Técnicas Digitales II

Lenguaje Máquina

Introducción

- El lenguaje Ensamblador, fue pensado para que sea mas conveniente a la lectura humana, representando con nemónico las diferentes instrucciones que poseía un microprocesador.
- Cada una de estas instrucciones escritas en ensamblador, debe ser trasladada a una serie de 1 y 0 entendible por los circuitos digitales llamado lenguaje máquina.
- El set de instrucciones de ARM es de 32 bits, esto simplifica el diseño al codificar todas las instrucciones en 1 word.
- Simplificaría aún mas si todas estuvieran echas con el mismo formato, ARM no lleva a ese extremo pero si lo minimiza.

Introducción

- ARM posee tres formatos principales de instrucciones.
 - Procesamiento de Dato.
 - Memoria.
 - Bifurcaciones o Saltos.
- Este pequeño conjunto de formatos, permite:
 - Mantener simplicidad en el diseño por la regularidad de las instrucciones de cada familia.
 - Satisfacer las necesidades de cada conjunto de instrucciones.

Introducción

- Las diferencias entre los conjuntos de formatos, son tan importante como para separarlos entre si
 - Las Instrucciones de procesamiento de Datos tiene un primer registro fuente, una segunda fuente que puede ser inmediato o un registro con un desplazamiento y finalmente un registro destino.
 - Las instrucciones de Memoria, poseen tres operandos, un registro base, un corrimiento que puede ser un inmediato o un registro con un desplazamiento opcional y finalmente un registro fuente o destino dependiendo del tipo de instrucción.
 - Las de Salto es básicamente uno pocos bits de configuración unto con 24 bits de un inmediato como desplazamiento.

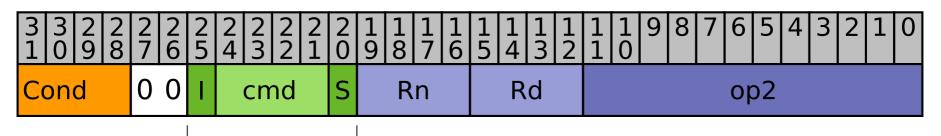
Procesamiento de Datos

- Son las mas comunes, sus campos son los siguientes:
- Op (operation code código de operación) para el caso este formato corresponde a 00
- Func, código de función, describe la función.
- Cond, condicional
- Los operadores, formados por los últimos 3 campos.

3 3 2 2 1 0 9 8	2 2 7 6	2 2 2 2 2 5 4 3 2 1	20	1 1 9 8	17	16	15	14	13	1 2	1	10	9	8	7	6	5	4	3	2	1	0
Cond	Ор	Func		F	Rn			R	d							ok	o2					

Procesamiento de Datos

- El campo func, se divide en tres partes
- I, indica 1 si el op2 es un inmediato o 0 en caso de registro
- **S** indica el bit de banderas, si está en 1 indica que la instrucción afecta a las banderas, caso contrario esta modificación es bloqueada.
- cmd indica la instrucción de procesamiento de datos a realizar.



func

Procesamiento de Datos

Código	Ejemplo	
0000	AND Rd,Rn,Src2	And Lógico
0001	EOR Rd,Rn,Src2	Or Lógico
0010	SUB Rd,Rn,Src2	Resta
0011	RSB Rd,Rn,Src2	Resta Reversa
0100	ADD Rd,Rn,Src2	Suma
0101	ADC Rd,Rn,Src2	Suma con acarreo
0110	SBC Rd,Rn,Src2	Resta con acarreo
0111	RSC Rd,Rn,Src2	Resta Rev. c/acarreo
1000	TST Rn,Src2	Activa banderas según
1001	TEQ Rn,Src2	Realiza or exclusiva
1010	CMP Rn,Src2	Realiza Resta y actualiza banderas
1011	CMN Rn,Src2	Realiza una suma entre dos registros
1100	ORR Rd,Rn,Op2	Or Lógca,

3 3 2 2 1 0 9 8	2 2 7 6	2 5	24	23	2 2	2	2	19	18	1 7	16	15	14	13	1 2	1	10	9	8	7	6	5	4	3	2	1	0
Cond	0 0	I		cm	nd		S		R	n			R	d							ok	ວ2					

Procesamiento de Datos - Operador 2

- El Valor Inmediato puede ser tres cosas distintas
- Valor Inmediato, 8 bit de dato y 4 de corrimiento hacia la derecha x 2

ullet

3 3 2 2 1 0 9 8	2 2 7 6	2 5	2 4	2 3	2 2 2 1	2	2 1	18	17	16	15	14	13	1 2	1	10	9	8	7	6	5	4	3	2	1	0
Cond	0 0		C	cm	d		5	R	ln			R	d							op	2					
			-	fur	nc											rc	n†				i	mı	m8	3		

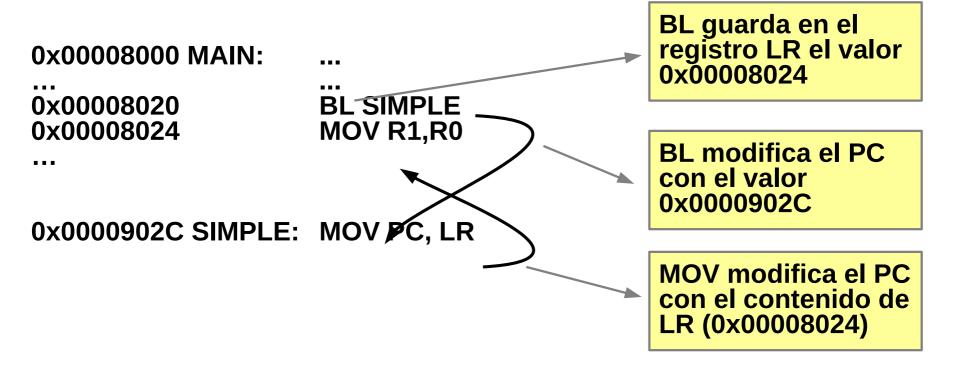
Funciones en ARM

- En ARM las funciones:
 - Devuelven el valor en R0
 - Utilizan R0-R3 para enviar los parámetros de entrada.

```
MAIN:
int main() {
                                                              MOV R0, #2
int y;
                                                              MOV R1, #3
                                                              MOV R2, #4
   diffofsums(2, 3, 4, 5);
                                                              MOV R3, #5
                                                              BL DIFFOFSUMS
                                                              MOV R4, R0
                                               DIFFOFSUMS: ADD R8, R0, R1
int diffofsums(int f, int g, int h, int i) {
                                                              ADD R9, R2, R3
int result;
                                                              SUB R4, R8, R9
result = (f + g) - (h + i);
                                                              MOV RO, R4
return result;
                                                              MOV PC. LR
```

Funciones en ARM

- ARM utiliza el **BL** como instrucción para el llamado a funciones.
- BL guarda el valor de retorno en RL permitiendo a la función una vez finalizada regresar desde donde fue llamada

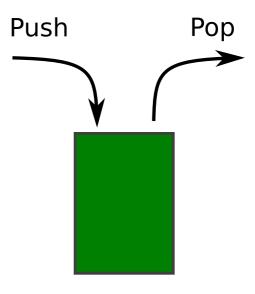


STACK

- Los parámetros son enviados de izquierda a derecha en los registros R0-R3, si se requiere mayor cantidad de parámetros, utilizaremos el STACK
- El STACK, es una porción de memoria que se utiliza para guardar información temporal dentro de una función o programa.
- Esta porción de memoria se expande cuando se requiere guardar información y se libera en el momento que esa información no es mas necesaria.
- Las funciones pueden por ejemplo utilizar el stack para las variables locales, desocupando la memoria al salir de la función.

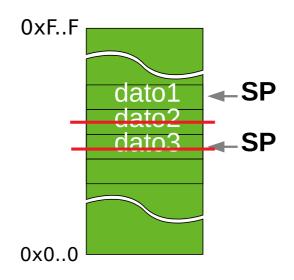
STACK

- El STACK es una lista ordenada del tipo last-in-first-out (LIFO),
- Como en una pila, el nuevo dato se apila sobre el anterior y cuando se extrae un dato es el último que se agregó.
- En ARM se utiliza el registro R13 o SP para el manejo del STACK, este registro contendrá la dirección del último dato guardado.
- Además por cada dato agregado, el stack se expandirá a posiciones de memoria mas bajas.



STACK

• Ejemplo simple de utilización del STACK



Al entrar a una función, se crean dos variables locales dato2 y dato3

Dentro de la función se utilizan las variables

Al salir de la función Se recupera el espacio

Uso del Stack

- Una función no debe afectar ningún registro o zona de memoria que el programa que llamó a la función utilice.
- Para evitar problemas, toda función guardará en el stack los registros utilizados dentro de la misma para recuperar su valor antes de retornar el programa que la llamó.

DIFFOFSUMS: ADD R8, R0, R1
ADD R9, R2, R3
SUB R4, R8, R9

debe ser modificado por contener el valor de retorno

Registros modificados dentro de la función

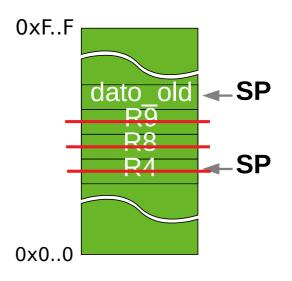
Registros modificados dentro de la función

MOV R0, R1
MOV R0, R4
MOV PC, LR

Uso del Stack

 Los pasos para guardar los registros en el stack serán los siguientes:

1) Hacer espacio DIFFOFSUMS: en el stack	SUB SP, SP, #12
2) Guardar los registros	STR R9, [SP, #8] STR R8, [SP, #4] STR R4, [SP] ADD R8, R0, R1
3) Ejecutar la función utilizando los registros	ADD R9, R2, R3 SUB R4, R8, R9 MOV R0, R4 LDR R4, [SP] LDR R8, [SP, #4]
4) Restaurar los valores originales del stack	LDR R9, [SP, #8] ADD SP, SP, #12
5) Desocupar el espacio en el stack	MOV PC, LR



 Para facilitar el almacenamiento y carga de los registros al stack, ARM provee funciones de carga y almacenamiento de múltiples registros.

Full Descending

Almacenamiento **DIFFOFSUMS**: **STMFD SP!**, **{R4**, **R8**, **R9**}

ADD R8, R0, R1 ADD R9, R2, R3 SUB R4, R8, R9

MOV R0, R4

LDMFD SP!, {R4, R8, R9}

Carga MOV PC, LŔ

LDM (b20=1) / STM (b20=0)

3 3 2 2 1 0 9 8	2 7	2 6	2 5	2 4	2	2 2	2	2	19	18	1 7	16	15	14	13	1 2	1	10	9	8	7	6	5	4	3	2	1	0
cond	1	0	0		СО	nfi	ig.			R	n					ı	_is	ta	do	de	re	egi	st	ros				

- Las Instrucciones LDM y STM permite ser configuradas para diferentes modos de funcionamiento.
- Como cualquier instrucción de load / store, requiere de un registro como base para acceder a la posiciones de memoria que interactúan con los registros.
- Además de esto, y como pueden ser varios los registros leídos o a guardar, se requiere que la propia instrucción calcule las posiciones donde se encuentran los restantes.

Según el sufijo elegido, se obtiene los siguientes modo de

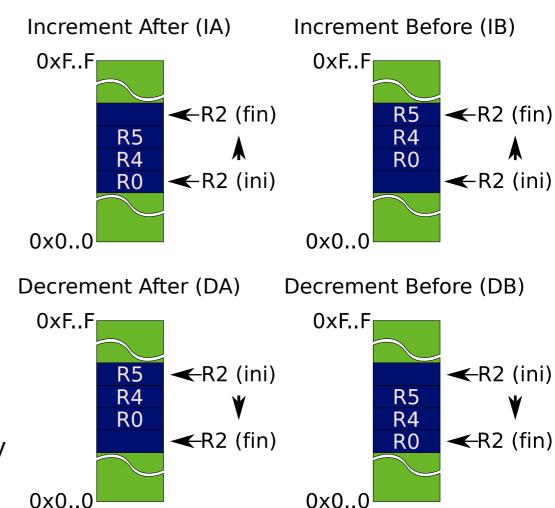
direccionamiento.

Nemónico según el modo de direccionamiento (IA,IB,DA,DB)

STMxx R2!,{R0,R4,R5}

Indica que el registro base se actualizará

No importa el orden, siempre se comienza por el registro mas bajo y se continua en orden ascendente



- Los modos indicados, permiten usar las instrucciones de CyA múltiples para una diversidad de aplicaciones.
- La utilización específica para el STACK, será una combinación de las indicada anteriormente.
- Combinando entonces estos modo de direcciones, se pueden implementar 4 tipos de stack.

Full/Empty: indica si el puntero apunta al último dato guardado (full) o no (empty)

Ascending/Descending: Se refiere a la manera que se guardan los datos en memoria

Nombre	Nemónico	Nemónico LDM	Nemónico STM
Full Ascending	FA	LDMDA	STMIB
Empty Ascending	EA	LDMDB	STMIA
Full Descending	FD	LDMIA	STMDB
Empty Descending	ED	LDMIB	STMDA

STACK Full Descending

- PUSH y POP son sinónimos para las instrucciones STM y LDM respectivamente configuradas en el modo FD y utilizando el stack como registro base.
 - PUSH {regs} → STMFD SP!, {regs}
 - POP {regs} → LDMFD SP!, {regs}
- El Full Descending queda entonces como el modelo estándar utilizado por ARM, y es utilizado en otras arquitecturas como modo único de manejo del STACK (ejemplo x86)

Reduciendo los registros salvados

- ARM divide a los registros, en registros preservados y no preservados.
 - Preservados: R4-R11,SP y LR, una función debe guardar estos registros antes de ser modificados y restaurado su valor antes de salir de la función El código que llama a una función supone que esta no va a modificar estos registros.
 - No preservados; R0-R3, R12 pueden ser cambiados libremente, no se puede asumir que no serán modificados luego de llamar a una función.

DIFFOFSUMS: PUSH {R4}
ADD R1, R0, R1
ADD R3, R2, R3
SUB R4, R1, R3
MOV R0, R4
POP {R4}
a ser salvado
(callee-saved)

No es necesario guardarlos o se encarga el llamador de hacerlo (callersave)

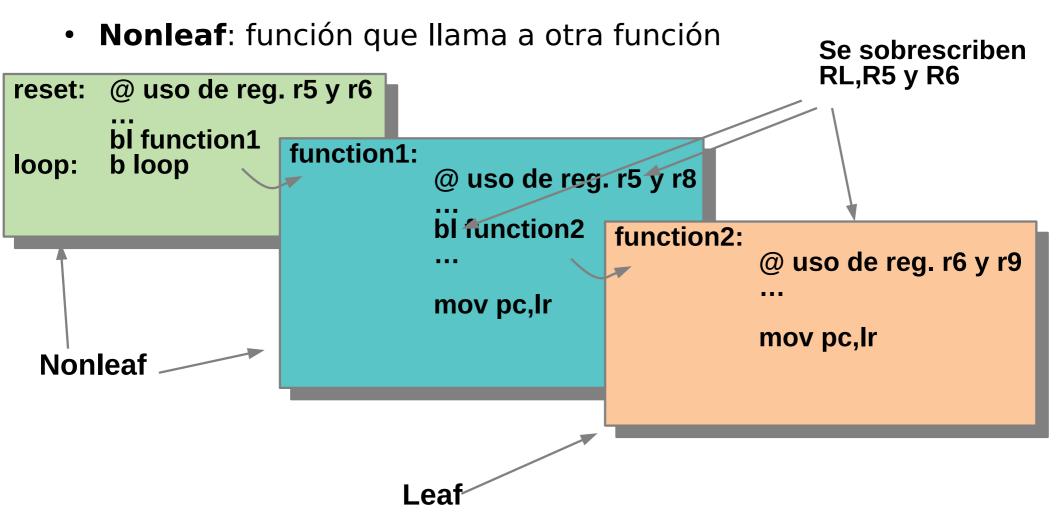
Reduciendo los registros salvados

• Tabla con resumen de los registros Preservados y No Preservados.

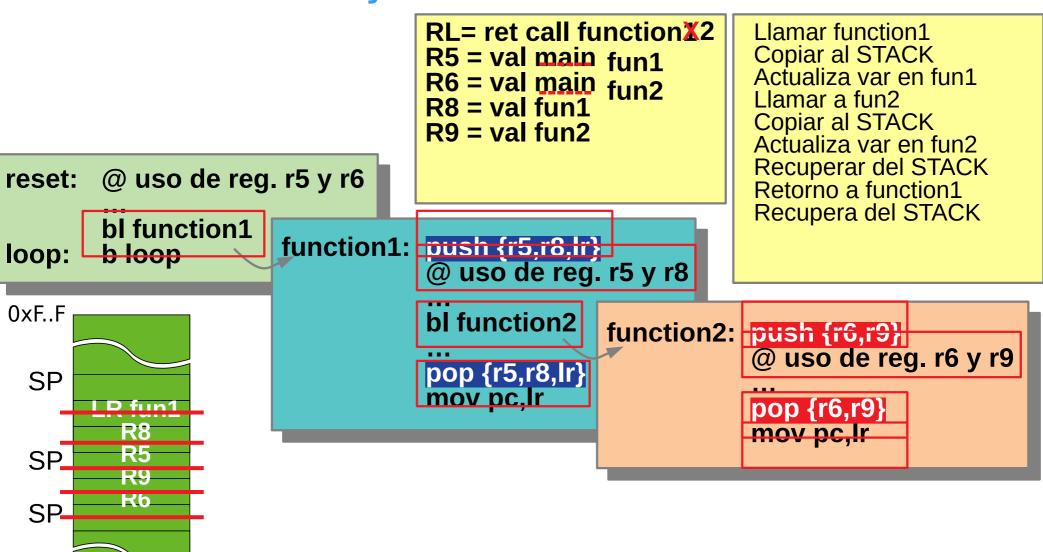
Preservados	No Preservados
R4-R11	Registro temporal R12
Stack Pointer R13 (SP) Esta implícito en el propio uso. Al entrar a una función los datos guardados en el stack son descargados al salir, recuperando su valor inicial.	Registros de Argumentos R0-R3
Dirección de Retorno R14 (LR) Si una función llama a otra, se deberá guardar la dirección de retorno de la primera.	Current Program Status Register (CPSR) Las Banderas no son preservadas entre llamado de funciones
Stack sobre el SP	Stack por debajo del SP

Funciones *leaf* y *nonleaf*

Leaf: función que no llama a otra función.



Funciones *leaf* y *nonleaf* uso del STACK



0x0..0

Uso del STACK

```
int f1(int a, int b) {
    int i, x;
    x = (a + b)*(a - b);
    for (i=0; i<a; i++)
        x = x + f2(b+i);
    return x;
}
```

```
int f2(int p) {
    int r;
    r = p + 5;
    return r + p;
}
```

```
PUSH {R4, R5, LR}
F1:
      ADD R5, R0, R1
      SUB R12, R0, R1
      MUL R5, R5, R12
      MOV R4, #0
FOR:
      CMP R4, R0
      BGE RETU
      PUSH {R0, R1}
      ADD R0, R1, R4
      BL F2
      ADD R5, R5, R0
      POP {R0, R1}
      ADD R4, R4, #1
      B FOR
RETU: MOV R0, R5
      POP {R4, R5, LR}
      MOV PC, LR
```

F2: PUSH {R4}
ADD R4, R0, 5
ADD R0, R4, R0
POP {R4}
MOV PC, LR

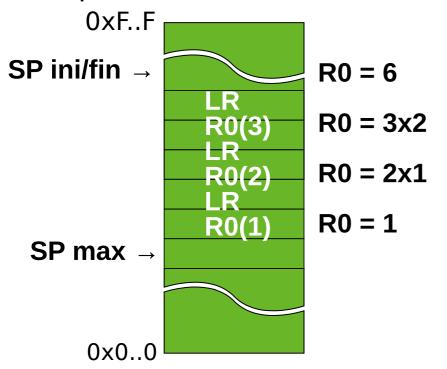
RL debe ser guardado en toda función nonleaf

Funciones Recursivas

- Son funciones nonleaf que se llaman a si mismas
- Son ambas llamador y llamada

• Se deben guardar los registros preservados, pero también los

no preservados.



FACTORIAL: PUSH {R0, LR}
CMP R0, #1
BGT ELSE
MOV R0, #1
ADD SP, SP, #8
MOV PC, LR
SUB R0, R0, #1
BL FACTORIAL
POP {R1, LR}
MUL R0, R1, R0
MOV PC, LR

Funciones Recursivas

FAC: PUSH {R0, LR}

CMP RO, #1 BGT ELSE

MOV R0, #1

ADD SP, SP, #8 MOV PC, LR

ELSE: SUB R0, R0, #1

BL FACTORIAL

POP {R1, LR}

MUL R0, R1, R0

MOV PC, LR

FAC: PUSH {R0, LR}

CMP R0, #1

BGT ELSE

MOV R0, #1

ADD SP, SP, #8

MOV PC, LR

ELSE: SUB R0, R0, #1

BL FACTORIAL

POP {R1, LR}

MUL R0, R1, R0

MOV PC, LR

FAC: PUSH {R0, LR}

CMP R0, #1 BGT ELSE

MOV DO #1

MOV R0, #1

ADD SP, SP, #8

MOV PC, LR

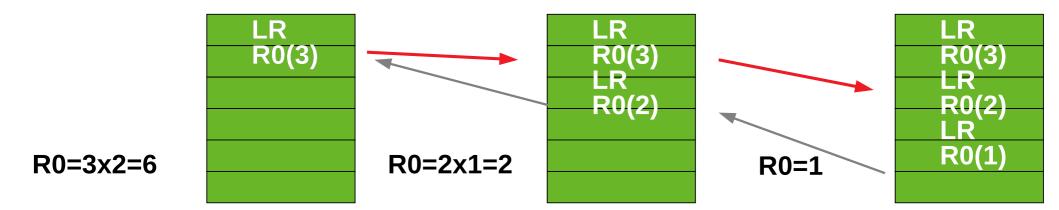
ELSE: SUB RO, RO, #1

BL FACTORIAL

POP {R1, LR}

MUL R0, R1, R0

MOV PC, LR



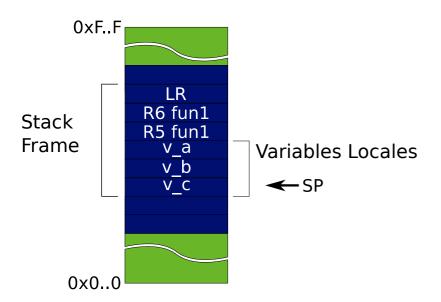
Variables Locales

Las variables locales solo son accedidas por la propia

función.

 Se utiliza el stack para guardarlas.

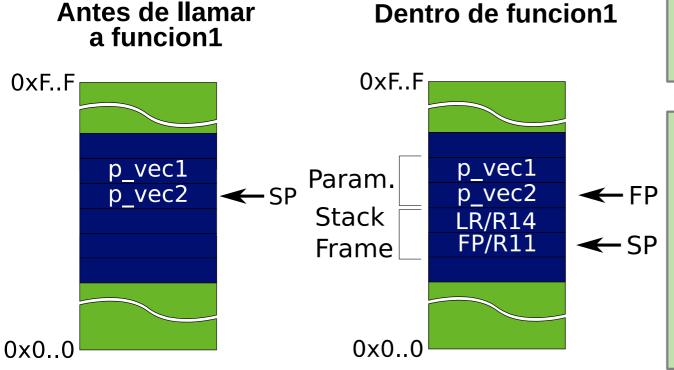
Al salir se recupera la memoria



```
Function1: push{r5,r6,lr}
           sub sp,#12
           str r0,[sp,#-8] @v_a
           str r1,[sp,#-4] @v_b
           str r2,[sp]
                          @v a
            bl function2
           add sp,#12
           pop {r5,r6,lr}
           mov pc,lr
```

Parámetros por stack

 Cuando se debe enviar mas de 4 parámetros se utiliza el STACK



Main:

ldr r1,=vector1

push {r1}

ldr r1,=vector2

push {r1}

bl funcion1

add sp,#8

...

function1: push {fp,lr}
add fp,sp,#8
...
ldr r0,[fp,]
ldr r1,[fp,#4]
...
pop {fp,lr}
mov pc,lr

Bibliografía

Harris & Harris. Digital design and computer architecture: ARM edition. Elsevier, 2015. Capítulo 6.

William Hohl & Christopher Hinds. ARM assembly language. Fundamentals and techniques. 2nd edition. CRC press, 2015. Capítulo 13.

¿ Preguntas ?