

SEMANA 4.

Session 4: Outline

- CMP Instruction
- Conditional Execution
- Branch Instructions
- Directives ADR, DCB and ALIGN
- STACK
- LDM-STM Instructions

Reading



Pontificia Universidad
JAVERIANA

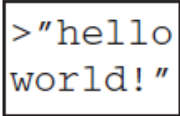


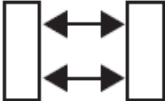
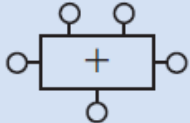

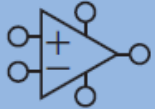




Res. 2333 del 2012

Vigilada Mineducación Resolución 12220 de 2016

- https://spectrum.ieee.org/nanoclast/semiconductors/processors/the-foundry-at-the-heart-of-darpar-plan-to-let-old-fabs-beat-new-ones?utm_source=semiconductors&utm_campaign=semiconductors-08-14-18&utm_medium=email

Principal	Funds (millions of US\$)	Institutions	Description
Max Shulaker	61	MIT, Skywater, Stanford University, Carbonics	Refined RRAM and CNFET processes for commercial fabrication of an integrated, monolithic 3D SoC

Application Software	
Operating Systems	
Architecture	
Micro-architecture	
Logic	
Digital Circuits	
Analog Circuits	
Devices	
Physics	

INSTRUCCIÓN CMP

- Compara dos números sin signo, y cambia las banderas de acuerdo al resultado de la comparación.

`CMP Rn,Op2 ;compare Rn with Op2 and set the flags`

- **Los operandos no se modifican.**
- No hay registro de destino.
- Es como si se realizara una SUBS, solo que el resultado se descarta, cambiando las banderas C y Z.
- Actualiza automáticamente las banderas.

Instruction	C	Z
Rn > Op2	1	0
Rn = Op2	1	1
Rn < Op2	0	0

EJECUCIÓN CONDICIONAL

- El concepto de ejecución condicional en ARM esta implementado para todas las instrucciones.
- No solo las instrucciones de salto puede ser condicionales.
- Si no se añade una condición al final de una instrucción, esta se ejecutará de manera incondicional.



- Para hacer una instrucción condicional, simplemente se coloca la sintaxis de la condición de la siguiente tabla en frente de ella.

Field Mnemonic	Condition Code Flags	Meaning	Code
EQ	Z set	Equal	0000
NE	Z clear	Not equal	0001
CS/HS	C set	Unsigned \geq	0010
CC/LO	C clear	Unsigned $<$	0011
MI	N set	Negative	0100
PL	N clear	Positive or zero	0101
VS	V set	Overflow	0110
VC	V clear	No overflow	0111
HI	C set and Z clear	Unsigned $>$	1000
LS	C clear and Z set	Unsigned \leq	1001
GE	$N \geq V$	Signed \geq	1010
LT	$N \neq V$	Signed $<$	1011
GT	Z clear, $N = V$	Signed $>$	1100
LE	Z set, $N \neq V$	Signed \leq	1101
AL	Always	Default	1110

Conditional branch instructions for value comparison operations

Required branch control	Unsigned data	Signed data
If (R0 equal R1) then branch	BEQ label	BEQ label
If (R0 not equal R1) then branch	BNE label	BNE label
If (R0 > R1) then branch	BHI label	BGT label
If (R0 >= R1) then branch	BCS label/BHS label	BGE label
If (R0 < R1) then branch	BCC label/BLO label	BLT label
If (R0 <= R1) then branch	BLS label	BLE label

EJEMPLO

```
MOV    R1, #10
```

```
MOV    R2, #12
```

```
CMP    R2, R1
```

```
MOVEQ  R4, #20 ;ESTA LINEA NO SE EJECUTA PORQUE NO SON IGUALES
```

```
HERE   B     HERE
```

```
CMP    R1, #0
```

```
ADDNE  R1, R1, #10 ;AÑADE 10 A R1 SI ESTE NO ES CERO
```

```
B      HERE
```

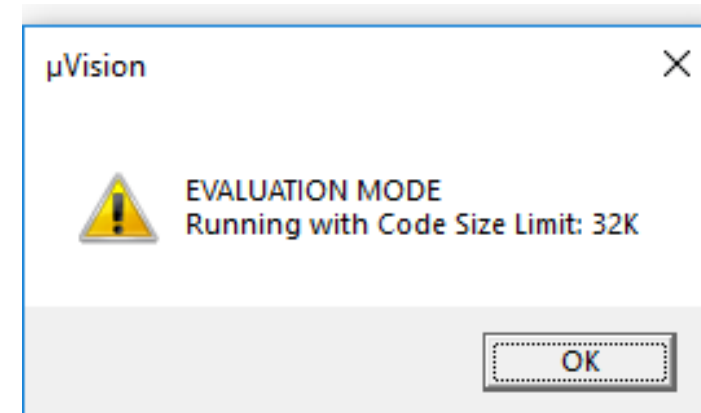
Se puede configurar banderas también en una ejecución condicional

```
ADDNES  R1, R1, #10
```

La ejecución condicional permite ganar tiempo al evitar la penalidad de salto.

BRANCH INSTRUCTION

- When the processor executes a branch instruction, this offset is added to the Program Counter.
- The 24-bit target address is shifted left twice and that allows a jump to -32M to $+32\text{M}$ bytes of memory locations from the address of current instruction.



EJEMPLO



Res. 2333 del 2012

Vigilada Mineducación Resolución 12220 de 2016

Rewrite following assembly language using conditional instructions

```
1      AREA SUMA_SIMPLE, CODE, READONLY
2      ARM
3      AD_RAM EQU 0XE0001000
4      RAM    RN  R11
5      ENTRY
6      LDR RAM, =AD_RAM
7      LDR R1, =0X01
8      LDR R2, =0X02
9      CMP R1, R2
10     BEQ RESTAR; SI R1 Y R2 SON IGUALES SALTA A RESTAR
11     ADDS R1, R2, R3
12     RESTAR SUB R3, R4, R5
13     HERE   B    HERE
14     END
```

Internal

PC \$

Mode

States

Sec

0x0000001C

Supervisor

10

0.00000017

SOLUCIÓN



Res. 2333 del 2012

```
1      AREA SUMA_SIMPLE, CODE, READONLY
2      ARM
3      AD_RAM EQU 0XE0001000
4      RAM    RN    R11
5      ENTRY
6      LDR    RAM, =AD_RAM
7      LDR    R1, =0X01
8      LDR    R2, =0X02
9      CMP    R1, R2
10     ADDNE   R1, R2, R3
11     SUBEQ   R3, R4, R5
12     HERE   B    HERE
13     END
```

Internal	
PC \$	0x00000018
Mode	Supervisor
States	8
Sec	0.00000013

EJEMPLO

- Escriba un programa que sume el número 0x99999999 10 veces. Use instrucciones de ejecución condicional.

```
1      AREA SUMA_SIMPLE, CODE, READONLY
2      ARM
3  AD_RAM EQU 0xE0001000
4  RAM    RN  R11
5      ENTRY
6      LDR RAM, =AD_RAM
7      LDR R0, =0X99999999
8      LDR R1, =10
9  CICLO_RESTA
10     ADDS R2, R2, R0
11     ADDCS R3, R3, #1
12  CICLO_FOR
13     SUBS R1, R1, #1
14     BNE CICLO_RESTA
15  HERE  B  HERE
16     END
```

Instrucción BNE

- Usa la bandera Z.

```
BACK      .....      ;start of the loop
          .....      ;body of the loop
          .....      ;body of the loop
SUBS      Rn,Rn,#1      ;Rn = Rn - 1, set the flag Z = 1 if Rn = 0
BNE      BACK          ;branch if Z = 0
```

- Rn contiene el número de repeticiones

EJEMPLO

- Escriba un programa que sume mil veces el número 9 al R0, coloque la suma final en R4.

```
LDR R2, =1000
MOV R0, #0
CICLO1 ADD R0, R0, #9
        SUBS R2, R2, #1
        BNE CICLO1
        MOV R4, R0
```

...	R0	0x00002328
...	R1	0x00000000
...	R2	0x00000000
...	R3	0x00000000
...	R4	0x00002328

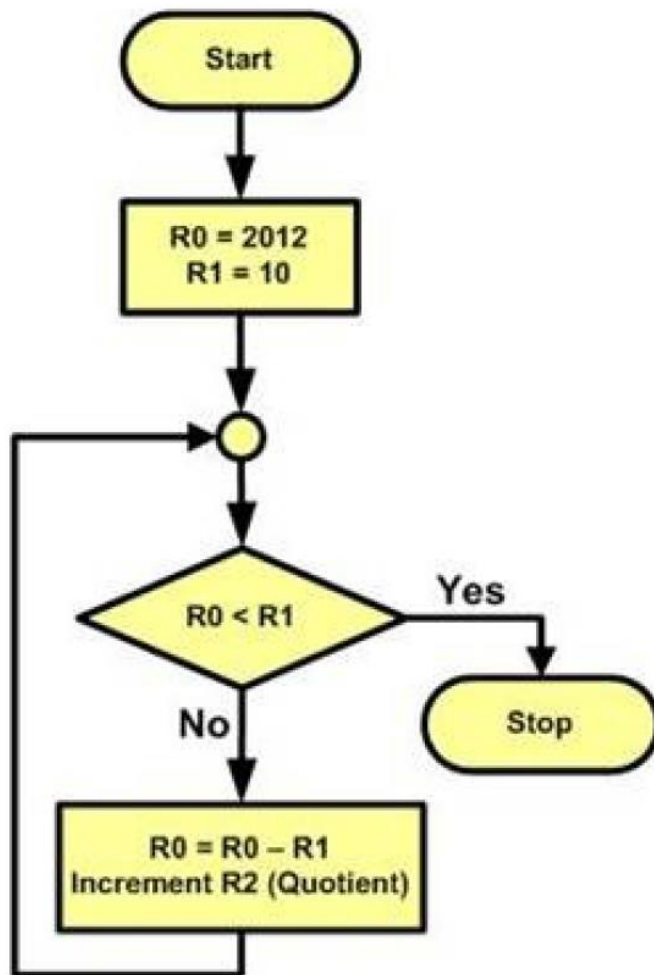
BCC (branch if carry clear, C=0)

- Se usa la bandera C para tomar la decisión de salto. Si C=0 realiza el salto correspondiente.
- EJEMPLO: Sumar 10 veces 0x99999999, sin usar ADC.

```
MOV      R1, #0
MOV      R0, #0
LDR      R2, =0x99999999
MOV      R3, #10          ; CONTADOR
CICLO1   ADDS      R0, R0, R2      ; SUMA Y CONFIGURA BANDERAS
        BCC  CICLO2
        ADD      R1, R1, #1      ; Si C=1, incrementa la palabra alta
CICLO2   SUBS      R3, R3, #1      ; Decremento
        BNE  CICLO1
```

EJEMPLO

- Escriba un programa que realice la operación de división mediante restas.



	LDR	R0, =2000	; NUMERADOR
	MOV	R1, #10	; DENOMINADOR
	MOV	R2, #0	; COCIENTE
CICLO1	CMP	R0, R1	; COMPARA R0 CON R1
	BLO	HERE	
	SUB	R0, R0, R1	
	ADD	R2, R2, #1	
	B	CICLO1	
HERE	B	HERE	

DIRECTIVA ADR.

- Carga registros con la dirección de ubicación en memoria de una etiqueta o conjunto de datos.

ADR Rn,label

start

MOV
ADR

r0,#10
r4,start

DIRECTIVA DCB.

- Inicializa uno o más bytes de memoria.

MYVALUE DCB 5 ;MYVALUE = 5

MYMSAGE DCB “HELLO WORLD” ;string

Data Size	Bits	Decimal	Hexadecimal	Directive	Instruction
Byte	8	0 – 255	0 - 0xFF	DCB	STRB/LDRB
Half-word	16	0 – 65535	0 - 0xFFFF	DCW	STRH/LDRH
Word	32	0 – $2^{32}-1$	0 - 0xFFFFFFFF	DCD	STR/LDR

Directive	Description
DCB	Allocates one or more bytes of memory, and defines the initial runtime contents of the memory
DCW	Allocates one or more halfwords of memory, aligned on two-byte boundaries, and defines the initial runtime contents of the memory.
DCWU	Allocates one or more halfwords of memory, and defines the initial runtime contents of the memory. The data is not aligned.
DCD	Allocates one or more words of memory, aligned on four-byte boundaries, and defines the initial runtime contents of the memory.
DCDU	Allocates one or more words of memory and defines the initial runtime contents of the memory. The data is not aligned.

EJEMPLO

- Almacenando valores constantes en la memoria.

```
LDR      R2, =DATOS_FIJOS
```

```
LDRB     R0, [R2]
```

```
ADD      R1, R1, R0
```

```
HERE     B     HERE
```

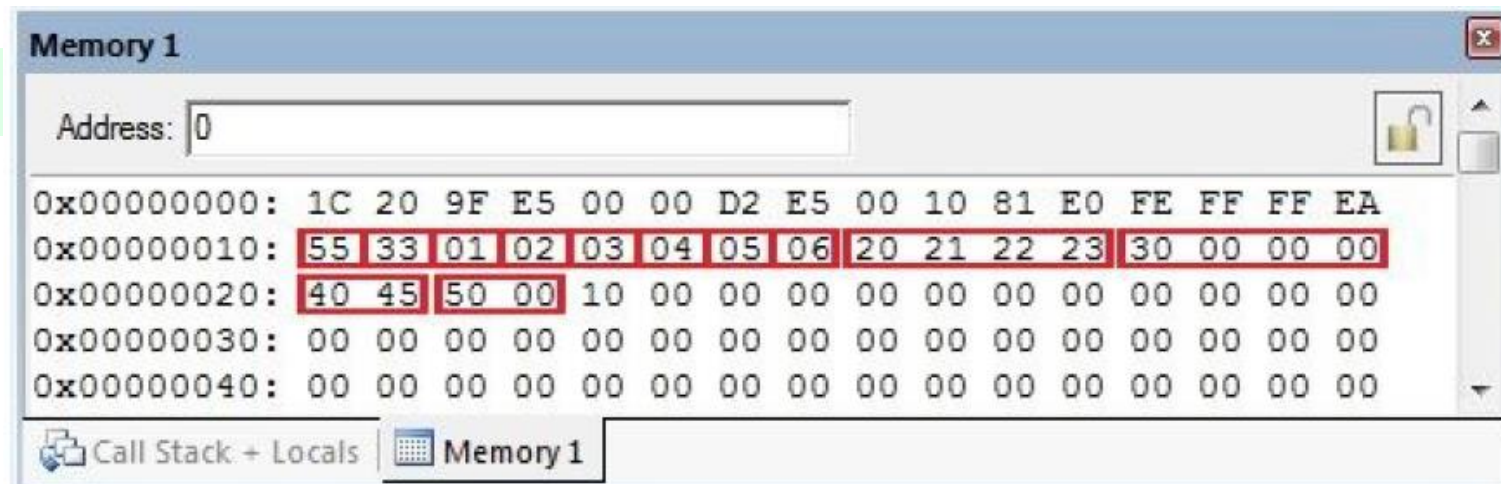
```
DATOS_FIJOS
```

```
DCB      0x55, 0x33, 1, 2, 3, 4, 5, 6
```

```
DCD      0x23222120, 0x30
```

```
DCW      0x4540, 0x50
```

```
END
```



ARM INDEXING MODES

Mode	ARM Assembly	Address	Base Register
Offset	LDR R0, [R1, R2]	$R1 + R2$	Unchanged
Pre-index	LDR R0, [R1, R2]!	$R1 + R2$	$R1 = R1 + R2$
Post-index	LDR R0, [R1], R2	R1	$R1 = R1 + R2$

Function Calls and Returns

ARM uses the branch and link instruction (BL) to call a function and moves the link register to the PC (BX LR) to return from a function.

High-Level Code

```
int main() {  
    int y;  
    ...  
    y = diffofsums(2, 3, 4, 5);  
    ...  
}
```

```
int diffofsums(int f, int g, int h, int i) {  
    int result;  
  
    result = (f + g) - (h + i);  
    return result;  
}
```

```
1      AREA BL_Example, CODE, READONLY  
2      ARM  
3      ENTRY  
4          MOV R0, #2  
5          MOV R1, #3  
6          MOV R2, #4  
7          MOV R3, #5  
8          BL DIFF_OF_SUMS ; call function  
9          MOV R4, R0      ; y = returned value  
10     HERE    B HERE  
11     DIFF_OF_SUMS  
12         ADD R8, R0, R1    ; R8 = f+g  
13         ADD R9, R3, R3    ; R9 = h + i  
14         SUB R4, R9, R8    ; result = (f + g) - (h + i)  
15         MOV R0, R4  
16         BX LR  
17     END
```


DIRECTIVA ALIGN

- **Alinea datos o código a un limite específico de memoria.**
- Indica como deben ser ubicadas las direcciones en memoria.
- Cuando se usa para CODE y READONLY, alinea direcciones de 4 bytes.
- Tiene un número n que indica que la información debe ser puesta en memoria con direcciones de 2^n .
- Por ejemplo, ALIGN=3, la información se colocara en 0x50000, 0x50008,...
- Alinea la ubicación actual a un limite específico, llenando con ceros o instrucciones NOP.
- Si no se especifica ningún parámetro, se alinea por defecto a una palabra.

EJEMPLO

- Cuando no hay ALIGN, DCB ubica los datos en la primer ubicación vacía.

```
1  AREA PRIMER_EJEMPLO, CODE, READONLY
2  ARM ;Define el código como ARM
3  ENTRY
4      ADR      R2, DTA
5      LDRB     R0, [R2]
6      ADD      R1, R1, R0
7  HERE      B      HERE
8  DTA  DCB  0x55
9      DCB  0x22
10 END
```

Address: 0X10

0x00000010: 55 22

- Se guarda en ubicaciones múltiplos de 4.

```
1      AREA PRIMER_EJEMPLO, CODE, READONLY
2      ARM ;Define el código como ARM
3      ENTRY
4          ADR      R2, DTA
5          LDRB     R0, [R2]
6          ADD      R1, R1, R0
7      HERE      B      HERE
8      DTA DCW 0x55
9          ALIGN    4
10     DCB 0x22
11     END
```

Address: 0X10

0x00000010: 55 00 00 00 22 00 00 00 00 00

PILA

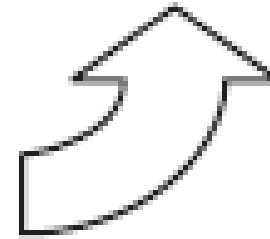
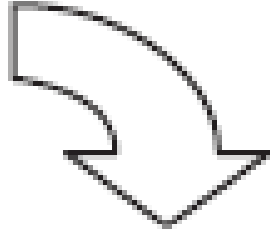
- La pila es una sección de la RAM usada por la CPU para almacenar información temporal.
- El R13 ó SP apunta la dirección de la pila.
- Para almacenar información en la pila se usa la instrucción **PUSH**.
- Para cargar contenidos de la pila se usa **POP**.
- En procesadores x86 el puntero de pila se disminuye automáticamente.

STR Rr,[R13] ;Rr can be any registers (R0-R12)

SUB R13,R13,#4 ;decrement stack pointer

PUSH

POP



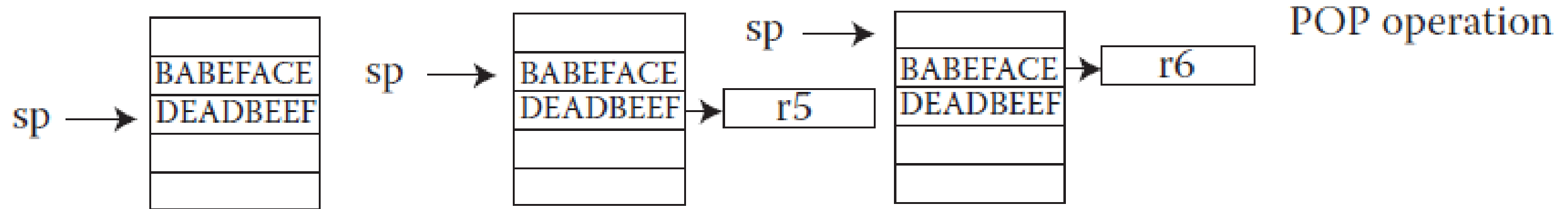
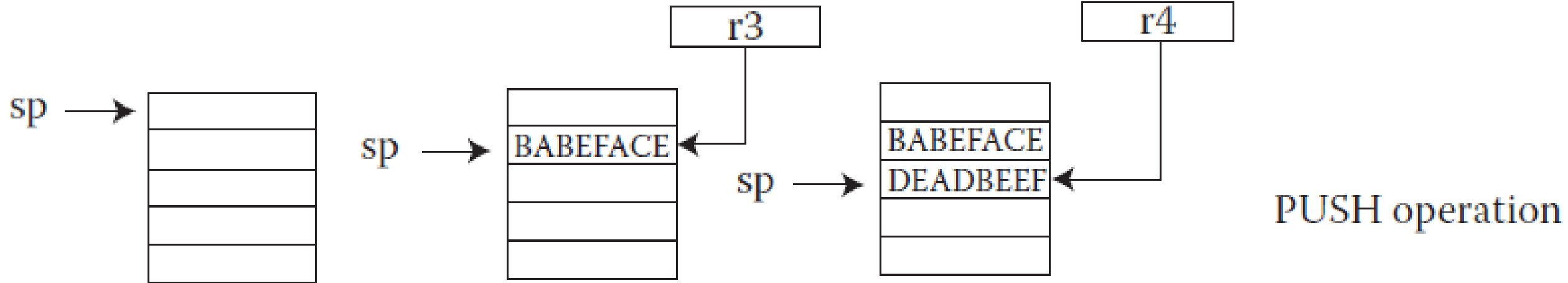
LIFO: (last In-First Out)

```
2      ARM
3  RAM  EQU  0xE0009000
4
5      ENTRY
6
7      LDR      SP, =RAM
8      LDR      R1, =0x40
9      PUSH     {R1}
10
11  HERE      B      HERE
12
13  END
```

```
2      ARM
3 RAM EQU 0xE0009000
4      ENTRY
5          LDR      SP, =RAM
6          LDR      R1, =0xBABEFACE
7          LDR      R2, =0xDEADBEEF
8          PUSH     {R1}
9          PUSH     {R2}
10 HERE      B      HERE
11          END
```

Address:

0xE0008FF0: 00 00 00 00 00 00 00 00 00 00 EF BE AD DE CE FA BE BA



1	AREA SEMANA14, CODE, READONLY	.. R3	0xBABEFACE
2	ARM	.. R4	0xDEADBEEF
3	PILA EQU 0xE0009000	.. R5	0xDEADBEEF
4	ENTRY	.. R6	0xBABEFACE
5	LDR SP, =PILA	.. R7	0x00000000
6	LDR R3, =0xBABEFACE	.. R8	0x00000000
7	LDR R4, =0xDEADBEEF	.. R9	0x00000000
8	PUSH {R3}	.. R10	0x00000000
9	PUSH {R4}	.. R11	0x00000000
10	POP {R5}	.. R12	0x00000000
11	POP {R6}	.. R13 (SP)	0xE0009000
12	HERE B HERE		
13	END		
14			

INSTRUCCIONES LDM/STM

- Instrucciones de carga y almacenamiento múltiple.
- Transfieren una o más palabras usando registros, y un puntero a memoria, que se conoce como el registro base.
- e.g. se usan para guardar el contenido de los registros antes de dar manejo a una excepción.
- Mejora el tiempo de ejecución, ya que solo se debe cargar una instrucción desde memoria.

STM R11,{R0-R10}

;Store R0 through R10 onto memory pointed to by R11

INCREMENTO Y DISMINUCIÓN DE STM Y LDM

Option	Description
--------	-------------

IA	Increment After
-----------	-----------------

IB	Increment Before
-----------	------------------

DA	Decrement After
-----------	-----------------

DB	Decrement Before
-----------	------------------

Se puede también especificar la acción que se realiza al puntero. La acción puede ser incrementar o disminuir después de la operación de pila, como se muestra en la Tabla.

LAS 4 OPCIONES PARA STM Y LDM.

Suponga que R1=0x100. La Figura muestra la memoria después de ejecutar STM R1,{R2-R3}, con las cuatro opciones disponibles.

	Before	After
Increment	<div><div><div>R1 →</div><div><div></div><div>R3</div><div>R2</div><div></div><div></div><div></div><div></div></div><div><div>0x10C</div><div>0x108</div><div>0x104</div><div>0x100</div><div>0xFC</div><div>0xF8</div><div>0xF4</div></div></div><div>After STMIB R1! , {R2 , R3}</div></div>	<div><div><div>R1 →</div><div><div></div><div></div><div>R3</div><div>R2</div><div></div><div></div><div></div></div><div><div>0x10C</div><div>0x108</div><div>0x104</div><div>0x100</div><div>0xFC</div><div>0xF8</div><div>0xF4</div></div></div><div>After STMIA R1! , {R2 , R3}</div></div>
Decrement	<div><div><div></div><div></div><div></div><div></div><div>R3</div><div>R2</div><div></div></div><div><div></div><div></div><div></div><div>0x100</div><div>0xFC</div><div>0xF8</div><div>0xF4</div></div></div> <div>After STMDB R1! , {R2 , R3}</div>	<div><div><div></div><div></div><div></div><div>R3</div><div>R2</div><div></div><div></div></div><div><div></div><div></div><div></div><div>0x100</div><div>0xFC</div><div>0xF8</div><div>0xF4</div></div></div> <div>After STMDA R1! , {R2 , R3}</div>

EJEMPLO: GUARDAR MÚLTIPLES REGISTROS EN PILA

```
1  AREA SEMANA14, CODE, READONLY
2  ARM
3  PILA EQU 0xE0009000
4  ENTRY
5      LDR    SP, =PILA
6      LDR    R0, =0xBABEFACE
7      LDR    R1, =0xDEADBEEF
8      LDR    R2, =0xBCCBB488
9      STM    SP, {R0-R2}
10     MOV    R0, #0
11     MOV    R1, #0
12     MOV    R2, #0
13     LDM    SP, {R0-R2}
14 HERE    B    HERE
15     END
```

-----	R0	0xBABEFACE
-----	R1	0xDEADBEEF
-----	R2	0xBCCBB488
-----	R3	0x00000000
-----	R4	0x00000000
-----	R5	0x00000000
-----	R6	0x00000000
-----	R7	0x00000000
-----	R8	0x00000000
-----	R9	0x00000000
-----	R10	0x00000000
-----	R11	0x00000000
-----	R12	0x00000000
-----	R13 (SP)	0xE0009000

Address: 0xE0008FF0

0xE0008FF0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 CE FA BE BA EF
0xE0009005: BE AD DE 88 B4 CB BC 00 00 00 00 00 00 00 00 00 00

AHORA DISMINUYENDO DESPUÉS

```
1 AREA SEMANA14, CODE, READONLY
2 ARM
3 PILA EQU 0xE0009000
4 ENTRY
5     LDR     SP, =PILA
6     LDR     R0, =0xBABEFACE
7     LDR     R1, =0xDEADBEEF
8     LDR     R2, =0xBCCBB488
9     STMDA   SP, {R0-R2}
10    MOV     R0, #0
11    MOV     R1, #0
12    MOV     R2, #0
13    LDMDA   SP, {R0-R2}
14 HERE B     HERE
15 END
```

Address: 0xE0008FF0

0xE0008FF0: 00 00 00 00 00 00 00 00 00 CE FA BE BA EF BE AD DE 88 B4 CB BC 00
0xE0009005: 00

...	R0	0xBABEFACE
...	R1	0xDEADBEEF
...	R2	0xBCCBB488
...	R3	0x00000000
...	R4	0x00000000
...	R5	0x00000000
...	R6	0x00000000
...	R7	0x00000000
...	R8	0x00000000
...	R9	0x00000000
...	R10	0x00000000
...	R11	0x00000000
...	R12	0x00000000
...	R13 (SP)	0xE0009000

PASANDO PARÁMETROS A SUBROUTINAS

- Las subrutinas necesitan ser capaz de intercambiar datos, estos valores se conocen como parámetros.

- **PASO DE PARÁMETROS EN REGISTROS.**

- Es la forma más rápida de transferir datos entre el programa y la subrutina, pero la subrutina debe esperar que el dato este en un registro específico.

PASO DE PARAMETROS POR REFERENCIA.

- Se envía información a la subrutina para ubicar una posición de un bloque de memoria

- **PASO DE PARÁMETROS POR PILA**

- Es similar a pasar parámetros en memoria, solo que ahora la subrutina usa un registro dedicado para un puntero en memoria.

S

T