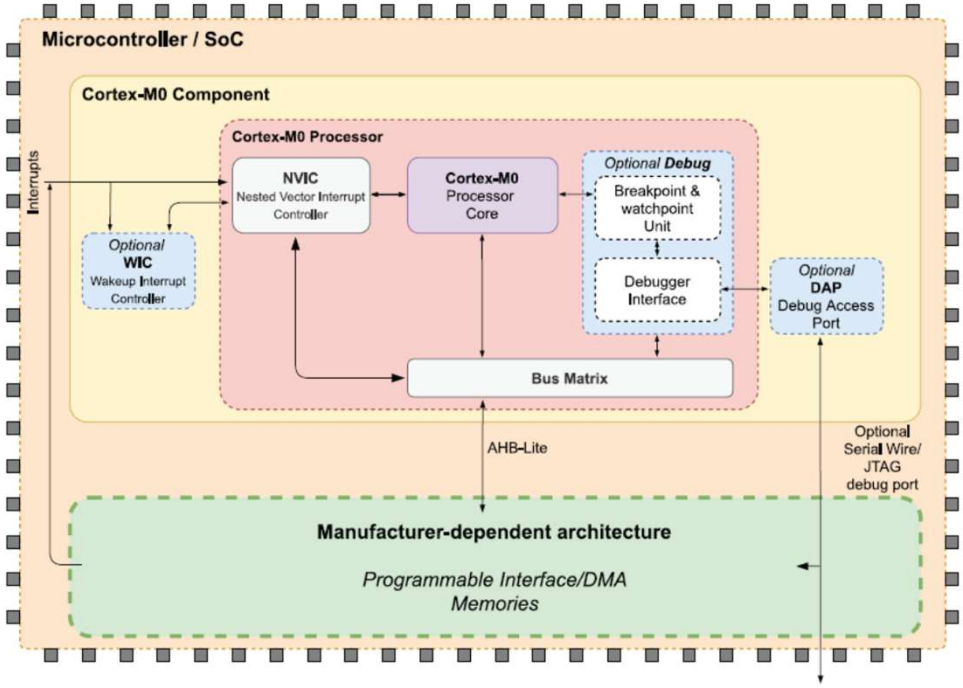


Figure 1.22 Microcontroller based on Cortex-M0

1.6 System Architecture and Complexity

- El componente que depende del fabricante, incluido:
  - memorias
    - Flash - RAM
    - no volátil reprogramable (opción)
    - generación de **reloj**
- todas las **interfaces** programables
- arquitectura de **bus** interna

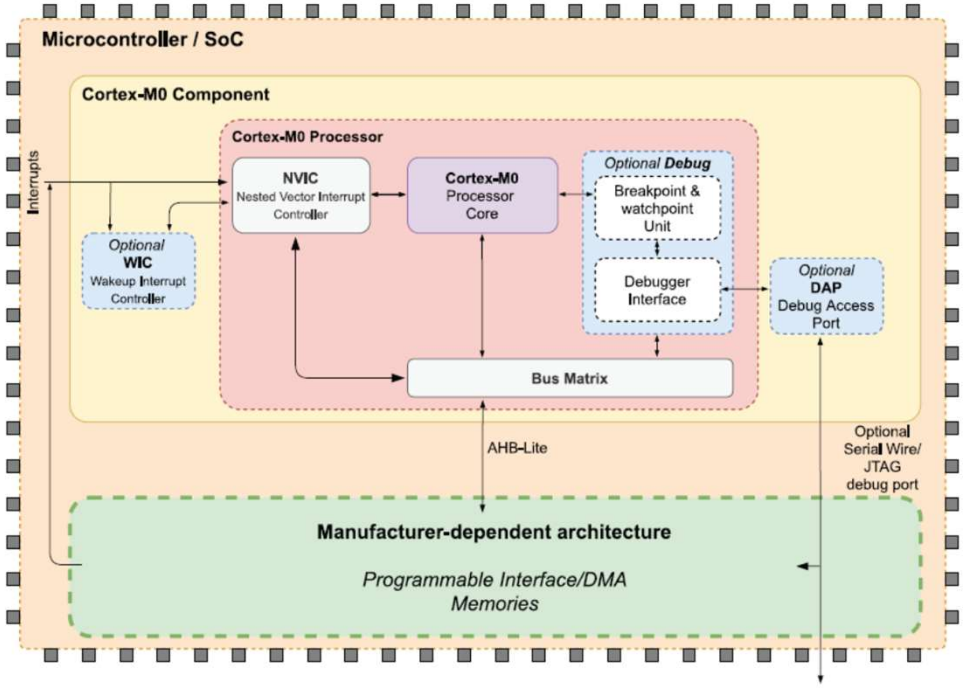


Figure 1.22 Microcontroller based on Cortex-M0

1.6 System Architecture and Complexity

- Hay muchas opciones para el diseñador de microcontroladores y algunos dolores de cabeza para un diseñador de sistemas que necesita seleccionar un microcontrolador específico para el diseño de su aplicación.
- Veamos el mayor nivel de complejidad involucrado con un microcontrolador basado en Cortex-M0+, como se muestra en la Figura 1.23.
- Podemos ver que tiene algunas unidades opcionales nuevas, principalmente una MPU para protección de áreas en el componente del procesador, pero también un micro trace buffer (MTB) y una extensión de bus para un puerto de I/O de un único ciclo.

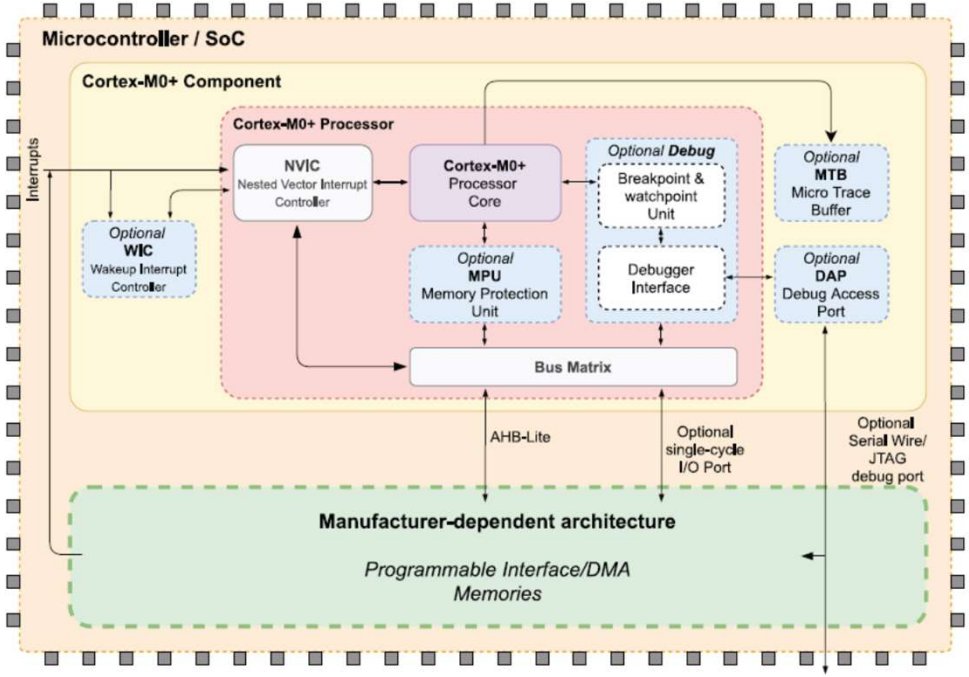


Figure 1.23 Microcontroller based on Cortex-M0+

1.6 System Architecture and Complexity

- Cortex-M0 y -M0+ ofrecen el consumo de energía más bajo de la familia Cortex-M, pero con un rendimiento menor y un conjunto de instrucciones mínimo.
- Podemos ir mucho más lejos con Cortex M3, M4, M4F, M23, M33, M7, M55 y algunos componentes Cortex-M de próxima aparición, pero la complejidad también aumenta con características adicionales, comenzando con el núcleo M3 y luego el IEEE-754 opcional. Unidad de punto flotante (FPU) de 32 bits del M4, como se muestra en la Figura 1.24. Aquí, el compilador utiliza las instrucciones asociadas para realizar operaciones de punto flotante mucho más rápido.

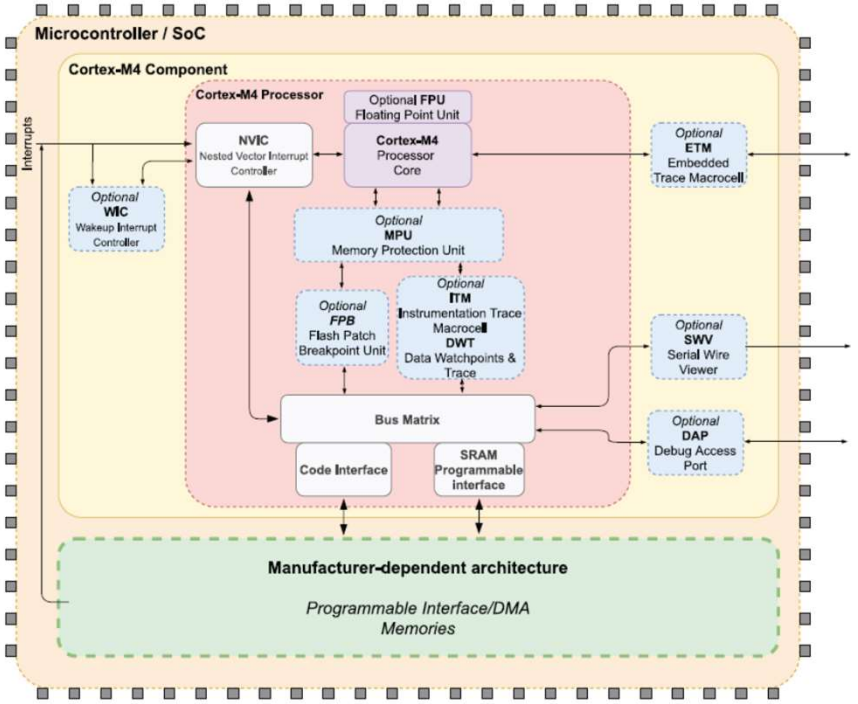


Figure 1.24 Microcontroller based on Cortex-M4/M4F

1.6 System Architecture and Complexity

- Podemos ver que ahora se pueden incluir muchas **funciones** de **depuración adicionales** en el componente **Cortex**, una característica valiosa para el seguimiento de programas.
- Todas las piezas opcionales que se muestran en la Figura 1.24 dependen para su implementación del fabricante por lo que, nuevamente, es necesario estar muy familiarizado con la documentación completa del dispositivo específico utilizado.

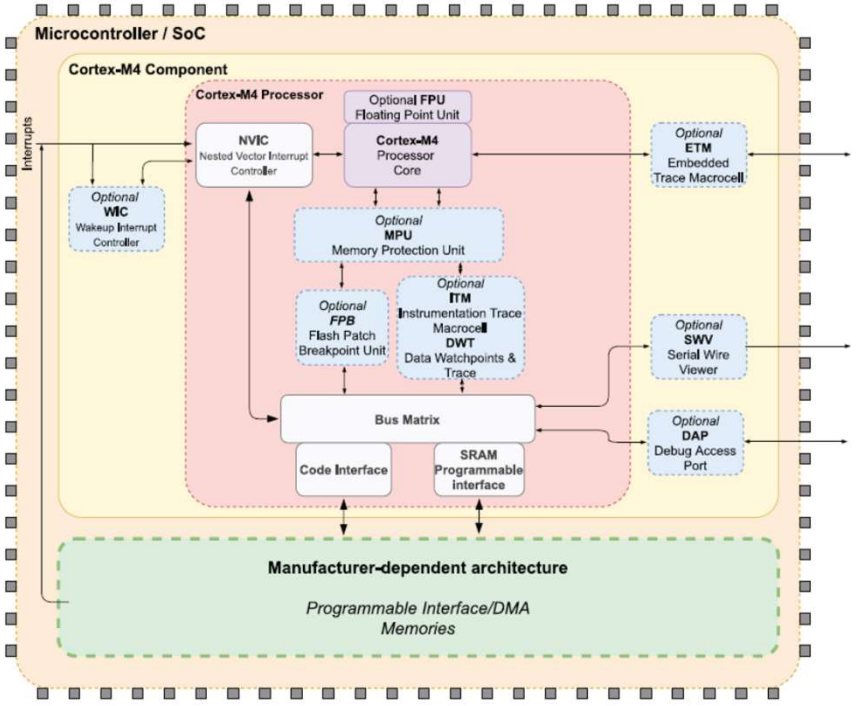


Figure 1.24 Microcontroller based on Cortex-M4/M4F

1.6 System Architecture and Complexity

- A medida que evolucionan las nuevas generaciones de procesadores, su complejidad generalmente aumenta.
  - Se utilizan diferentes arquitecturas internas y se ofrecen diferentes procesadores a los fabricantes integradores.
- Arm propone muchas arquitecturas para usar con los diferentes procesadores de la familia Cortex-M (como se resume en la Tabla 1.1):
  - Arm **v6-M** on Cortex-M0/M0+
  - Arm **v7-M** on Cortex-M3
  - Arm **v7E-M** on Cortex-M4
  - Arm **v8-M** (with TrustZone) on Cortex-M23/M33
  - Arm **v8.1-M** on Cortex-M55.

1.6 System Architecture and Complexity

Table 1.1 Comparison of Cortex-M processors

Cortex-M	Architecture	Arch. Type	MPU	Caches	Pipeline	FPU	DSP	Interface	Security	Max no. IRQs
M0	Arm v6-M	von Neumann	-	-	3	-	-	AHB-Lite	-	1..32
M0+	Arm v6-M	von Neumann	option	-	2	-	-	AHB-Lite Optional single-cycle I/O port	-	1..32
M3	Arm v7-M	Harvard	option	-	3	-	-	AHB-Lite APB	-	1..240
M4	Arm v7E-M	Harvard	option	-	3	option	DSP	AHB-Lite APB	-	1..240
M7	Arm v7-M	Harvard	option	I / D	3..6	option	DSP	AXI AHBS/AHBP TCM	-	1..240
M23	Arm v8-M	Harvard	option	-	3		DSP	AMBA 5 AHB 5 Optional single-cycle I/O port	TrustZone	1..240
M33	Arm v8-M	Harvard	option	-	3	option	DSP	AMBA 5 AHB 5	TrustZone	1..480
M55	Arm v8.1-M	Harvard	option	-	4	option	DSP/SIMD Helium custom instruction	AXI5, M-AXI AHB5, S-AHB AHB5, P-AHB	TrustZone	1..480

TCM: Tightly coupled memory

- Para esta clase de procesador, el ancho del registro es de **32 bits**.
- Para la familia **Cortex-A**, más potente, el tamaño del registro suele ser también de **32 bits**, pero se propone una nueva generación con la arquitectura **AArch64** que tendría **64 bits** de ancho.

1.7 Software for Embedded System Development

- Cuando llega el momento de desarrollar software para el mundo de los sistemas embebidos, es necesario responder muchas preguntas.
- Si queremos desarrollar una aplicación para un sistema embebido basado en la familia Cortex-M, ¿cómo empezamos?
- Primero, debemos entender que estamos haciendo un desarrollo cruzado.
  - Esto significa que el software se desarrolla en una estación de trabajo, como una PC, y se descarga a la memoria del sistema embebido a través de un enlace serie, a menudo un enlace JTAG o Serial Wire Debug (SWD) a través de una interfaz USB en la PC.
- Existen muchas herramientas de desarrollo para las familias Cortex.
  - Como mínimo, necesitamos un editor de texto, un compilador y/o ensamblador, un enlazador y una forma de descargar el código en la memoria.
- Así era como se desarrollaban los programas hace 50 años y todavía se hace en la actualidad.

1.7 Software for Embedded System Development

- Otra herramienta obligatoria es un **depurador**, para poder iniciar la ejecución de un programa, insertar **breakpoints** para detener el programa en lugares clave, avanzar **paso** a **paso** a través del código, **examinar** la memoria, **desensamblar** el código del programa y **observar** los registros internos del procesador y las interfaces programables. Idealmente, también nos gustaría una forma de **medir** el **consumo** de **energía**.
- Muchas empresas ofrecen herramientas de desarrollo integradas con un editor codificado por colores adaptado al lenguaje de programación utilizado.
- Para esta clase de procesador, la programación a nivel de ensamblador se utiliza con bastante frecuencia desde el principio, pero C/C++ es claramente el lenguaje principal para los microcontroladores.
- Las herramientas que admiten MicroPython también están cada vez más disponibles; es un lenguaje interpretado, por lo que es un poco más lento, pero más fácil de usar.

1.7 Software for Embedded System Development

- Las herramientas de desarrollo como ésta suelen estar basadas en un entorno llamado Eclipse.
  - Esta es una herramienta de desarrollo de código abierto desarrollada originalmente por IBM y posteriormente puesta en el dominio público.
  - Las herramientas de compilación y depuración de GNU, gcc y gdb, están disponibles gratuitamente y muy a menudo se incluyen en las herramientas de desarrollo.
- Los proveedores de herramientas especializados ofrecen muchas otras herramientas; por ejemplo, Keil, Altium TASKING, SEGGER, Arm Development Studio, Emprog ThunderBench, Green Hills MULTI, IAR Embedded Workbench, Lauterbach TRACE32, así como la plataforma de desarrollo Mbed.
  - En general, no están disponibles de forma gratuita o tienen límites en la cantidad de código que pueden manejar, pero pueden ser muy potentes y universales, y admiten casi todos los fabricantes de procesadores.

1.7 Software for Embedded System Development

- Algunos fabricantes de componentes ponen sus herramientas a disposición de forma gratuita, al menos para un tamaño limitado de software.
  - Los ejemplos incluyen: Code Composer Studio de Texas Instruments; LPCXpresso, MCUXpresso y Kinetis Design Studio para procesadores NXP; Herramientas STM32Cube para procesadores ST.
- Una vez abordadas las herramientas de desarrollo, la pregunta pendiente en materia de software a responder a la hora de desarrollar un sistema embebido es si queremos o no utilizar un sistema operativo (SO):
  - En caso afirmativo, ¿qué categoría?
    - ¿Es de tiempo real Hard?, ¿ Es de tiempo real Soft?
    - ¿No es en tiempo real?
    - ¿Basado en Linux (normalmente requiere una clase de procesador superior, como Cortex-A)?

1.7 Software for Embedded System Development

- Si, en cambio, optamos por la codificación básica (ejecutar instrucciones directamente en el hardware), ¿queremos:
  - ¿Nuestra propia solución bare-metal totalmente desarrollada en casa?
  - ¿Para hacer uso de las bibliotecas de códigos disponibles?
  - ¿Usar un planificador kernel?

- Se debe considerar seriamente el uso de un sistema operativo o un kernel; después de un tiempo de aprendizaje inicial, hará que el proceso de desarrollo sea más fácil y rápido.
  - Muchos sistemas operativos y kernels están disponibles en el mercado.
  - Un ejemplo particular es el Estándar de interfaz de software del microcontrolador Cortex (CMSIS), una capa de abstracción de hardware independiente del proveedor para microcontroladores basados en procesadores Arm Cortex.

1.7 Software for Embedded System Development

- Esto ayuda en el uso y programación de los elementos de bajo nivel de los microcontroladores.
- Está estandarizado y utilizado por casi todos los fabricantes y sistemas de herramientas de desarrollo.
- Muchas otras empresas ofrecen sistemas operativos certificados para aplicaciones basadas en microcontroladores en tiempo real; los ejemplos incluyen µC/OS-III, Keil RTX y FreeRTOS.
- Se puede encontrar más información sobre este tema en los Capítulos 3 y 10.

1.8 Summary

- En el transcurso de este capítulo introductorio, hemos examinado los diversos temas que cubrirá este libro, comenzando desde una visión de muy alto nivel de un sistema informático con servidores, nubes y acceso a Internet para PC y dispositivos IoT.
  - Todos estos elementos utilizan procesadores de diferente complejidad.
- Hemos dado una visión de los diferentes niveles que se encuentran en un sistema embebido en forma de microcontrolador basado en un procesador, memorias e interfaces programables. Hemos analizado los principios de interrupciones y gestión y protección de la memoria, y hemos proporcionado información básica sobre las interconexiones.
  - Presentamos la familia de microprocesadores Cortex-M y describimos brevemente su evolución de Cortex-M0 a Cortex-M7.
- El siguiente capítulo presenta la microelectrónica y los conceptos básicos del diseño de chips.

Referencias

- Fundamentals of System-on-Chip Design on Arm Cortex-M Microcontrollers - René Beuchat (Author), Florian Depraz (Author), Andrea Guerrieri (Author), Sahand Kashani (Author)
- The Definitive Guide to ARM Cortex -M0 and Cortex-M0+ Processors – Joseph Yiu (Author)
- Arm Architecture for the Digital World - https://www.arm.com/architecture
- What is an Arm processor? - https://www.techtarget.com/whatis/definition/ARM-processor
- STM32 32-bit Arm Cortex MCUs - https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html

Manos a la obra con el . . .

. . . Proyecto Intermedio

. . . un enfoque centrado en la práctica propia de la carrera más que en el desarrollo teórico disciplinar, con eje en la participación de las y los estudiantes


Las y los estudiantes preguntarán:
¿en qué lío nos metimos?

¡Muchas gracias!

¿Preguntas?

. . .

Consultas a: jcruz@fi.uba.ar