

# Laboratorio ARM

Organización del Computador

# Parte I

# Temas

Utilizar operaciones aritméticas y lógicas

Imprimir cadenas de caracteres

Abrir y leer números enteros de archivos

Llamar y codificar subrutinas

# Código en ARM

# Código en ARM

## Original

```
x = 5;
```

```
y = 7;
```

```
z = x + y;
```

## ARM

```
mov    r0,    #5
```

```
mov    r1,    #7
```

```
add    r2,    r0,    r1
```

# Código en ARM

## Original

x = 5;

y = 7;

z = x + y;

## ARM

**mov**    **r0**,    **#5**

mov    r1,    #7

add    r2,    r0,    r1

**move:** copia el valor a un registro

**r0:** registro 0

# Código en ARM

## Original

x = 5;

y = 7;

z = x + y;

## ARM

mov r0, #5

**mov r1, #7**

add r2, r0, r1

**move:** copia el valor a un registro

**r1:** registro 1

# Código en ARM

## Original

x = 5;

y = 7;

z = x + y;

## ARM

mov r0, #5

mov r1, #7

**add r2, r0, r1**

**add:** suma los registros de la derecha y copia el resultado en el primer registro

**r2:** registro 2



# Interrupciones de Software

# Interrupciones de Software

Necesitamos una forma de leer información y mostrar resultados

¿Qué implica esto?

Implica hablar con los dispositivos que manejan el sistema operativo

Necesitamos una manera de decirle al sistema operativo que tome el control

# Interrupciones de Software

¿Cómo podríamos imprimir algo?

ARM cuenta con una instrucción *swi* que provoca una interrupción de software

# Swi

Tenemos la atención del SO pero ¿cómo sabe lo que queremos?

# SWi

Tenemos la atención del SO pero ¿cómo sabe lo que queremos?

*Operando swi*: recibe un entero que dice qué hacer

El SO también puede leer los registros para obtener información adicional si es necesario

- "Entero que dice qué hacer": por ejemplo, "5" significa "imprimir algo"
- Con la lectura de los registros, estos podrían incluir exactamente qué imprimir

# Imprimiendo un entero

El operando que indica “imprimir un entero” es 0x6B

El registro r1 contiene el entero a imprimir

El registro r0 contiene dónde imprimirlo

l significa “imprimir en stdout (pantalla)”

# Imprimiendo un entero

```
mov    r0, #5
mov    r1, #7
add    r2, r0, r1

mov    r1, r2          ; r1: entero a imprimir
mov    r0, #1          ; r0: dónde imprimir
swi    0x6B           ; 0x6B: imprimir entero
```

# Finalizar programa

Necesitamos indicar el fin del programa

¿Cómo puede hacerse?



# Finalizar programa

Necesitamos indicar el fin del programa

¿Cómo puede hacerse?

*swi* con un operando particular (0x11)

# Finalizar programa

```
mov    r0, #5
mov    r1, #7
add    r2, r0, r1

mov    r1, r2          ; r1: entero a imprimir
mov    r0, #1          ; r0: dónde imprimir
swi    0x6B            ; 0x6B: imprimir entero
swi    0x11            ; 0x11: salir del programa
```

# Secciones del programa

# Secciones del programa

Todo es solo un montón de bits

Necesitamos decirle al ensamblador qué bits  
deben colocarse en qué parte de la memoria

# Secciones del programa

La directiva `.text` especifica la sección de código

```
.text
mov r0, #5
mov r1, #7
add r2, r0, r1

mov r1, r2
mov r0, #1
swi 0x6B

swi 0x11
```

# Secciones del programa

La directiva `.data` especifica la sección de variables

```
.data
string1:
    .asciz "hola"
string2:
    .asciz "chau"
```

ARMSim#

# ARMSim#

*ARMSim-UserGuide* explica como hacer uso de ARMSim#

9557 - *Apunte ARM* contiene los temas que vamos a investigar durante el laboratorio

En ambos documentos se detallan las diferentes funcionalidades que provee la `swi`.



# ARMSim#

## *Estructura básica de un programa*

```
.data
cadena:
    .asciz "linea"
entero:
    .word 78
    .text
    .global _start
_start:
    @ Comentario
    swi 0x11
    .end
```

# ARMSim#

*Salida standard de cadena de caracteres*

```
.data
cadena:
    .asciz "Soy una cadena"
    .text
    .global _start
_start:
    ldr r0, =cadena
    swi 0x02
```

# Práctica I

Hola Mundo

# Stack + Subrutinas

# Stack + Subrutinas

Necesitamos almacenar el estado del procesador cuando hacemos llamadas anidadas

Las instrucciones de transferencia de datos múltiples proporcionan un mecanismo para almacenar el estado en la pila (señalado por R13 o SP)

# Stack + Subroutines

## Load Multiple

```
LDM{addr_mode}{cond} Rn{!}, reglist{^}
```

## Store Multiple

```
STM{addr_mode}{cond} Rn{!}, reglist{^}
```

# Stack + Subrutinas

Los modos de instrucciones de STM y LDM tienen alias para acceder a la pila:

- FD = Full Descending
  - STMFD/LDMFD = STMDB/LDMIA
- ED = Empty Descending
  - STMED/LDMED = STMDB/LDMIB
- FA = Full Ascending
  - STMFA/LDMFA = STMIB/LDMDB
- EA = Empty Ascending
  - STMEA/LDMEA = STMIA/LDMDB

# Stack + Subrutinas

Los modos de instrucciones de STM y LDM tienen alias para acceder a la pila:

- FD = Full Descending
  - STMFD/LDMFD = STMDB/LDMIA

- ED = Empty Descending

- STMED/LDMED = STMDB/LDMIA

- FA = Full Ascending
- Se suele utilizar el modo Full Descending

- STMFA/LDMFA = STMIB/LDMIA

- EA = Empty Ascending

- STMEA/LDMEA = STMIA/LDMDB



# Stack + Subrutinas

## *Llamado a subrutina*

```
bl print_r1_int  
...
```

## *Subrutina*

```
print_r1_int:  
    stmfd sp!, {r0,lr} @ Stack R0  
    ldr r0, #0x1  
    swi SWI_Print_Int   @ Mostrar entero en R1 por consola  
    ldr r1, =EOL  
    swi SWI_Print_Str   @ Mostrar EOL por consola  
    ldmfd sp!, {r0,pc} @ Unstack r0
```

# Práctica 2

Mostrar strings

# Instrucciones Aritméticas

# Instrucciones Aritméticas

## **Add**

`ADD{cond}{S} Rd, Rn, <Oprnd2>`

## **Subtract**

`SUB{cond}{S} Rd, Rn, <Oprnd2>`

## **Reverse subtract**

`RSB{cond}{S} Rd, Rn, <Oprnd2>`

## **Multiply**

`MUL{cond}{S} Rd, Rm, Rs`

# Instrucciones Lógicas

# Instrucciones Lógicas

## **And**

AND{cond}{S} Rd, Rn, <Oprnd2>

## **Exclusive Or**

EOR{cond}{S} Rd, Rn, <Oprnd2>

## **Or**

ORR{cond}{S} Rd, Rn, <Oprnd2>

# Shifts

# Shifts

## **Logical Shift Left**

```
MOV R0, R1, LSL #1
```

## **Logical Shift Right**

```
MOV R0, R1, LSR #1
```

## **Arithmetic Shift Right**

```
MOV R0, R1, ASR #1
```



# Barrel Shifter

Cuando se especifica que el segundo operando es un registro *shifteado*, la operación del Barrel Shifter es controlada por el campo Shift en la instrucción.

Este campo indica el tipo de *shift* a realizar.

La cantidad de bits a *shiftear* puede estar contenida en un campo inmediato o en el byte inferior de otro registro (que no sea el R15)

Debugging con ARMSim#

# Debugging con ARMSim#

- Visibilidad de los Registros
- Visibilidad del contenido de la memoria
- Ejecución paso a paso

# Demo

Debugging con ARMSim#

# Práctica 3

Cálculos aritméticos  
y lógicos

# Archivos

# Archivos

## *Definiciones*

```
.data
filename:
    .asciz "archivo.txt"
    .align
InFileHandle:
    .word 0
```

# Archivos

## *Apertura*

```
ldr r0,=filename      @ (R0) -> nombre de archivo
mov r1,#0              @ (R1) -> modo: entrada
swi 0x66
bcs InFileError
ldr r1,=InFileHandle  @ (R1) -> InFileHandle
str r0,[r1]            @ almacena handler
```



# Archivos

*Leer entero desde archivo*

```
ldr r0,=InFileHandle    @ (R0) -> InFileHandle
ldr r0,[r0]              @ (R0) = InFileHandle
swi 0x6C

                           @ (R0) = Entero leído
```

# Archivos

## *Cerrar archivo*

```
ldr r0,=InFileHandle    @ (R0) -> InFileHandle  
ldr r0,[r0]              @ (R0) = InFileHandle  
swi 0x68
```

# ARMSim#

*Salida standard de entero*

```
mov r0,#1 @ (R0) = Stdout (salida por pantalla)
mov r1,r2 @ (R1) = entero a mostrar
swi 0x6B
```

# Práctica 4

E/S de números  
enteros

# Práctica 5

Negar enteros desde  
archivo

# Revisión de Temas

Utilizar operaciones aritméticas y lógicas

Imprimir cadenas de caracteres

Abrir y leer números enteros de archivos

Llamar y codificar subrutinas

# Laboratorio ARM

Organización del Computador