

Laboratorio ARM

Organización del Computador

Introducción

¿Por qué ARM?

- La simplicidad de los procesadores los hace ideales para aplicaciones de baja potencia.
- Dominantes dentro del mercado de la electrónica móvil
 - Microprocesadores pequeños, de bajo consumo y bajo costo ideales para teléfonos móviles, tablets o netbooks.

Características Principales

- Arquitectura Load/Store
- Instrucciones de longitud fija (32 bits)
- Formatos de instrucciones de 3 direcciones
 - 2 Registros de operandos
 - 1 Registro de resultado
- Ejecución condicional de TODAS las instrucciones.
- Instrucciones de Load-Store de Registros Múltiples.

Organización de Registros

- R0 a R12 son registros de propósito general (32 bits)
 - Usados por el programador para (casi) cualquier propósito sin restricción
- R13 es el *Stack Pointer* (SP)
- R14 es el *Link Register* (LR)
- R15 es el *Program Counter* (PC)
- El *Current Program Status Register* (CPSR) contiene indicadores condicionales y otros bits de estado

Organización de Registros

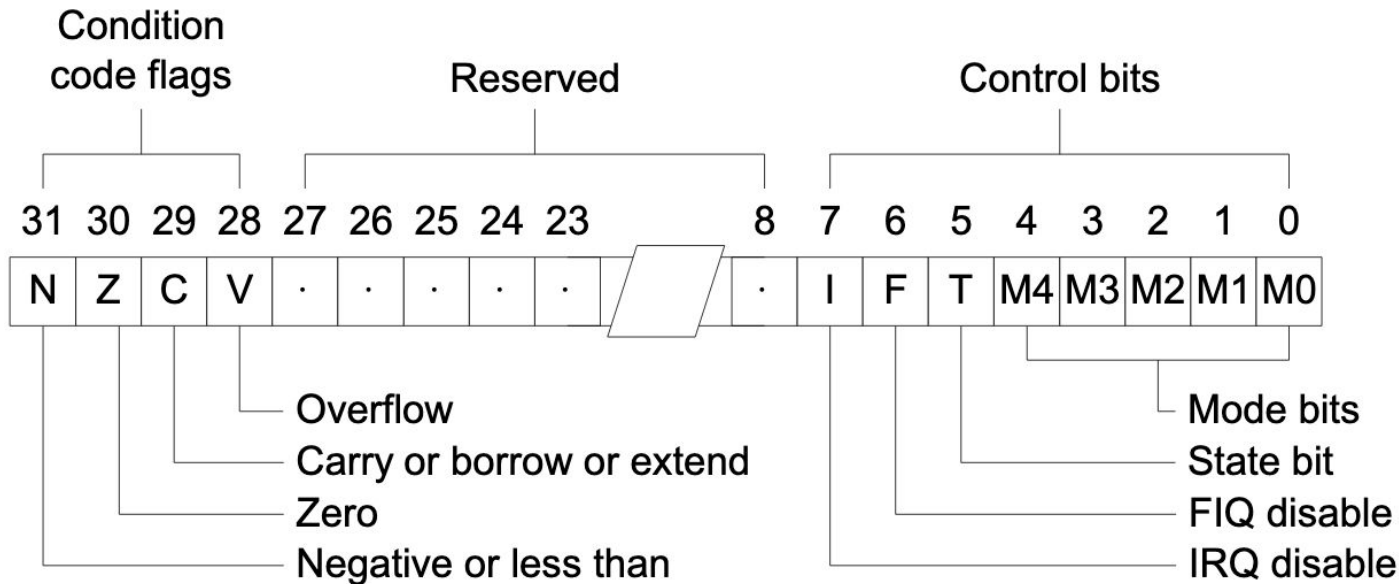
R0	
...	
R12	
R13 (SP)	Stack Pointer
R14 (LR)	Link Register
R15 (PC)	Program Counter

CPSR	Current Program Status Register
-------------	---------------------------------

Organización de Registros

CPSR

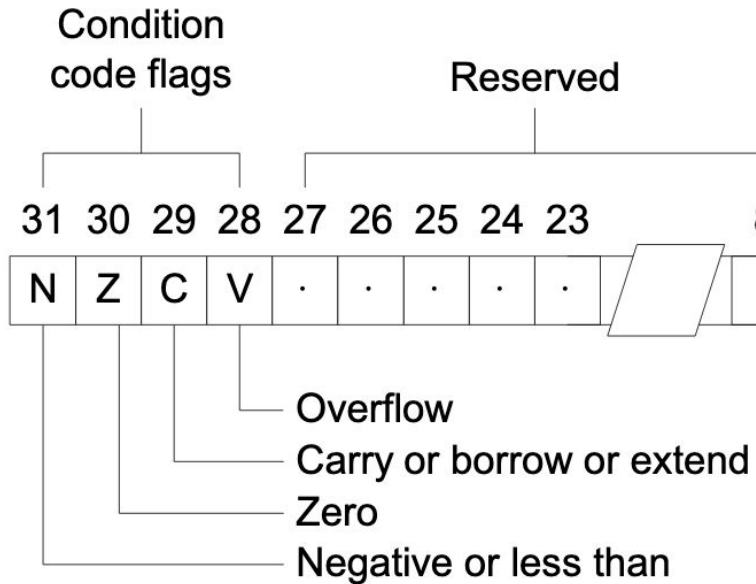
Current Program Status Register



Organización de Registros

CPSR

Current Program Status Register



Flag	Significado
N	Resultado N egativo del ALU
Z	Resultado (Z)ero del ALU
C	Operación ALU con a C arreo hacia afuera
V	Operación ALU con o V erflow

Organización de Memoria

- Máximo: 2^{32} bytes de memoria
- *Word* = 32-bits
- *Half-word* = 16 bits
- *Words* están alineadas en posiciones divisibles por 4
- *Half-words* están alineadas en posiciones pares

Estructura de un Programa

- La forma general de una línea en un módulo ARM es:

```
label <espacio> opcode <espacio> operandos <espacio> @ comentario
```

- Cada campo debe estar separado por uno o más espacios.
- Las instrucciones no empiezan en la primer columna, dado que deben estar precedidas por un espacio en blanco, incluso aunque no haya label.
- ARM acepta líneas en blanco para mejorar la claridad del código.

Código - Ejemplo

```
.text                                @ Indica que los siguientes
                                     @ ítems en memoria son
                                     @ instrucciones

start:                               @ Seteo de parámetros
    mov     r0, #15
    mov     r1, #20
    bl      func                    @ Llamado a subrutina
    swi     0x11                    @ Fin de programa
func:                               @ Subrutina
    add     r0, r0, r1              @ r0 = r0 + r1
    mov     pc, lr                  @ Retornar desde subrutina
    .end                               @ Marcar fin de archivo
```

Código - Ejemplo

.text

@ Indica que los siguientes
@ ítems en memoria son
@ instrucciones

start:

mov r0, #15
mov r1, #20
bl func
swi 0x11

@ Seteo de parámetros

@ Llamado a subrutina

@ Fin de programa

func:

add r0, r0, r1
mov pc, lr
.end

@ Subrutina

@ $r0 = r0 + r1$

@ Retornar desde subrutina

@ Marcar fin de archivo

Labels

Código - Ejemplo

```
.text                                @ Indica que los siguientes
                                     @ ítems en memoria son
                                     @ instrucciones

start:                               @ Seteo de parámetros
    mov    r0, #15
    mov    r1, #20
    bl     func                     @ Llamado a subrutina
    swi    0x11                    @ Fin de programa
func:                               @ Subrutina
    add    r0, r0, r1              @ r0 = r0 + r1
    mov    pc, lr                 @ Retornar desde subrutina
.end                               @ Marcar fin de archivo
```

Op. Code

Código - Ejemplo

```
.text                                @ Indica que los siguientes
                                     @ ítems en memoria son
                                     @ instrucciones

start:                               @ Seteo de parámetros
    mov    r0, #15
    mov    r1, #20
    bl     func                      @ Llamado a subrutina
    swi    0x11                     @ Fin de programa
func:                                   @ Subrutina
    add    r0, r0, r1               @ r0 = r0 + r1
    mov    pc, lr                  @ Retornar desde subrutina
    .end                               @ Marcar fin de archivo
```

Operandos

Código - Ejemplo

```
.text

start:
    mov     r0, #15
    mov     r1, #20
    bl      func
    swi     0x11

func:
    add     r0, r0, r1
    mov     pc, lr
    .end
```

```
@ Indica que los siguientes
@ ítems en memoria son
@ instrucciones
```

```
@ Seteo de parámetros
```

```
@ Llamado a subrutina
```

```
@ Fin de programa
```

```
@ Subrutina
```

```
@ r0 = r0 + r1
```

```
@ Retornar desde subrutina
```

```
@ Marcar fin de archivo
```

Comentarios

Código - Ejemplo

- La rutina principal (etiquetada *start*) carga los valores 15 y 20 en los registros *r0* y *r1* respectivamente.

```
start:
    mov     r0, #15           @ Seteo de parámetros
    mov     r1, #20
    bl      func             @ Llamado a subrutina
    swi     0x11             @ Fin de programa
func:
    add     r0, r0, r1        @ r0 = r0 + r1
    mov     pc, lr           @ Retornar desde subrutina
.end                          @ Marcar fin de archivo
```


Código - Ejemplo

- El programa llama a la subrutina *func* haciendo uso de una instrucción *branch with link (bl)*.

```
.text
start:
    mov     r0, #15           @ Indica que los siguientes
    mov     r1, #20           @ ítems en memoria son
    bl      func              @ instrucciones
    swi     0x11              @ Fin de programa
func:
    add     r0, r0, r1        @ Subrutina
    mov     pc, lr            @ r0 = r0 + r1
    .end                    @ Retornar desde subrutina
                             @ Marcar fin de archivo
```

Código - Ejemplo

- La subrutina suma los dos parámetros que recibe y deja el resultado en *r0*.

```
.text
start:
    mov     r0, #15
    mov     r1, #20
    bl      func
    swi     0x11

func:
    add     r0, r0, r1
    mov     pc, lr
.end

@ Indica que los siguientes
@ ítems en memoria son
@ instrucciones

@ Seteo de parámetros

@ Llamado a subrutina
@ Fin de programa

@ Subrutina
@ r0 = r0 + r1
@ Retornar desde subrutina
@ Marcar fin de archivo
```

Código - Ejemplo

- Luego, retorna simplemente restaurando el *Program Counter* (*r15*) a la dirección que fue guardada en el *Link Register* (*r14*).

```
start: .text @ Indica que los siguientes
        @ instrucciones
        mov     r0, #15 @ Seteo de parámetros
        mov     r1, #20
        bl      func @ Llamado a subrutina
        swi     0x11 @ Fin de programa
func:    @ Subrutina
        add     r0, r0, r1 @ r0 = r0 + r1
        mov     pc, lr @ Retornar desde subrutina
        .end @ Marcar fin de archivo
```


Herramientas

Herramientas

- ARMSim#

- Editor de texto

Sugerido: Sublime

ARMSim#

- Aplicación de escritorio para simular la ejecución de programas ARM en un sistema basado en ARM7TDMI.
- Desarrollado por el Departamento de Ciencias de la Computación de la Universidad de Victoria, Canadá.
- De distribución libre para uso académico.

ARMSim# - Instalador

<https://gitlab.com/ramiroberruezo/arm-lab>

ARMSim# - Enlaces de interés

- ARM7TDMI Technical Reference Manual

<https://developer.arm.com/docs/ddi0029/latest/preface>

- Instalación (v2.1)

<http://webhome.cs.uvic.ca/~nigelh/ARMSim-V2.1/index.html>

- Guía de usuario

<https://www.lri.fr/~de/ARM-Tutorial.pdf>

- Primeros pasos

<https://homepage.cs.uiowa.edu/~ghosh/GettingStarted.pdf>

- Página principal

<https://connex.csc.uvic.ca/access/content/group/ARMSim/SIMWeb/index.html>

Dudas

Consultas

Feedback

Oigo y olvido, veo y recuerdo, hago y aprendo.

- Proverbio chino

Laboratorio ARM

Organización del Computador