TEMA 12: ENSAMBLADOR

Sistemas Digitales basados en Microprocesador

Grado en Ingeniería Telemática

© Raúl Sánchez Reíllo

ÍNDICE

Representación RTL

- El *Register Transfer Language* (RTL) es un lenguaje que simboliza la transferencia de datos entre registros
- Se utiliza, entre otras cosas, para representar el funcionamiento de un determinado proceso
  - La ejecución de una instrucción
  - El funcionamiento de la Unidad de Control
  - La secuencia de pasos que conlleva una interrupción
  - etc.
- Las variantes son muchas, por lo que se pueden encontrar especificaciones en RTL muy distintas de unos autores a otros
  - Mientras haya consistencia entre las representaciones, se puede considerar como válida

Representación RTL

- En este curso se va a utilizar una representación muy simplificada:
  - Paso del contenido de un registro a otro:
    - MAR ⇐ PC ; lleva el contenido del PC al MAR
  - Paso del contenido de la memoria a un registro (operación de lectura)
    - MBR ⇐ (MAR) ; lleva el contenido que tiene la memoria en la posición indicada por MAR, al registro MBR
  - Paso del contenido de un registro a memoria (operación de escritura)
    - (MAR) ⇐ MBR ; lleva el contenido del registro MBR a la memoria, a la posición indicada por MAR
  - Operaciones con el operando fuente
    - PC ⇐ PC + 1 ; incrementa el PC en una unidad

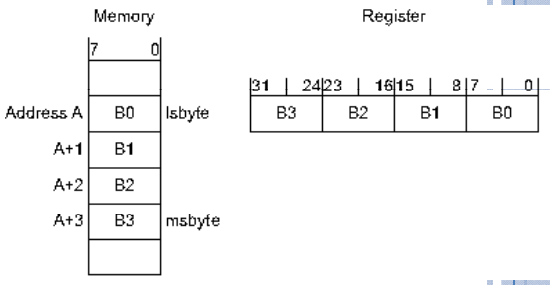
TEMA 12: ENSAMBLADOR
Modelo del Programador

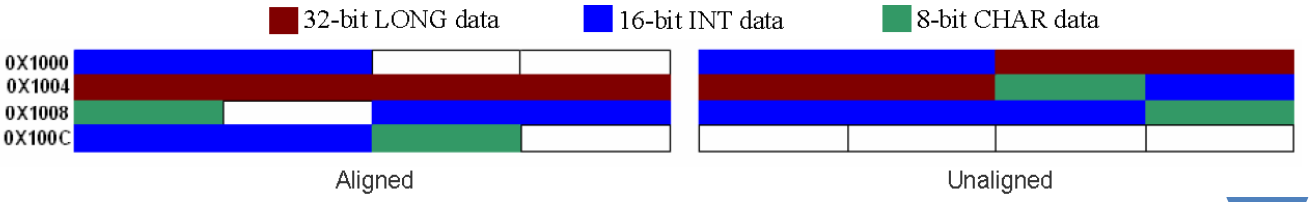
MODELO DEL PROGRAMADOR DEL CORTEX-M3

- A la hora de programar una CPU, muchos elementos internos se ocultan (por ejemplo registros como el MAR, MBR, IR, etc)
- El Modelo de Programador establece el conjunto de elementos que son necesarios conocer, de la arquitectura interna de la CPU, para realizar programas
  - Modelo de memoria
  - Tipos de Datos
  - Modos de Operación
  - Registros
  - Mecanismos de Programación (se verán más adelante)

Modelo del Programador del Cortex-M3

- Modelo de Memoria:
  - El Cortex-M3 direcciona bytes (es decir sus 32 bits de direcciones son de datos de 8 bits)
  - Se utiliza codificación little-endian
  - Los datos se pueden almacenar de forma alineada o no alineada.
  - El Cortex-M3 puede trabajar con 3 tipos de datos:
    - Palabra (***word***) de 32 bits, que en memoria debe estar colocada alineada en múltiplos de 4
    - Semipalabra (***halfword***) de 16 bits, que en memoria debe estar colocada alineada en múltiplos de 2
    - ***Byte***, que en memoria puede colocarse en cualquier dirección





Modelo del Programador del Cortex-M3

- Tipos de Datos:
- Modos de Operación:
  - Thread mode: para ejecutar aplicaciones software. Es el modo en el que entra la CPU cuando sale del estado de reset
  - Handler mode: para gestionar las excepciones. El procesador vuelve al modo Thread cuando ha terminado de procesar la excepción
- Niveles de privilegio para la ejecución de software:
  - Unprivileged: en este nivel, el software:
    - Tiene acceso limitado a las instrucciones MSR y MRS, y no puede utilizar la instrucción CPS
    - No puede acceder al reloj del sistema, al NVIC o al bloque de control del sistema
    - Puede tener acceso restringido a memoria o periféricos
  - Privileged: el software tiene acceso a todas las instrucciones y todos los recursos

TEMA 12: ENSAMBLADOR
Mecanismos de Programación

MECANISMOS DE PROGRAMACIÓN

- El desarrollo de un programa se realiza mediante la concatenación de instrucciones que se ejecutarán siguiendo un flujo lineal
- Sin embargo, ese flujo lineal se puede alterar mediante:
  - Saltos
  - Subrutinas (al igual que las llamadas a función en C)
  - Interrupciones (al igual que ya se ha visto)
- Saltos:
  - El programa continúa la ejecución en otro punto de la memoria
  - Se provoca mediante una instrucción de salto condicional B{cond}, en la que se indica la dirección a la que saltar
  - Si se cumple la condición, esa instrucción realiza el cambio del valor del PC por el de la posición donde tiene que seguir el programa

MECANISMOS DE PROGRAMACIÓN

- Subrutinas:
  - Se trata de un salto temporal, volviendo posteriormente a la instrucción siguiente a la del salto a subrutina
  - Se utilizan para estructurar el programa en funciones a las que se llama repetidas veces (optimizan código)
  - La CPU obtiene la dirección de salto de los bits que acompañan al opcode
  - Antes de realizar el salto, la CPU debe guardar el contenido del PC, para recuperarlo posteriormente (cuando se de la instrucción de volver)
  - Las instrucciones, en el caso del ARM7, se denominan BL{cond} (*branch with link*), y guardan el valor del PC (R15) en el registro de link LR (R14)

MECANISMOS DE PROGRAMACIÓN

- Interrupciones:
  - Es un mecanismo por el cual se permite que la CPU realice determinadas operaciones, cuando haya ocurrido un determinado evento externo
  - Es como una subrutina, pero que no es llamada por el programador (no es una instrucción concreta en el programa)
    - Cada vez que se ejecuta, el programa principal puede estar en una instrucción u otra
  - Cuando ocurre ese evento se pasa a ejecutar una subrutina, conocida como **Rutina de Atención a la Interrupción** o como **Rutina de Servicio**
    - Antes de entrar a ejecutar la Rutina de Servicio, la CPU tiene que guardar el valor del PC y el contexto (que en su versión mínima es el SR)
      - En algunas CPUs también se inhiben las interrupciones
    - Se termina la ejecución de la Rutina de Servicio recuperando el contexto (el valor del PC y del registro de estado)

MECANISMOS DE PROGRAMACIÓN

- Interrupciones:
  - La Rutina de Servicio es algo que se puede ejecutar en cualquier momento, y que está interrumpiendo el curso normal del programa
    - Entre la CPU y el programador se debe garantizar que, cuando se vuelve de la Rutina de Servicio, la CPU se encuentra en las mismas condiciones que si no se hubiese producido la interrupción (salvo modificaciones intencionadas)
    - Debe ser una rutina razonablemente corta y efectiva, que no deje al programa en una espera demasiado larga (un cuelgue)
    - Tradicionalmente tendrá que comunicar al programa principal que ha ocurrido una interrupción y las consecuencias de la misma
  - Una CPU puede tener varias fuentes de interrupción, y pueden estar activas más de una al mismo tiempo
    - Algunas estarán activadas por defecto, mientras otras habrá que activarlas

TEMA 12: ENSAMBLADOR

Juego de Instrucciones del Cortex-M3

JUEGO DE INSTRUCCIONES DEL CORTEX-M3

- El Juego de Instrucciones del Cortex-M3 es reducido en número de instrucciones, pero muy amplio en variantes de las mismas
  - Por ejecución condicional
  - Por modo de direccionamiento y tamaño de la palabra utilizada
- Para simplificar la explicación se van a seguir las siguientes pautas:
  - Se comentará primero las distintas instrucciones
  - Después se detallarán:
    - la forma de referirse a los distintos operandos (direccionamiento)
    - las modificaciones del CPSR si S=1 (si se especifica S en la instrucción)
    - las condiciones de ejecución para las instrucciones
  - Las instrucciones se van a desglosar en:
    - Transferencia de Datos
      - Entre memoria y CPU
      - Entre registros
    - Aritméticas y Lógicas
    - Control
    - Misceláneas
  - Los desplazamientos y rotaciones no son instrucciones específicas, sino forma de tratar uno de los operandos (<Op2>)

JUEGO DE INSTRUCCIONES DEL CORTEX-M3

- Simplificaciones a realizar en el curso
  - Todas las instrucciones van a poder ser condicionales
    - Esto se refleja en que cada mnemónico de instrucción va seguido, opcionalmente por un código de condición {cond}
    - Durante este curso **no se van a utilizar los códigos de condición en ninguna instrucción salvo en las de Salto Condicional**
      - Por tanto se ignorará en todas la parte {cond} salvo en B{cond}
  - Muchas instrucciones pueden determinar si modifica o no el registro de estado
    - **En este curso se va a considerar que todas aquellas instrucciones que puedan modificar el registro de estado, lo van a hacer**
      - Toda instrucción que pueda utilizar el parámetro {S}, se le pondrá de forma fija (MOVS, ADDS, etc.)
  - Sobre la posibilidad de que una instrucción pueda realizar más de una operación (tal como un desplazamiento unido a una suma), esto SI que se va a utilizar, y por tanto no se simplificará el formato del <op2>
  - La utilización avanzada de la Arquitectura Interna (gestiones de registros de estados, salvaguardas, pilas, etc.) se va a simplificar al máximo, utilizándose las recomendaciones dadas en la plantilla expuesta al final de este tema
- Hay que tener en cuenta que hay fundamentalmente dos tipos de instrucciones:
  - Transferencia de datos con memoria: utilizará un operando que llamaremos <am2> o <am3>, el cual está relacionado con cómo se indica la dirección de memoria (esto se verá más adelante)
  - De tipo general: el segundo operando puede tener muchas variantes, por lo que normalmente se va a denotar como <op2>, y nunca será una posición de memoria

JUEGO DE INSTRUCCIONES DEL CORTEX-M3

- Transferencia de Datos con Memoria:

Mnemónico	Función	Sintaxis	RTL	N	Z	C	V
LDR (Load Register)	Carga una *word*	LDR{cond} Rd, <am2>	Rd ⬅ (<am2>)$_{32}$	-	-	-	-
	Carga un *byte*	LDR{cond}B Rd, <am2>	Rd ⬅ 000000:(<am2>)$_{8}$	-	-	-	-
	Carga un *byte* con signo	LDR{cond}SB Rd, <am3>	Rd ⬅ (<am3>)$_{8,sign\_ext32}$	-	-	-	-
	Carga una *halfword*	LDR{cond}H Rd, <am3>	Rd ⬅ 0000:(<am3>)$_{16}$	-	-	-	-
	Carga una *halfword* con signo	LDR{cond}SH Rd, <am3>	Rd ⬅ (<am3>)$_{16,sign\_ext32}$	-	-	-	-
LDM	Carga múltiples registros	*Hay 7 variantes*		-	-	-	-
STR (Store Register)	Graba una *word*	STR{cond} Rd, <am2>	(<am2>)$_{32}$ ⬅ Rd	-	-	-	-
	Graba un *byte*	STR{cond}B Rd, <am2>	(<am2>)$_{8}$ ⬅ Rd$_{7..0}$	-	-	-	-
	Graba una *halfword*	STR{cond}H Rd, <am3>	(<am3>)$_{16}$ ⬅ Rd$_{15..0}$	-	-	-	-
STM	Graba múltiples registros	*Hay 6 variantes*		-	-	-	-

JUEGO DE INSTRUCCIONES DEL CORTEX-M3

- Transferencia de Datos entre Registros:

Mnemónico	Función	Sintaxis	RTL	N	Z	C	V
MOV	Mueve un dato	MOV{cond}{S} Rd, <op2>	Rd ⬅ <op2>	x	x	x	-
MVN	Mueve el negado del dato	MVN{cond}{S} Rd, <op2>	Rd ⬅ !<op2>	x	x	x	-
PUSH	Mete registros en la pila	PUSH{cond} lista_de_reg	SP ⬅ SP-4 (SP) ⬅ lista_de_reg[i]	-	-	-	-
POP	Saca registros de la pila	POP{cond} lista_de_reg	lista_de_reg[i] ⬅ (SP) SP ⬅ SP+4	-	-	-	-
MRS	Mueve el SPSR a un Rd	MRS{cond} Rd, SPSR	Rd ⬅ SPSR	-	-	-	-
	Mueve el CPSR a un Rd	MRS{cond} Rd, CPSR	Rd ⬅ SPSR	-	-	-	-
MSR	Carga el SPSR ({f} indica que grupo/s se actualizan: c=[7:0], x=[15:8], s=[23:16], f=[31:24])	MSR{cond} SPSR_{f}, Rm	SPSR ⬅ Rm	-	-	-	-
		MSR{cond} SPSR_{f}, #im	SPSR ⬅ #im	-	-	-	-
	Carga el CPSR	MSR{cond} CPSR_{f}, Rm	CPSR ⬅ Rm	x	x	x	x
		MSR{cond} CPSR_{f}, #im	CPSR ⬅ #im	x	x	x	x

Juego de Instrucciones del Cortex-M3

- Aritméticas:

Mnemó.	Función	Sintaxis	RTL	N	Z	C	V
ADD	Suma	ADD{cond}{S} Rd, Rn, <op2>	Rd ⬅ Rn + <op2>	x	x	x	x
ADC	Suma con acarreo	ADC{cond}{S} Rd, Rn, <op2>	Rd ⬅ Rn + <op2> + C	x	x	x	x
SUB	Resta	SUB{cond}{S} Rd, Rn, <op2>	Rd ⬅ Rn - <op2>	x	x	x	x
SBC	Resta con acarreo	SBC{cond}{S} Rd, Rn, <op2>	Rd ⬅ Rn - <op2> - !C	x	x	x	x
RSB	Resta (inversa)	RSB{cond}{S} Rd, Rn, <op2>	Rd ⬅ <op2> - Rn	x	x	x	x
MUL	Multiplica	MUL{cond}{S} Rd, Rm, Rs	Rd ⬅ Rm * Rs	x	x	x	x
MLA	Multiplica y Acumula	MLA{cond}{S} Rd, Rm, Rs, Rn	Rd ⬅ Rm * Rs + Rn	x	x	?	x
SDIV	Divide con signo	SDIV{cond}{S} Rd, Rn, Rm	Rd ⬅ Rn / Rm	x	x	x	x
UDIV	Divide sin signo	UDIV{cond}{S} Rd, Rn, Rm	Rd ⬅ Rn / Rm	x	x	x	x
UMULL	Multiplica unsigned long	UMULL{cond}{S} RdLo, RdHi, Rm, Rs	RdHi:RdLo ⬅ Rm * Rs	x	x	?	?
UMLAL	Multiplica y Acumula unsigned long	UMLAL{cond}{S} RdLo, RdHi, Rm, Rs	RdHi:RdLo ⬅ Rm * Rs + RdHi:RdLo	x	x	?	?
SMULL	Multiplica signed long	SMULL{cond}{S} RdLo, RdHi, Rm, Rs	RdHi:RdLo ⬅ Rm * Rs	x	x	?	?
SMLAL	Multiplica y Acumula signed long	SMLAL{cond}{S} RdLo, RdHi, Rm, Rs	RdHi:RdLo ⬅ Rm * Rs + RdHi:RdLo	x	x	?	?

Juego de Instrucciones del Cortex-M3

- Lógicas:

Mnemó.	Función	Sintaxis	RTL	N	Z	C	V
AND	And lógico	AND{cond}{S} Rd, Rn, <op2>	Rd ⬅ Rn & <op2>	x	x	x	-
EOR	Or exclusivo	EOR{cond}{S} Rd, Rn, <op2>	Rd ⬅ Rn ^ <op2>	x	x	x	-
ORR	Or lógico	ORR{cond}{S} Rd, Rn, <op2>	Rd ⬅ Rn | <op2>	x	x	x	-
BIC	Bit clear	BIC{cond}{S} Rd, Rn, <op2>	Rd ⬅ Rn & !<op2>	x	x	x	-
ORN	Or lógico negado	ORN{cond}{S} Rd, Rn, <op2>	Rd ⬅ Rn | !<op2>	x	x	x	-

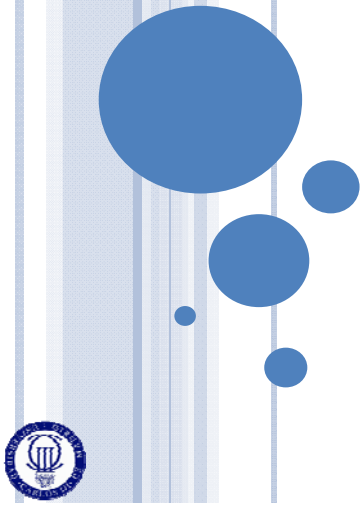
- Control de Flujo:

Mnemó.	Función	Sintaxis	RTL	N	Z	C	V
CMP	Compara	CMP{cond} Rn, <op2>	Rn - <op2>	x	x	x	x
CMN	Compara con negado	CMN{cond} Rn, <op2>	Rn + <op2>	x	x	x	x
TST	Test	TST{cond} Rn, <op2>	Rn & <op2>	x	X	x	X
TEQ	Test de equivalencia	BIC{cond} Rn, <op2>	Rn ^ <op2>	x	x	x	x
B	Branch	B{cond} etiqueta	PC ⬅ PC + <dir_rel>	-	-	-	-
BL	Branch con Link	BL{cond} etiqueta	LR ⬅ PC; PC ⬅ PC + <dir_rel>	-	-	-	-
BX	Branch y cambio de juego de instrucciones	BX{cond} Rn	T ⬅ Rn[0]; PC ⬅ Rn & 0xFFFFFFFE	-	-	-	-

JUEGO DE INSTRUCCIONES DEL CORTEX-M3

- Códigos para ejecución condicional:

Suffix	Flags	Meaning
EQ	Z = 1	Equal
NE	Z = 0	Not equal
CS or HS	C = 1	Higher or same, unsigned ≥
CC or LO	C = 0	Lower, unsigned <
MI	N = 1	Negative
PL	N = 0	Positive or zero
VS	V = 1	Overflow
VC	V = 0	No overflow
HI	C = 1 and Z = 0	Higher, unsigned >
LS	C = 0 or Z = 1	Lower or same, unsigned ≤
GE	N = V	Greater than or equal, signed ≥
LT	N != V	Less than, signed <
GT	Z = 0 and N = V	Greater than, signed >
LE	Z = 1 and N != V	Less than or equal, signed ≤
AL	Can have any value	Always. This is the default when no suffix is specified.



TEMA 12: ENSAMBLADOR
Modos de Direccionamiento

Modos de Direccionamiento

- Para ejecutar una instrucción hay que indicarle donde se encuentran sus operandos
  - Hay que indicarle la dirección donde se encuentran (sus **direcciones efectivas**)
- Para indicar la dirección donde se encuentra el operando se pueden utilizar distintos **modos de direccionamiento**
- Genéricamente se suelen mencionar lo siguientes:
  - Inherente o Implícito
  - Inmediato o Literal
  - Directo o Absoluto
  - Indirecto
  - Indirecto con Desplazamiento
  - Indexado
  - Relativo a PC

Modos de Direccionamiento

- Inherente o Implícito:
  - El operando se encuentra especificado por el propio código de la instrucción (opcode)

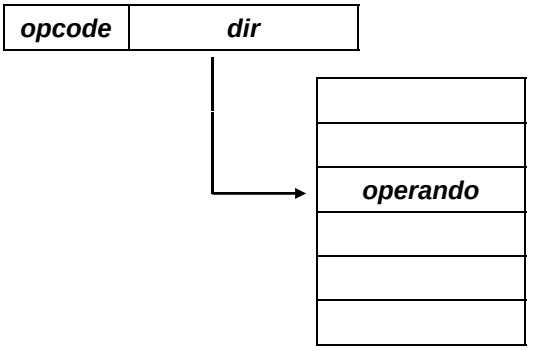
opcode	---

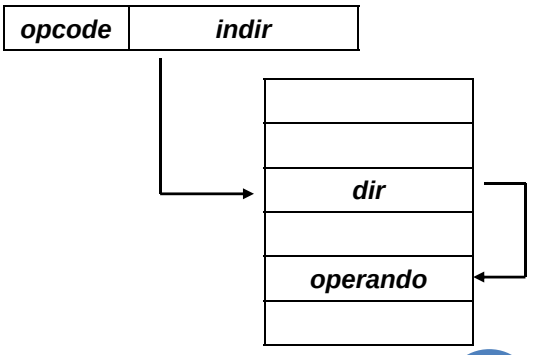
- Inmediato o Literal:
  - El dato se encuentra en la propia instrucción, en los bits siguientes al opcode
    - El dato se suele poner precedido de #
    - Ejemplo MOVS R2, #3
      - R2 ⇐ 3

opcode	*dato*

Modos de Direccionamiento

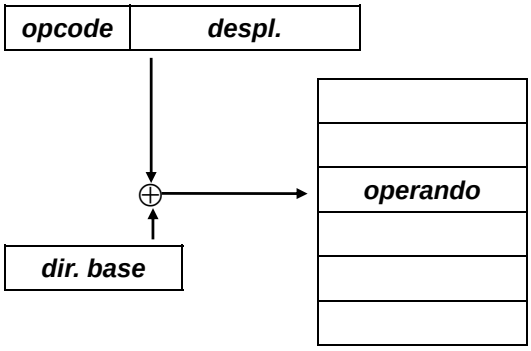
- **Directo o Absoluto:**
  - El operando se encuentra en la dirección de memoria indicada en la instrucción (tras el opcode)
    - Ejemplo: LDR R1, **0x00000015**
      - R1 ⇐ (0x00000015)
      - Este ejemplo es ficticio, ya que este modo de direccionamiento no usado en el ARM7
  - Existe una variante denominada "Directo a Registro", que es indicar que el dato se encuentra en uno de los registros internos del micro
    - Por ejemplo el R1 del primer operando del ejemplo anterior
- **Indirecto:**
  - El operando se encuentra en la dirección de memoria que se encuentra especificada en un determinado registro o en otra posición de memoria
    - Ejemplo: LDR R1, **[R3]**
      - R1 ⇐ (R3)





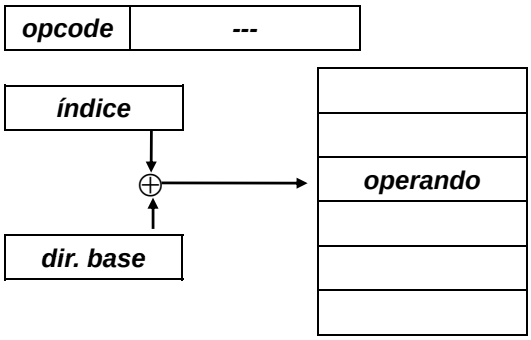
Modos de Direccionamiento

- **Indirecto con Desplazamiento:**
  - La dirección efectiva del operando se obtiene al operar el contenido de un registro (que actúa de dirección base), con una determinada cantidad que viene indicada en los bits siguientes al opcode
  - Ejemplo: LDR R1, **[R3, #45]**
    - R1 ⇐ (R3 + 45)
- **Indirecto con Incremento:**
  - Es un direccionamiento indirecto, pero donde el registro que da la dirección base se modifica:
  - Previamente al cálculo de la dirección efectiva (***pre-incremento***)
  - O posteriormente al cálculo de la dirección efectiva (***post-incremento***)
    - LDR R1, **[R3], #4** // R1 ⇐ (R3); R3 ⇐ R3 +4
- **Indirecto con Decremento:**
  - Análogo al de incremento. Existe ***pre-decremento*** y ***post-decremento***
    - LDR R1, **[R3],# -4** // R1 ⇐ (R3); R3 ⇐ R3 -4

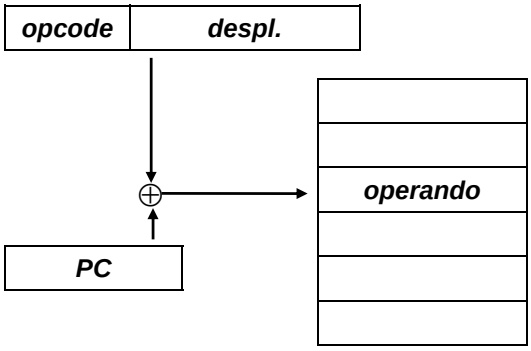


Modos de Direccionamiento

- **Indexado:**
  - Idéntico al indirecto con desplazamiento, pero en el que la cantidad que se le suma a la dirección base, está en otro registro
  - Ejemplo: LDR R1, **[R3, R8]**
    - R1 ⇐ (R3 + R8)
  - Puede presentarse como variante el modo "indexado con desplazamiento"



- **Relativo a PC:**
  - Se trata de un direccionamiento indirecto o indexado, donde la dirección base se encuentra en el PC
  - Tradicionalmente se trata de la versión "indirecto con desplazamiento"
  - Es el modo de direccionamiento utilizado por determinadas llamadas a subrutinas locales
  - Ejemplo: LDR R1, **[R15, #300]**
    - R1 ⇐ (PC + 300)



Modos de Direccionamiento del Cortex-M3

- La documentación del ARM7 suele hacer una clasificación de sus modos de direccionamiento, atendiendo a su momento de uso:
  - **Grupo 1: Para especifica el segundo operando en las instrucciones de procesamiento de datos**
    - Se referirá a dicho operando con <op2>
  - En las operaciones de transferencias de datos a/desde memoria, el operando que no es un registro se puede dar de las siguientes formas:
    - **Grupo 2 <am2>: En operaciones de tamaño *word* o *unsigned byte***
    - Grupo 3 <am3>: En operaciones de tamaño *halfword* o *signed byte*
    - Grupo 4 <am4>: En operaciones de transferencia múltiple
      - Se verán sólo ejemplos muy concretos (para gestión de pila en subrutinas y RAIs)
    - Grupo 5 <am5>: En operaciones relativas a co-procesadores
      - No se cubrirán en este curso

Modos de Direccionamiento del Cortex-M3

- Grupo 1 (<op2>)
  - Se toma como ejemplo la instrucción ADD Rd, Rn, <op2>
  - El operando puede ser una constante o un registro con un desplazamiento opcional
  - Inmediato (ADD Rd, Rn, #inmediato)
    - El operando inmediato tiene que ser un número de 8 bits, o ese número desplazado a la izquierda un número par de bits (entre 0 y 30)
      - Por lo tanto, todos los números de 32 bits no son válidos
  - Directo a Registro (ADD Rd, Rn, Rm)
  - Directo a Registro, con el contenido del registro desplazado
    - El desplazamiento puede ser:
      - ASR – desplazamiento aritmético a derechas
      - LSL – desplazamiento lógico a izquierdas
      - LSR – desplazamiento lógico a derechas
      - ROR – rotación a derechas
      - RRX – rotación a derechas de un único bit, introduciendo por la izquierda C
    - El número de bits a desplazar puede ser (salvo para RRX):
      - Un número inmediato (ADD Rd, Rn, Rm, LSL #inmediato)
      - El contenido de un registro (ADD Rd, Rn, Rm, LSL Rs)
    - Si S=1, entonces el flag C se actualiza con el valor del salida del registro de desplazamiento

Modos de Direccionamiento del Cortex-M3

- Grupo 2 (<am2>)
  - Se toma como ejemplo la instrucción LDR Rd, <am2>
  - Indirecto con Desplazamiento (LDR Rd, [Rn, #+/-<offset_12>])
    - La dirección efectiva se obtiene sumándole o restándole a la dirección base (dada por Rn), el desplazamiento (positivo o negativo) dado de forma inmediata en 12 bits sin signo
      - Rd ⬅ (Rn + <offset_12>) o también Rd ⬅ (Rn - <offset_12>)
  - Indexado (LDR Rd, [Rn, +/-Rm])
    - La dirección efectiva se obtiene sumándole o restándole a la dirección base (dada por Rn) el contenido del registro índice (Rm)
      - Rd ⬅ (Rn + Rm) o también Rd ⬅ (Rn - Rm)
  - Indexado con escalado por desplazamiento (LDR Rd, [Rn, +/-Rm ASR #inmediato]
    - Al igual que en Modo 1, los desplazamientos pueden ser ASR, LSL, LSR, ROR o RRX
      - Rd ⬅ (Rn + Shift(Rm)) o también Rd ⬅ (Rn – Shift(Rm))
  - Pre-incremento / Pre-decremento (LDR Rd, [Rn, #offset]!)
    - Ídem a los anteriores, pero actualizando el contenido de la dirección base
  - Post-incremento / Post-decremento (LDR Rd, [Rn], #offset)
    - Ídem a los anteriores, pero se utilizando Rn a posteriori (dir. efect. = Rn)