FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Arquitectura de Microprocesadores
Introducción a SIMD

Mg. Ing. Facundo Larosa
Lic. Danilo Zecchin
Carrera de Especialización en Sistemas Embebidos

Temario

- Contexto: DSP
- Definiciones SIMD
- Tipos de datos
- ISA
- Ejemplo

DSP (Digital Signal Processing)

El procesamiento digital de señales (DSP, por sus siglas en inglés) es la manipulación matemática de una señal digital para diferentes fines:

- Filtrar la señal, atenuando el ruido fuera de la banda pasante o interferencias
- Detectar si una señal está presente dentro de otra (por ejemplo, en una señal de radar, en el reconocimiento de voz, en ciertas modulaciones, etc.)
- Extraer información de la señal a través de un dominio transformado (por ejemplo, aplicando la FFT)
- ¡Entre muchas otras!

Empecemos con un ejemplo: filtro digital

Si tenemos la señal discreta x[n] y queremos aplicarle un filtro definido por su respuesta al impulso h[n], bastará con aplicar la operación de convolución discreta:

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k]$$

Donde **y[n] es la señal filtrada**.

Vemos que para aplicar el filtro, bastará con realizar una **sumatoria de productos**. Así, un procesador que provea soporte para realizar esta operación deberá procesar eficientemente una **sumatoria de productos**.

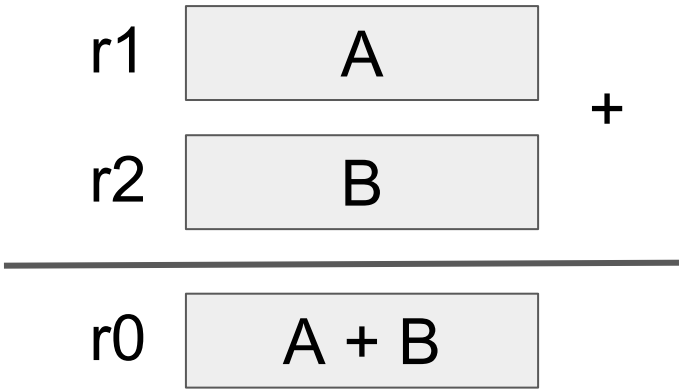
SIMD - Introducción

- SIMD = “Single Instruction **Multiple** Data”
- Una instrucción SIMD procesa múltiples datos en una única operación
- Cortex M4, a diferencia de M3, incluye soporte para instrucciones SIMD.
- Complementan a las instrucciones de aritmética saturada para el procesamiento DSP

Instrucciones escalares

- Una instrucción de suma ordinaria (no SIMD) realiza la suma de dos escalares:

add r0,r1,r2 ≡ r0=r1+r2;



Instrucciones SIMD

- Una instrucción de suma múltiple (SIMD) realiza la suma de dos vectores:

sadd16 r0,r1,r2

SIMD - Tipos de datos

Los registros se pueden interpretar como vectores para realizar operaciones múltiples:

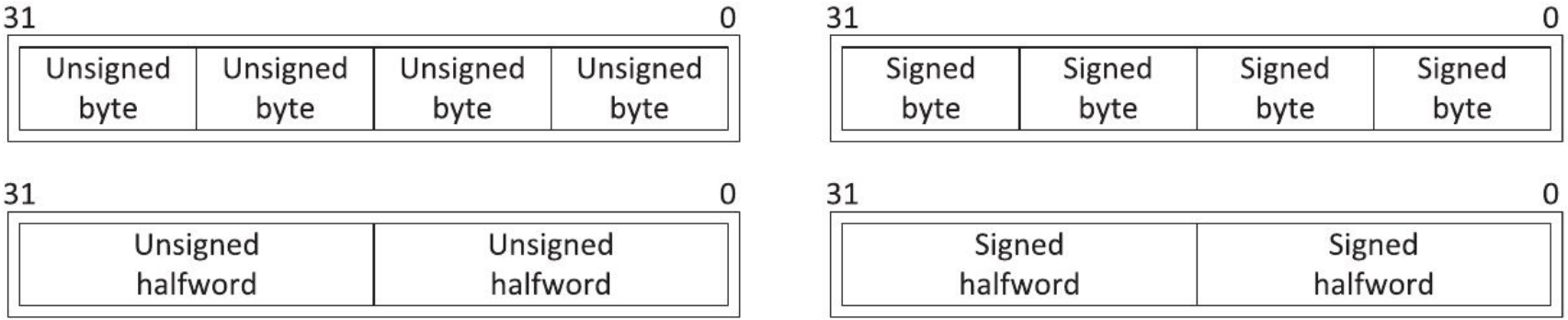


FIGURE 5.13

Various possible SIMD data representations in a 32-bit register

Fuente: YIU, Joseph, “The Definitive Guide to ARM Cortex M3 and M4 processors”

SIMD - ISA

Table 5.49 SIMD Instructions

Prefix Operation (see next table)	S[1] Signed	Q[2] Signed Saturating	SH[3] Signed Halving	U[1] Unsigned	UQ[2] Unsigned Saturating	UH[3] Unsigned Halving
ADD8	SADD8	QADD8	SHADD8	UADD8	UQADD8	UHADD8
SUB8	SSUB8	QSUB8	SHSUB8	USUB8	UQSUB8	UHSUB8
ADD16	SADD16	QADD16	SHADD16	UADD16	UQADD16	UHADD16
SUB16	SSUB16	QSUB16	SHSUB16	USUB16	UQSUB16	UHSUB16
ASX	SASX	QASX	SHASX	UASX	UQASX	UHASX
SAX	SSAX	QSAX	SHSAX	USAX	UQSAX	UHSAX

[1]GE bits updates.
[2]Q bit is set when saturation occurs.
[3]Each data in the SIMD operation result is divided by 2 in Signed Halving (SH) and Unsigned Halving (UH) operations.

Fuente: YIU, Joseph, “The Definitive Guide to ARM Cortex M3 and M4 processors”

SIMD - ISA

Table 5.50 Base Operations for SIMD Instructions

Operation	Descriptions
ADD8	Add 4 pairs of 8-bit data
SUB8	Subtract 4 pairs of 8-bit data
ADD16	Add 2 pairs of 16-bit data
SUB16	Subtract 2 pairs of 16-bit data
ASX	Exchange half-words of the second operand register, then add top half-words and subtract bottom half-word
SAX	Exchange half-words of the second operand register, then subtract top half-words and add bottom half-word

Fuente: YIU, Joseph, “The Definitive Guide to ARM Cortex M3 and M4 processors”

SIMD - ISA

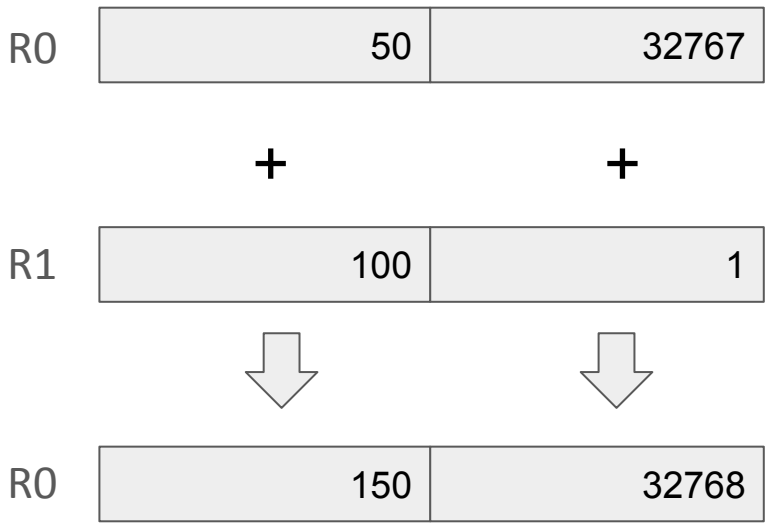
Table 5.51 Additional SIMD Instruction

Operation	Descriptions
USAD8	Unsigned Sum of Absolute Difference between 4 pairs of 8-bit data
USADA8	Unsigned Sum of Absolute Difference between 4 pairs of 8-bit data and Accumulate
USAT16	Unsigned saturate 2 signed 16-bit values to a selected unsigned range
SSAT16	Signed saturate 2 signed 16-bit values to a selected unsigned range
SEL	Select Byte from first or second operand based on GE flags

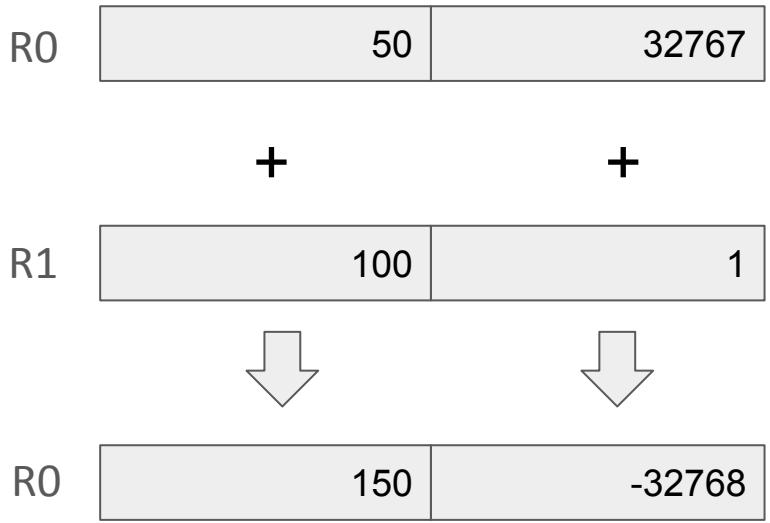
Fuente: YIU, Joseph, “The Definitive Guide to ARM Cortex M3 and M4 processors”

Ejemplo I: xADD16

UADD16 r0, r0, r1

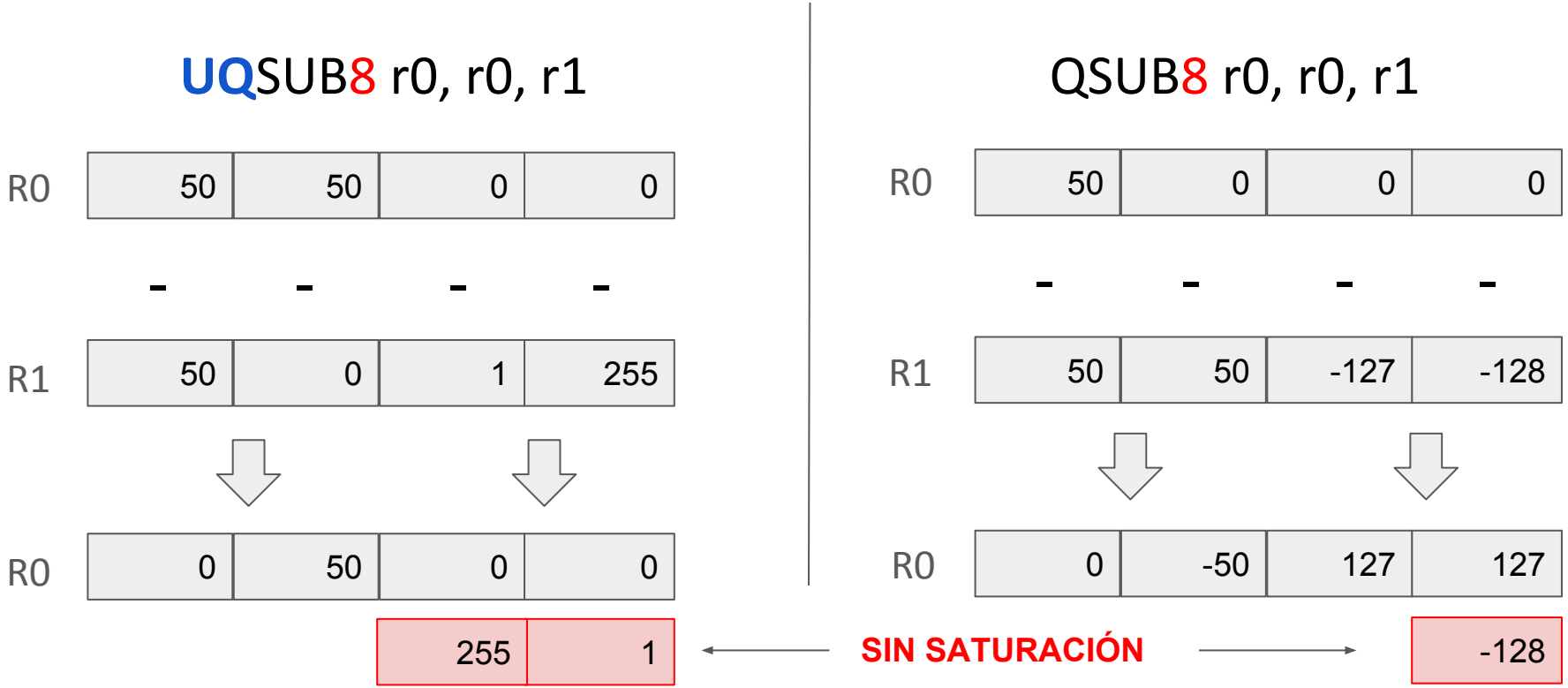


SADD16 r0, r0, r1



Fuente: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0553a/CHDDFBDI.html

Ejemplo II: xQSUB8

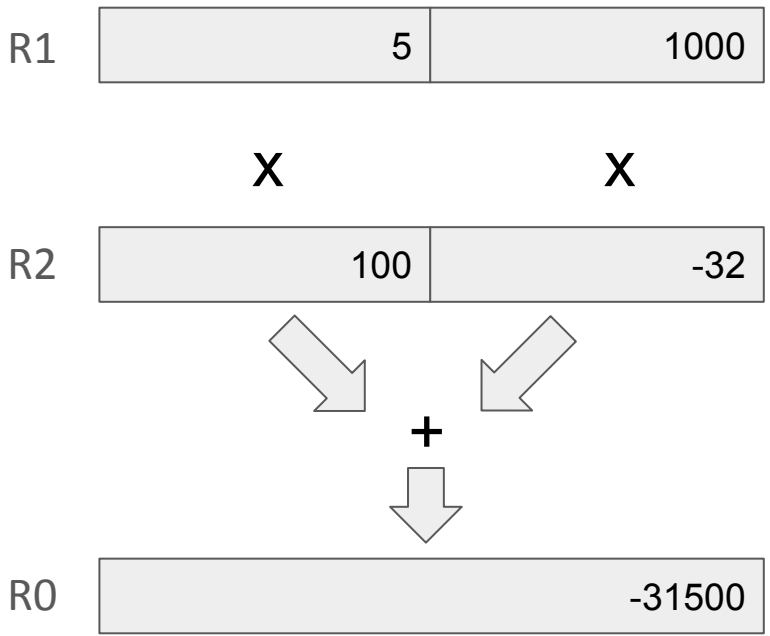


Fuente: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0553a/CHDDFBDI.html

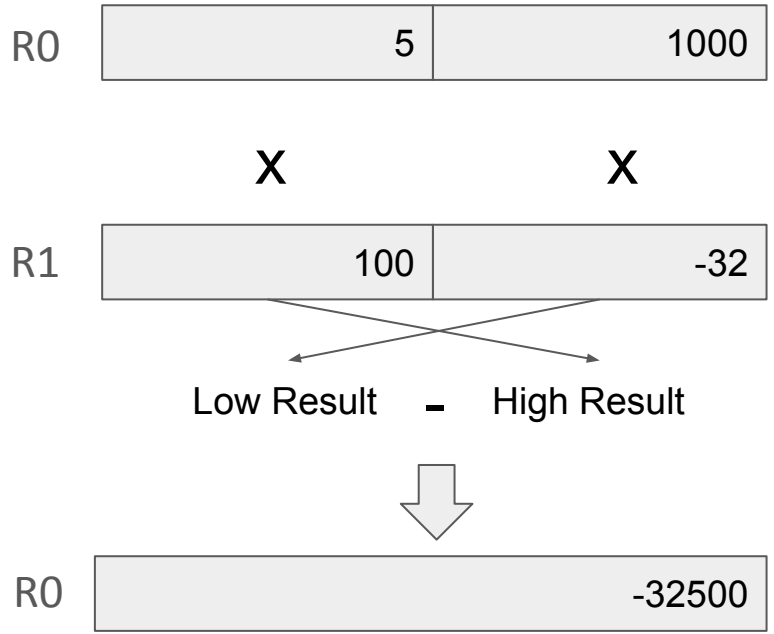
Ejemplo III: SMUAD/SMUSD

Q = 1 si la suma desborda
*La multiplicación **no puede desbordar***

SMUAD R0, R1, R2



SMUSD R0, R1, R2



Fuente: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.dui0553a/CHDDFBDI.html

Hagamos un ejercicio...

Supongamos que tenemos dos señales representadas por dos vectores discretos:

$$X[n] , Y[n]$$

Queremos “combinar” ambas señales para formar una nueva señal Z[n]:

$$Z[n]=½ (X[n]+Y[n])$$

Vamos a implementar el algoritmo de diferentes maneras, para luego comparar la performance:

- En C
- En *assembler*, con instrucciones ordinarias
- En *assembler*, con instrucciones SIMD

Función en C

```
void fusion_c (uint16_t * salida,uint16_t * s1,uint16_t * s2,uint16_t cant)

{

    uint32_t i;

    for(i=0;i<cant;i++)

        *(salida+i)=(*(s1+i)+*(s2+i))/2;  //salida[i] = ( s1[i] + s2[i] ) /
2;

}
```

/* **s1:** Es el puntero a la señal (vector) 1 : X

s2: Es el puntero a la señal (vector) 2 : Y

salida: Es el puntero a la señal de salida combinada : Z

cant: Es la longitud de las señales (vectores)*/

Función en assembler con instrucciones ordinarias

```
#define salida r0
#define s1 r1
#define s2 r2
#define cant r3

Fusion_asm:
    push {r4-r5,lr}
    sub cant,1
Lazo:
    ldrh r4,[s1,cant,LSL 1]
    ldrh r5,[s2,cant,LSL 1]
    add r5,r4
    asr r5,1     //Shift 1 bit a derecha
    strh r5,[salida,cant,LSL 1]
    subs cant,1
    bpl Lazo     //Saltamos si es positivo o cero
    pop {r4-r5,pc}
```

Función en assembler con instrucciones SIMD

```
#define salida r0
#define s1 r1
#define s2 r2
#define cant r3

fusion_simd:

    push {r4-r5,lr}
    asr cant,1
    sub cant,1
Lazo:
    ldr r4,[s1,cant,LSL 2]
    ldr r5,[s2,cant,LSL 2]
    uhadd16 r5,r5,r4
    str r5,[salida,cant,LSL 2]
    subs cant,1
    bpl Lazo     //Saltamos si es positivo o cero
    pop {r4-r5,pc}
```

Comparación

```
#define salida r0
#define s1 r1
#define s2 r2
#define cant r3

Fusion_asm:
    push {r4-r5,lr}
    sub cant,1
Lazo:
    ldrh r4,[s1,cant,LSL 1]
    ldrh r5,[s2,cant,LSL 1]
    add r5,r4
    asr r5,1
    strh r5,[salida,cant,LSL 1]
    subs cant,1
    bpl Lazo
    pop {r4-r5,pc}
```

```
#define salida r0
#define s1 r1
#define s2 r2
#define cant r3

fusion_simd:

    push {r4-r5,lr}
    asr cant,1
    sub cant,1
Lazo:
    ldr r4,[s1,cant,LSL 2]
    ldr r5,[s2,cant,LSL 2]
    uhadd16 r5,r5,r4
    str r5,[salida,cant,LSL 2]
    subs cant,1
    bpl Lazo
    pop {r4-r5,pc}
```