Arquitectura de Microcontroladores

Carrera de Especialización en Sistemas Embebidos

Clase 3: Instruction Set Architecture (ISA)

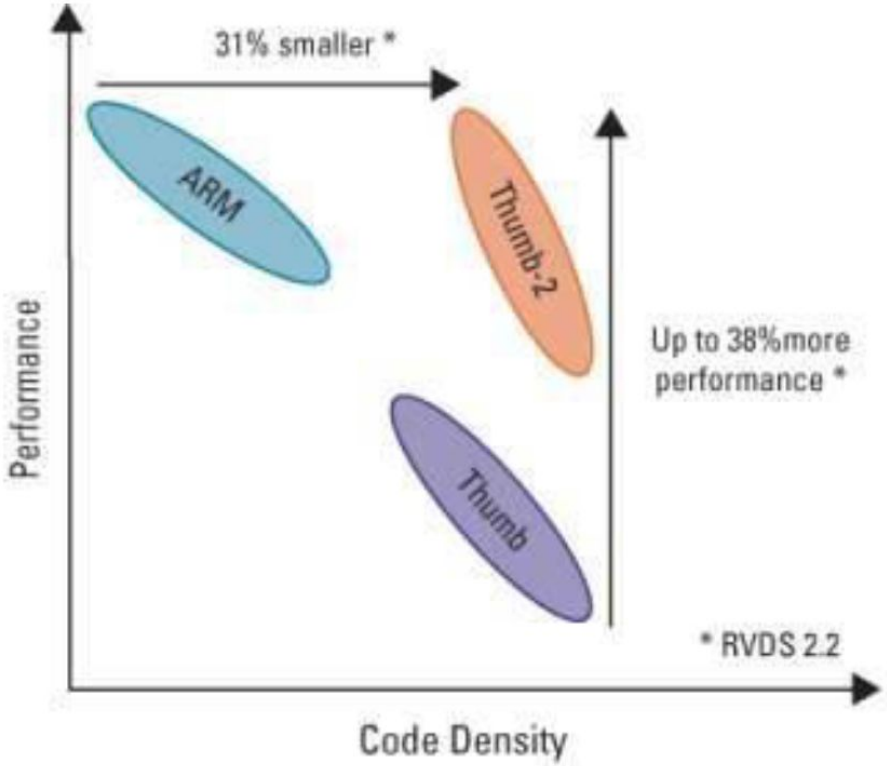
Contenidos de la clase

1. Características generales
2. Tipos de instrucciones
3. Aritmética saturada
4. Interfaz entre C y Assembly

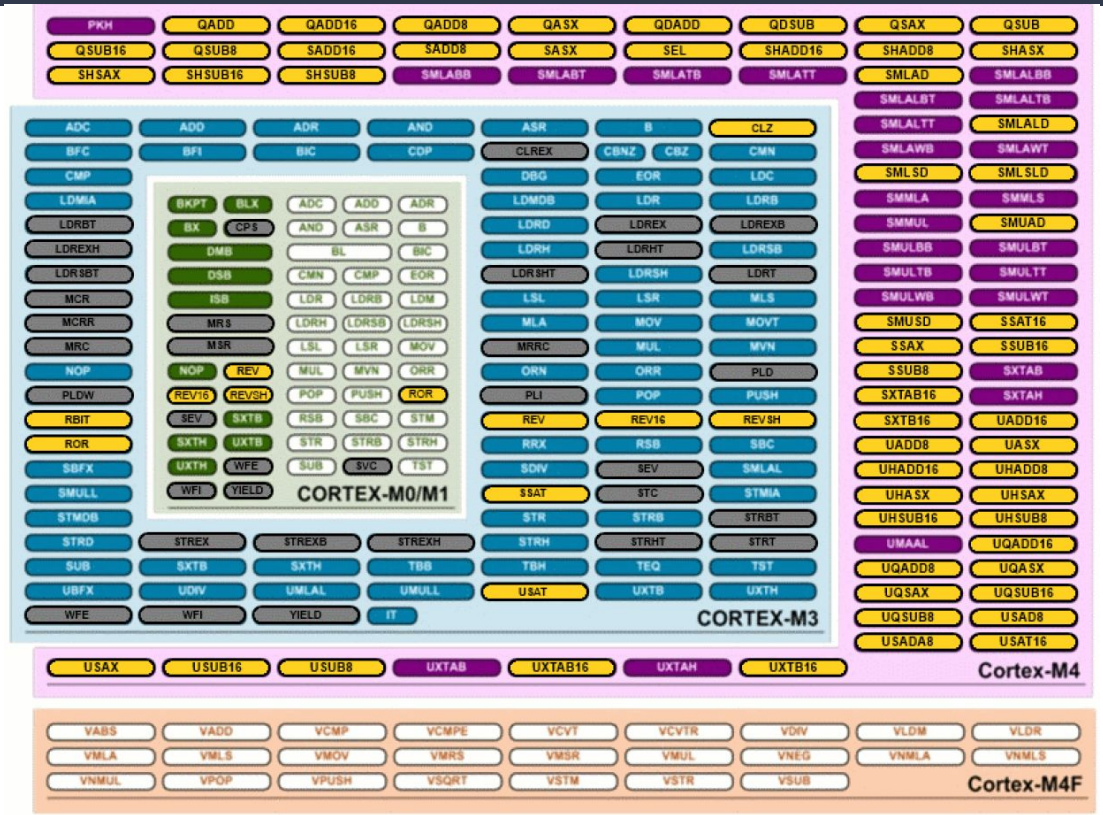
1. Características generales

Al set de instrucciones de la arquitectura ARMv7 se lo conoce como Thumb-2 y es el resultado de la “combinación” entre otros 2 set de instrucciones.

Thumb-2 fue desarrollado por ARM para que sea compatible con “Unify Assembly Language (UAL), aporta mayor compatibilidad con arquitecturas antiguas: Link UAL



1. Características generales



2. Tipos de instrucciones

Las instrucciones se pueden agrupar según su propósito en:

- Movimiento de datos
- Acceso a memoria (lectura/escritura)
- Operaciones aritméticas
- Operaciones lógicas
- Control de flujo de programa
- Otras que veremos más adelante…

2. Tipos de instrucciones

Movimiento de datos

La instrucción “raíz” es MOV, sin embargo, se le pueden especificar algunas cuestiones…

Table 5.4 Instructions for Transferring Data within the Processor

Instruction	Dest	Source	Operations
MOV	R4,	R0	; Copy value from R0 to R4
MOVS	R4,	R0	; Copy value from R0 to R4 with APSR (flags) update
MSR	CONTROL,	R2	; Copy value of R2 into CONTROL (special register)
MOV	R3,	#0x34	; Set R3 value to 0x34
MOVS	R3,	#0x34	; Set R3 value to 0x34 with APSR update
MOVW	R6,	#0x1234	; Set R6 to a 16-bit constant 0x1234
MOVT	R6,	#0x8765	; Set the upper 16-bit of R6 to 0x8765
MVN	R3,	R7	; Move negative value of R7 into R3

2. Tipos de instrucciones

Acceso a memoria

Se permite cargar distintos tipos (tamaños) de datos en los registros de propósito general (r0 - r12).

Esto será de utilidad más adelante…

Table 5.6 Memory Access Instructions for Various Data Sizes

Data Type	Load (Read from Memory)	Store (Write to Memory)
8-bit unsigned	LDRB	STRB
8-bit signed	LDRSB	STRB
16-bit unsigned	LDRH	STRH
16-bit signed	LDRSH	STRH
32-bit	LDR	STR
Multiple 32-bit	LDM	STM
Double-word (64-bit)	LDRD	STRD
Stack operations (32-bit)	POP	PUSH

2. Tipos de instrucciones

Acceso a memoria: cuestiones a tener en cuenta

```
ldr r0, [r1]      // r0 = *r1

ldr r0, [r1,64]   // r0 = *(r1+64)

ldr r0, [r1,4]!   // r0 = *(r1+4) y r1 += 4

ldr r5, [r1, r2, LSL 2] // r5 = *(r1 + (r2 << 2))

str r0, [r1], -4 // *r1 = r0; r1-=4

str r4, [r1, r2, LSL #2]   // *(r1 + (r2 << 2)) = r4
```

2. Tipos de instrucciones

Operaciones aritméticas

Se debe recordar que estas operaciones también pueden afectar al APSR. Para ello es necesario colocar el sufijo “s” al final de la instrucción:

```
subs r3, #2

adds r0, r4
```

Table 5.22 Instructions for Arithmetic Data Operations

Commonly Used Arithmetic Instructions (optional suffixes not shown)	Operation
`ADD Rd, Rn, Rm ; Rd = Rn + Rm`	ADD operation
`ADD Rd, Rn, # immed ; Rd = Rn + #immed`	
`ADC Rd, Rn, Rm ; Rd = Rn + Rm + carry`	ADD with carry
`ADC Rd, #immed ; Rd = Rd + #immed + carry`	
`ADDW Rd, Rn,#immed ; Rd = Rn + #immed`	ADD register with 12-bit immediate value
`SUB Rd, Rn, Rm ; Rd = Rn - Rm`	SUBTRACT
`SUB Rd, #immed ; Rd = Rd - #immed`	
`SUB Rd, Rn,#immed ; Rd = Rn - #immed`	
`SBC Rd, Rn, #immed ; Rd = Rn - #immed - borrow`	SUBTRACT with borrow (not carry)
`SBC Rd, Rn, Rm ; Rd = Rn - Rm - borrow`	
`SUBW Rd, Rn,#immed ; Rd = Rn - #immed`	SUBTRACT register with 12-bit immediate value
`RSB Rd, Rn, #immed ; Rd = #immed - Rn`	Reverse subtract
`RSB Rd, Rn, Rm ; Rd = Rm - Rn`	
`MUL Rd, Rn, Rm ; Rd = Rn * Rm`	Multiply (32-bit result)
`UDIV Rd, Rn, Rm ; Rd = Rn /Rm`	Unsigned and signed divide
`SDIV Rd, Rn, Rm ; Rd = Rn /Rm`	

2. Tipos de instrucciones

Operaciones lógicas

Operaciones bit a bit (bitwise).

Table 5.24 Instructions for Logical Operations

Instruction (optional S suffix not shown)		Operation
AND Rd, Rn	; Rd = Rd & Rn	Bitwise AND
AND Rd, Rn,#immed	; Rd = Rn & #immed	
AND Rd, Rn, Rm	; Rd = Rn & Rm	
ORR Rd, Rn	; Rd = Rd | Rn	Bitwise OR
ORR Rd, Rn,#immed	; Rd = Rn | #immed	
ORR Rd, Rn, Rm	; Rd = Rn | Rm	
BIC Rd, Rn	; Rd = Rd & (~Rn)	Bit clear
BIC Rd, Rn,#immed	; Rd = Rn &(~#immed)	
BIC Rd, Rn, Rm	; Rd = Rn &(~Rm)	
ORN Rd, Rn,#immed	; Rd = Rn | (~#immed)	Bitwise OR NOT
ORN Rd, Rn, Rm	; Rd = Rn | (~Rm)	
EOR Rd, Rn	; Rd = Rd ^ Rn	Bitwise Exclusive OR
EOR Rd, Rn,#immed	; Rd = Rn | #immed	
EOR Rd, Rn, Rm	; Rd = Rn | Rm	

2. Tipos de instrucciones

Operaciones lógicas

Operaciones sobre registros…

¿Por qué no hay una Arithmetic Shift Left (ASL)? ¿Y un ROtate Left (ROL)?

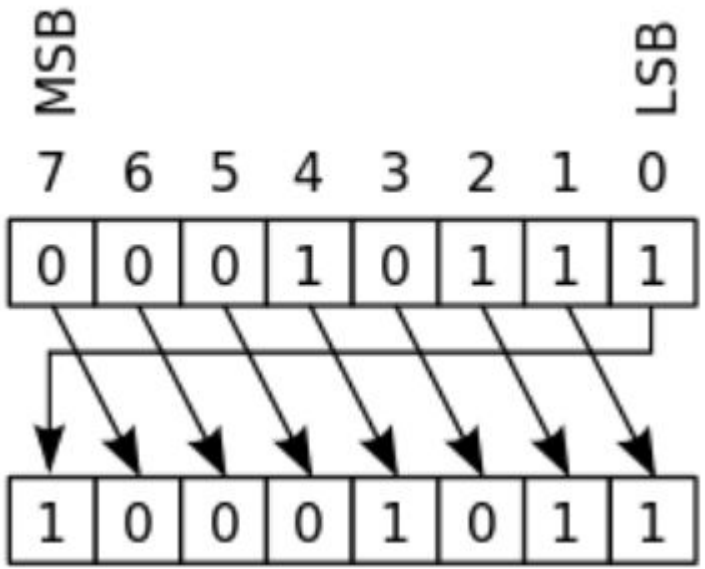
Table 5.25 Instructions for Shift and Rotate Operations

Instruction (optional "S" suffix not shown)			Operation
ASR	Rd, Rn,#immed	; Rd = Rn >> immed	Arithmetic shift right
ASR	Rd, Rn	; Rd = Rd >> Rn	
ASR	Rd, Rn, Rm	; Rd = Rn >> Rm	
LSL	Rd, Rn,#immed	; Rd = Rn << immed	Logical shift left
LSL	Rd, Rn	; Rd = Rd << Rn	
LSL	Rd, Rn, Rm	; Rd = Rn << Rm	
LSR	Rd, Rn,#immed	; Rd = Rn >> immed	Logical shift right
LSR	Rd, Rn	; Rd = Rd >> Rn	
LSR	Rd, Rn, Rm	; Rd = Rn >> Rm	
ROR	Rd, Rn	; Rd rot by Rn	Rotate right
ROR	Rd, Rn, Rm	; Rd = Rn rot by Rm	

2. Tipos de instrucciones

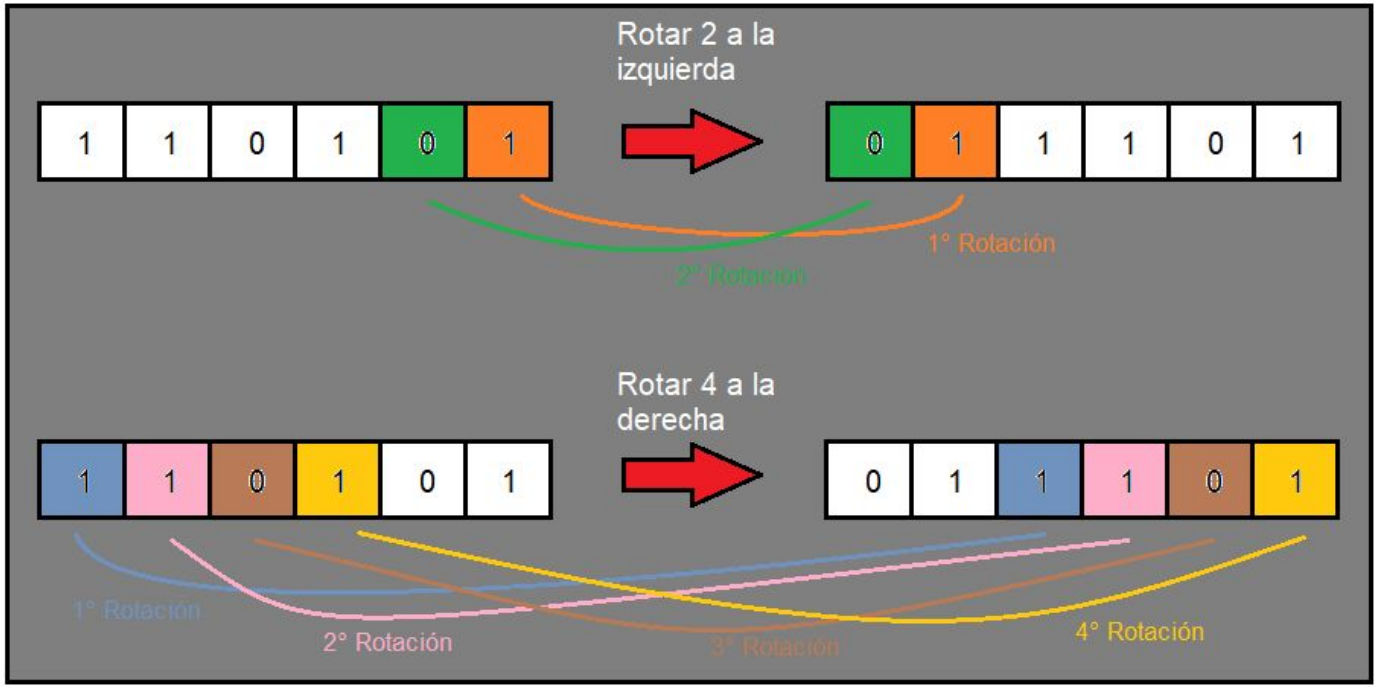
Operaciones lógicas: Rotación de bits a la derecha

No hay rotación a la izquierda por que se puede reemplazar con un desplazamiento a la derecha. Ejemplo en 6 bits: rotar dos a la izquierda, da el mismo resultado que rotar 4 veces a la derecha. Veamos un ejemplo…



2. Tipos de instrucciones

Operaciones lógicas: Rotación de bits a la izquierda vs. a la derecha



2. Tipos de instrucciones

Operaciones lógicas: Rotación de bits a la izquierda vs. a la derecha.

Extracto del libro de Joseph Yiu, página 148 y 149:

> You might wonder why there are rotate right instructions but no instructions for rotate left. Actually, a rotate left operation can be replaced by a rotate right operation
>
> .
>
> .
>
> .
>
> with a different rotate amount. For example, a rotate left by 4 bits can be written as a rotate right by 28 bits. This gives you the same result in the destination register (note that the C flag will be different from rotate left) and takes same amount of time to execute.

2. Tipos de instrucciones

Operaciones lógicas: Shift lógico vs. aritmético

Representación de números con signo de 4 bits →

El shift lógico se basa en agregar ceros en la parte LSB si es LSL o en la parte MSB del dato si es LSR. Ejemplo:

```
LSL de 2 posiciones: 1 → 0001 << 2 = 0100 (4)

LSR de 1 posición: 6 → 0110 >> 1 = 0011 (3)
```

¿Qué sucede con los números negativos?

0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	−1
1110	−2
1101	−3
1100	−4
1011	−5
1010	−6
1001	−7
1000	−8

2. Tipos de instrucciones

Operaciones lógicas: Shift lógico vs. aritmético

Si hacemos un desplazamiento lógico con números con signo, podemos cometer errores:

```
LSL de 2 posiciones: 2 → 0010 << 2 = 1000 (-8)

LSR de 3 posiciones: -1 → 1111 >> 3 = 0001 (1)

LSR de 1 posición: -2 → 1110 >> 1 = 0111 (7)
```

En el caso de los signos positivos podemos tener un “overflow” que se manifiesta como un número negativo. En los negativos perdemos el signo.

0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	−1
1110	−2
1101	−3
1100	−4
1011	−5
1010	−6
1001	−7
1000	−8

2. Tipos de instrucciones

Operaciones lógicas: Shift lógico vs. aritmético

Por este motivo existe el Shift aritmético (ASR):

```
ASR de 1 posición: -8 → 1000 >> 1 = 1100 (-4)
```

Si es a la izquierda el desplazamiento, se puede utilizar el Shift lógico:

```
LSL de 2 posiciones: -1 → 1111 << 2 = 1100 (-4)

LSL de 1 posición: -3 → 1101 << 1 = 1010 (-6)
```

0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	−1
1110	−2
1101	−3
1100	−4
1011	−5
1010	−6
1001	−7
1000	−8

2. Tipos de instrucciones

Control de flujo de programa: Comparaciones*

Table 5.30 Instructions for Compare and Test

Instruction	Operation
CMP <Rn>, <Rm>	Compare: Calculate Rn-Rm. APSR is updated but the result is not stored.
CMP <Rn>, #<immed>	Compare: Calculate Rn – immediate data.
CMN <Rn>, <Rm>	Compare negative: Calculate Rn + Rm. APSR is updated but the result is not stored.
CMN <Rn>, #<immed>	Compare negative: Calculate Rn + immediate data. APSR is updated but the result is not stored.

2. Tipos de instrucciones

Control de flujo de programa:
Saltos condicionales

Table 5.34 Instructions for Conditional Branch

Instruction	Operation
B<cond> <label> B<cond> <label>	Branch to label if condition is true. E.g., CMP R0, #1 BEQ loop ; Branch to "loop" if R0 equal 1.

Table 5.35 Suffixes for Conditional Branches and Conditional Execution

Suffix	Branch Condition	Flags (APSR)
EQ	Equal	Z flag is set
NE	Not equal	Z flag is cleared
CS/HS	Carry set / unsigned higher or same	C flag is set
CC/LO	Carry clear / unsigned lower	C flag is cleared
MI	Minus / negative	N flag is set (minus)
PL	Plus / positive or zero	N flag is cleared
VS	Overflow	V flag is set
VC	No overflow	V flag is cleared
HI	Unsigned higher	C flag is set and Z is cleared
LS	Unsigned lower or same	C flag is cleared or Z is set
GE	Signed greater than or equal	N flag is set and V flag is set, or N flag is cleared and V flag is cleared (N == V)
LT	Signed less than	N flag is set and V flag is cleared, or N flag is cleared and V flag is set (N != V)
GT	Signed greater then	Z flag is cleared, and either both N flag and V flag are set, or both N flag and V flag are cleared (Z == 0 and N == V)
LE	Signed less than or equal	Z flag is set, or either N flag set with V flag cleared, or N flag cleared and V flag set (Z == 1 or N != V)

2. Tipos de instrucciones

Control de flujo de programa: Saltos condicionales → cbz y cbnz. Ejemplo:

```
        mov r0, #5          ; Se setea el contador del bucle
loop1   cbz r0, exit1       ; Si r0 = 0, se salta a exit1
        . . .
        subs r0, #1         ; decremento del contador
        b loop1             ; Salto incondicional
exit1
```

2. Tipos de instrucciones

Control de flujo de programa: Instrucción IT (If - Then)

Las condiciones son las mismas que se vieron para saltos condicionales (Tabla 5.35).

Table 5.36 IT Instruction Block of Various Sizes

	IT block (each of <x>, <y> and <z> can either be T (true) or E (else))	Examples
Only one conditional instruction	`IT <cond>` `instr1<cond>`	`IT EQ` `ADDEQ R0, R0, R1`

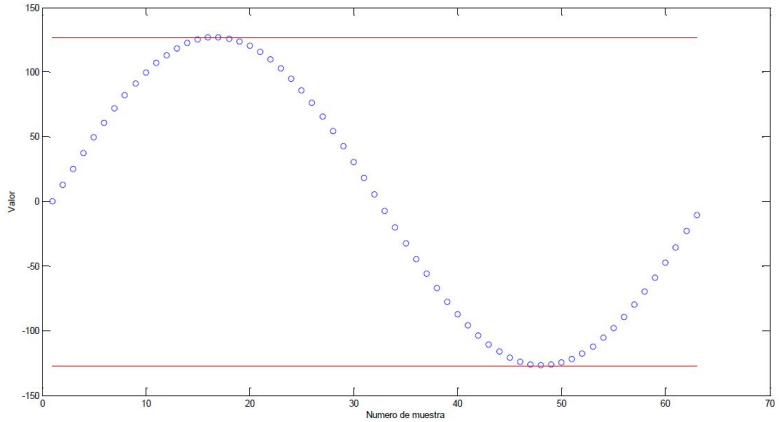
Two conditional instructions	`IT<x> <cond>`
	`instr1<cond>`
	`instr2<cond or ~(cond)>`
		`ITE GE`
		`ADDGE R0, R0, R1`
		`ADDLT R0, R0, R3`

Three conditional instructions	`IT<x><y> <cond>`
	`instr1<cond>`
	`instr2<cond or ~(cond)>`
	`instr3<cond or ~(cond)>`
		`ITET GT`
		`ADDGT R0, R0, R1`
		`ADDLE R0, R0, R3`
		`ADDGT R2, R4, #1`

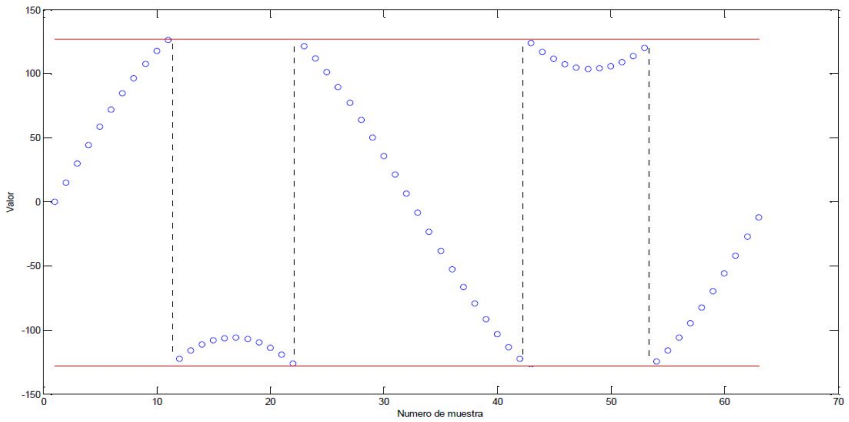
Four conditional instructions	`IT<x><y><z><cond>`
	`instr1<cond>`
	`instr2<cond or ~(cond)>`
	`instr3<cond or ~(cond)>`
	`instr4<cond or ~(cond)>`
		`ITETT NE`
		`ADDNE R0, R0, R1`
		`ADDEQ R0, R0, R3`
		`ADDNE R2, R4, #1`
		`MOVNE R5, R3`

3. Aritmética saturada

Se produce **overflow** y **underflow**



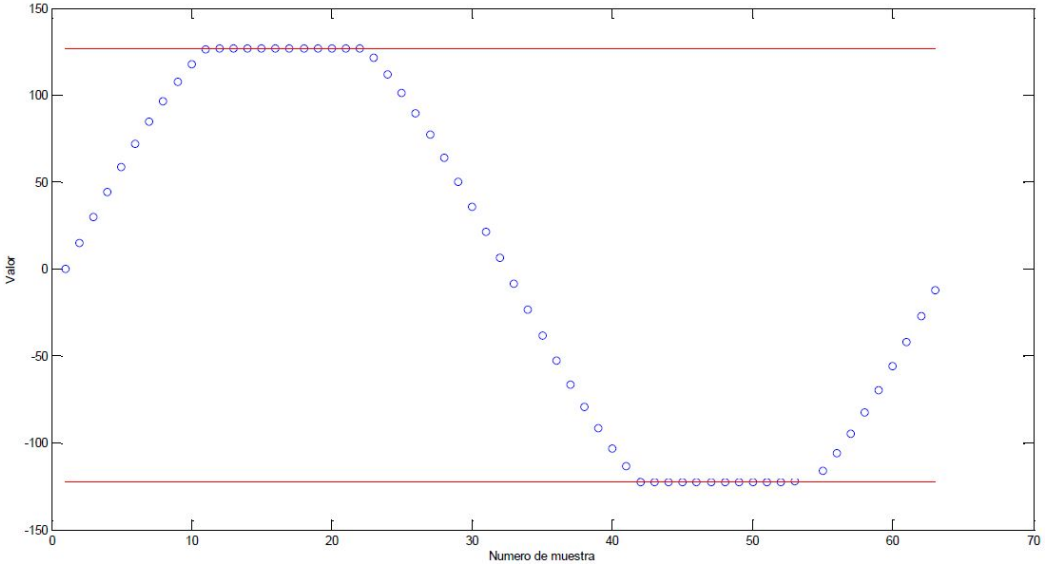
Amplificación



Si tenemos un número determinado de bits para representar una señal, si no se exceden se obtiene una representación digital sin problemas. ¿Qué sucede si nos excedemos del rango?

3. Aritmética saturada

En el campo del procesamiento de señales el overflow no es deseado, por lo cual se prefiere limitar la el valor máximo o mínimo y que la señal mantenga ese valor. De esta forma se obtiene una representación más “natural” de la señal. Instrucciones para esto:




```
ssat r1, #16, r0 ; números con signo

ussat r4, #8, r2 ; números sin signo
```

4. Interfaz entre C y Assembly

La interfaz entre C y Assembly en Cortex está definida en el documento “ARM Architecture Procedure Call Standar”. Conocido también como AAPCS.

Los más destacado de este documento es:

- r0, r1, r2 y r3 son los registros que se utilizan para pasar como parámetros a una función:

  ```
  myFunc (arg0, arg1, arg2, arg3) → r0=arg0; r1=arg1 …
  ```

- r0 se utiliza para devolver el resultado de una función.