- MICROCONTROLADORES CORTEX -

ASSEMBLER Y SU INTERFAZ EN C

Cátedra: Técnicas Digitales II
UTN – FRBA
Marzo 2019

Introducción

- Las computadoras en general se manejan con palabras binarias que le indican que acción debe realizar. Las hemos denominado códigos de operación y los Cortex la leen en el primer ciclo del pipeline que llamamos “Búsqueda de código de operación”.
- Esas palabras binarias son legibles por la computadora pero ilegibles para el programador que prefiere ***nemónicos*** como ADD, SUB, MOV,… los cuales son ilegibles para el microcontrolador sin una previa traducción.
- Hasta ahora hemos utilizado como traductor el compilador C que permite que una sentencia sea traducida a múltiples instrucciones.
- El compilador ***Ensamblador*** (Assembler) genera una correspondencia biunívoca entre una sentencia y un código de operación.

Repertorio de Instrucciones

- Instrucciones de Branch y control.
- Instrucciones de procesamiento de datos.
- Instrucciones de carga y almacenamiento de registros.
- Instrucciones de carga y almacenamiento múltiple de registros.
- Instrucciones de acceso a registros de Status.
- Instrucciones de Coprocesador.

Formato de una línea en Assembler

{*etiqueta*} {*instrucción*|*directivas*|*pseudo-instrucción*} {;*comentario*}

Espacios

o Tab

Ejemplo

```
        AREA    ARMex, CODE, READONLY
        ENTRY                       ; Primera instrucción a ejecutar
start                               ; Etiqueta
        MOV     r0, #10
        MOV     r1, #3
        ADD     r0, r0, r1          ; r0 = r0 + r1
stop
        MOV     r0, #0x18
        LDR     r1, =0x20026
        SVC     #0x123456           ; Llamado a sistema (Ex – SWI)
        END                         ; Fin de archivo
```

Pseudoinstrucción

Excepciones

Exception Number	Exception Type	Priority (Default to 0 if Programmable)	Description
0	NA	NA	No exception running
1	Reset	−3 (Highest)	Reset
2	NMI	−2	Nonmaskable interrupt (external NMI input)
3	Hard fault	−1	All fault conditions, if the corresponding fault handler is not enabled
4	MemManage fault	Programmable	Memory management fault; MPU violation or access to illegal locations
5	Bus fault	Programmable	Bus error (Prefetch Abort or Data Abort)
6	Usage fault	Programmable	Exceptions due to program error
7–10	Reserved	NA	Reserved
11	SVCall	Programmable	System service call
12	Debug monitor	Programmable	Debug monitor (break points, watchpoints, or external debug request)
13	Reserved	NA	Reserved
14	PendSV	Programmable	Pendable request for system device
15	SYSTICK	Programmable	System tick timer
16	IRQ #0	Programmable	External interrupt #0
17	IRQ #1	Programmable	External interrupt #1
...	...	...	...
255	IRQ #239	Programmable	External interrupt #239

Aclaraciones

- Cada archivo resultante del linkeo puede tener:
- Uno o varias secciones de código. Usualmente son sectores de lectura solamente (Read-only)
- Una o varias secciones de datos, Usualmente son sectores de lectura y escritura (Read-write). Suelen ser inicializadas a cero.

Subrutinas

```
    AREA    subrout, CODE, READONLY          ; Nombre de este bloque de código
    ENTRY                                    ; Marca la primera instrucción a
    ejecutar
start
    MOV     r0, #10                          ; Parámetros
    MOV     r1, #3
    BL      doadd                            ; Llamado a subrutina
Stop
    MOV     r0, #0x18                        ;
    angel_SWIreason_ReportException
    LDR     r1, =0x20026                     ; ADP_Stopped_ApplicationExit
    SVC     #0x123456                        ; Llamado al sistema (antes SWI)
doadd
    ADD     r0, r0, r1                       ; Código de la subrutina
    BX  lr                                   ; Retorno de la subrutina
    END                                      ; Fin de archivo
```

Ejecución condicional

EQ	Z seteado	Igual
NE	Z en cero	No igual
CS or HS	C set	Mayor o igual (no signado >=)
CC or LO	C en cero	Menor (no signado <)
MI	N seteado	Negativo
PL	N en cero	Positivo o cero
VS	V set	Overflow
VC	V en cero	Sin overflow
HI	C set y Z en cero	Mayor (no signado >)
LS	C en cero o Z set	Menor o igual (no signado <=)
GE	N y V iguales	Signado >=
LT	N y V difieren	Signado <
GT	Z en cero, N y V iguales	Signado >
LE	Z set, N y V difieren	Signado <=
AL	Siempre.	

Ejemplo

```
ADD      r0, r1, r2        ; r0 = r1 + r2, no actualiza flags
ADDS     r0, r1, r2        ; r0 = r1 + r2, y actualiza flags
ADDSCS r0, r1, r2          ; Si C esta seteado r0 = r1 + r2, y actualiza flags
CMP      r0, r1            ; actualiza flags basada en r0-r1.
```

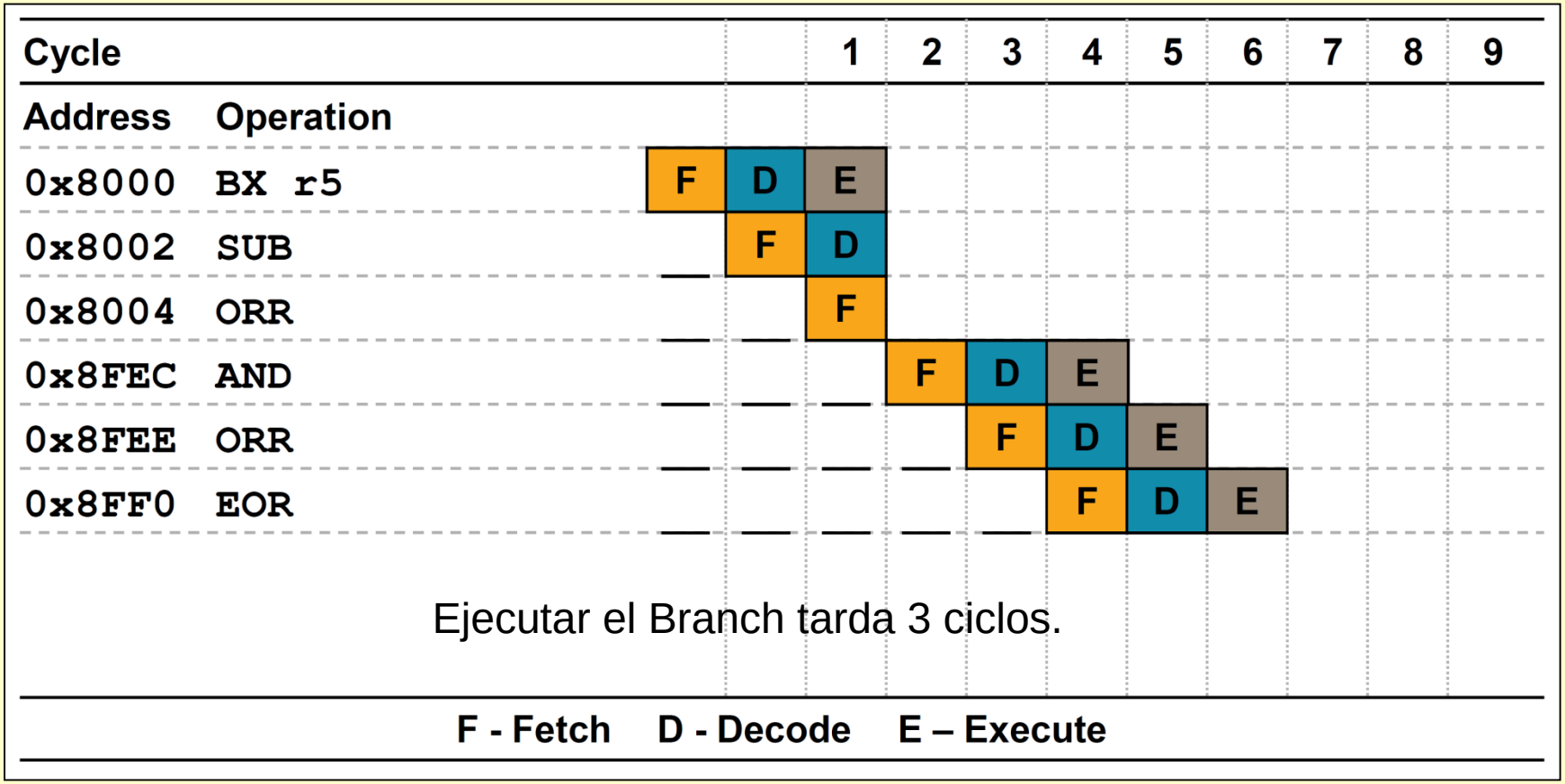
Ejemplo máximo común divisor

```
int gcd (int a, int b)
{
   while (a != b)
     {
if (a > b)
   a = a - b;
else
   a = b - a;
     }
   return a;
}
```

Ejemplo máximo común divisor

```
gcd
        CMP    r0, r1
        BEQ    end
        BLT    less
        SUBS   r0, r0, r1  ; podría ser SUB r0, r0, r1 para ARM
        B      gcd
less
        SUBS   r0, r1, r0  ; podría ser SUB r1, r1, r0 para ARM
        B      gcd
end
```

Caso más desfavorable



Ejemplo máximo común divisor

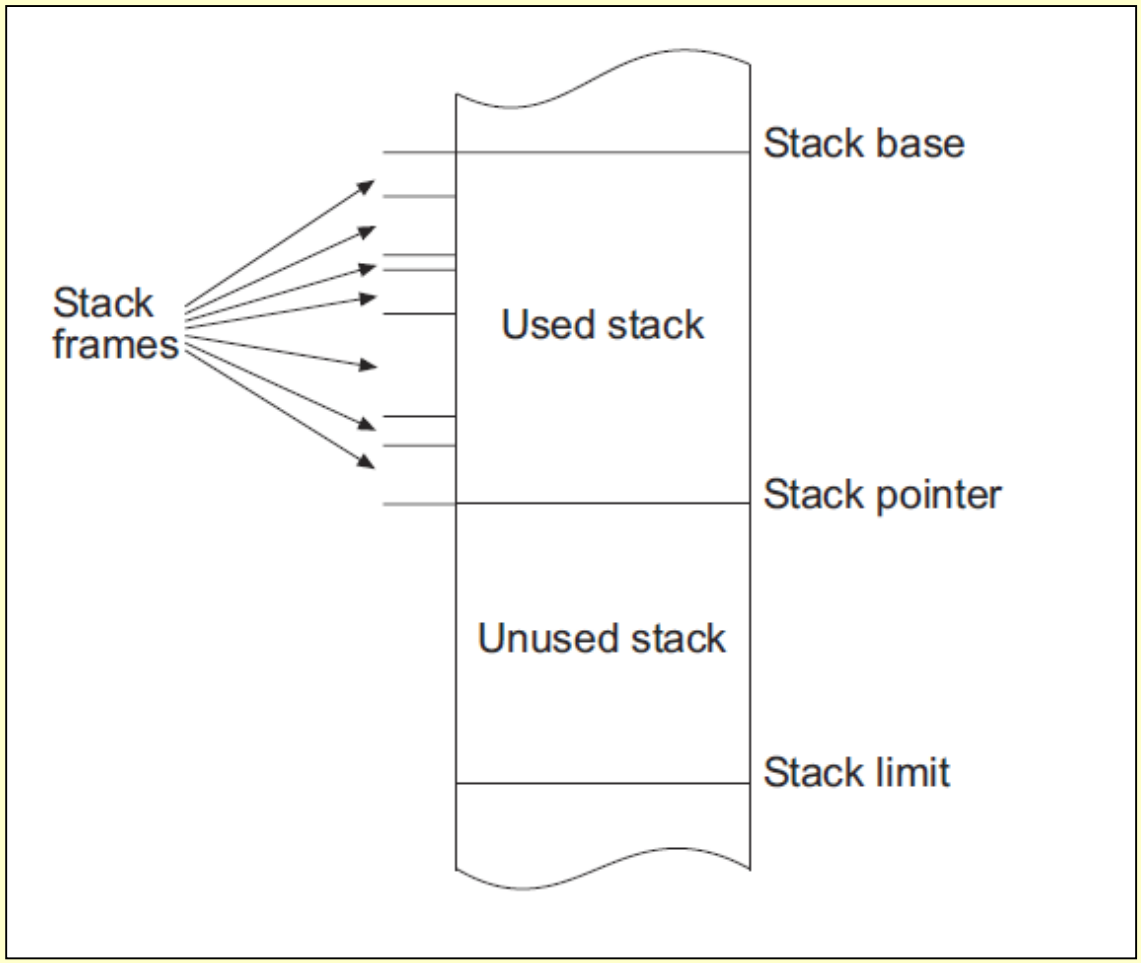
Al tener tantos branches se demora mucho tiempo pues debe rellenarse el pipeline. Es mejor:

```
gcd
        CMP     r0, r1
        SUBGT   r0, r0, r1
        SUBLE   r1, r1, r0
        BNE     gcd
```

Cargar direcciones

```
 AREA    Jump, CODE, READONLY
        ARM                     ; Código en ARM
num     EQU     2               ; Numero de entradas en la tabla de saltos
        ENTRY                   ; Primera instrucción a ejecutar
start
        MOV     r0, #0          ; Parámetros
        MOV     r1, #3
        MOV     r2, #2
        BL      arithfunc       ; Llamar a la función
stop
        MOV   r0, #0x18         ; angel_SWIreason_ReportException
        LDR     r1, =0x20026 ; ADP_Stopped_ApplicationExit
        SVC     #0x123456    ; Llamado al sistema
```

Pila


Stack base
Stack frames
Used stack
Stack pointer
Unused stack
Stack limit

Cargar direcciones

```
 arithfunc                          ; Etiqueta
      CMP    r0, #num               ; Tratar la función como no signada
      BXHS   lr                     ; Si código >= num simplemente retorna
      ADR    r3, JumpTable          ; Carga la dirección de la tabla de saltos
      LDR    pc, [r3,r0,LSL#2]      ; Salta a la rutina apropiada
JumpTable
      DCD    DoAdd
      DCD    DoSub
DoAdd
      ADD    r0, r1, r2             ; Operación 0
      BX     lr                     ; Retorna
DoSub
      SUB    r0, r1, r2             ; Operación 1
      BX     lr                     ; Retorna
      END                           ; Marcar fin de archivo
```

String copy con pseudoinstrucción

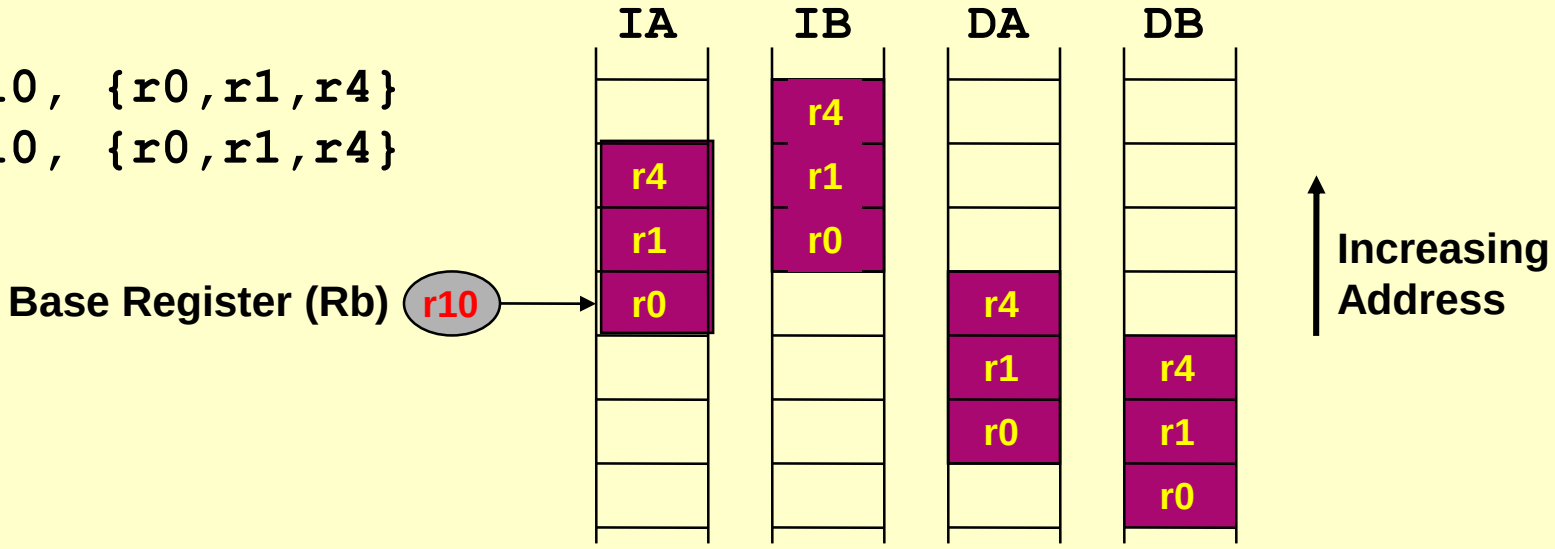
```
AREA    StrCopy, CODE, READONLY
        ENTRY
start
        LDR   r1, =srcstr          ; Puntero al primer string
        LDR   r0, =dststr          ; Puntero al segundo string
        BL    strcopy              ; Llamado a subrutina de copia
stop
        MOV r0, #0x18              ; angel_SWIreason_ReportException
        LDR   r1, =0x20026         ; ADP_Stopped_ApplicationExit
        SVC   #0x123456            ; Llamado al sistema
strcopy
        LDRB r2, [r1],#1           ; Lee byte y actualiza dirección
        STRB r2, [r0],#1           ; Escribe byte actualiza dirección
        CMP r2, #0                 ; ¿Terminó?
        BNE  strcopy               ; Sigue
        MOV pc,lr                  ; Retorna

        AREA              Strings, DATA, READWRITE
srcstr      DCB     “Primer string - fuente",0
```

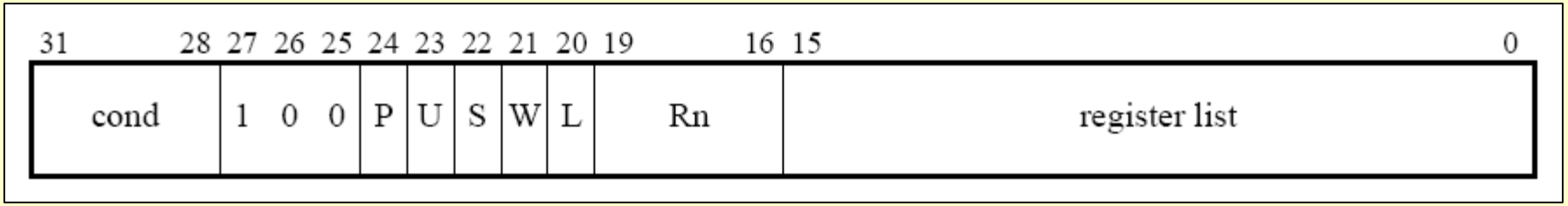
Multiples Load y Store Registros

- Sintaxis:
  - **<LDM|STM>**{<cond>}<addressing_mode> Rb{!}, <register list>
- 4 modos de direccionamiento:
  - **LDMIA / STMIA** increment after
  - **LDMIB / STMIB** increment before
  - **LDMDA / STMDA** decrement after
  - **LDMDB / STMDB** decrement before

```
LDMxx r10, {r0,r1,r4}
STMxx r10, {r0,r1,r4}
```



LDMIA y STMIA




```
LDMIA  R7!, {R0-R3, R5}        ; Load R0 to R3-R5 desde R7, add 20 to R7
STMIA  R0!, {R3, R4, R5}       ; Store R3-R5 to R0: add 12 to R0
```

Movimientos múltiples

```
STMFD       sp!, {r0-r5}     ; Push a una pila descendente
LDMFD       sp!, {r0-r5}     ; Pop de una pila descendente
```

En llamadas anidadas se prefiere llamarlas como PUSH y POP

```
Subrutina1
            PUSH    {r5-r7,lr} ; Push registros de trabajo y lr
        ; código
            BL      subrutina 2
        ; código
            POP     {r5-r7,pc} ; Pop registros de trabajo y pc
```

Macros

```
MACRO            ; Definición de la macro
$label   TestAndBranch  $dest, $reg, $cc
$label   CMP     $reg, #0
         B$cc    $dest
         MEND            ; Macro end
```

Uso de la macro

```
test     TestAndBranch    NonZero, r0, NE
         ...
NonZero
```

Que resulta:

```
test     CMP     r0, #0
         BNE     NonZero
         ...
NonZero
```

Inferfaz C Assembler

- r0-r3 son los registros de argumentos y borradores; r0-r1 son también los de resultados
- r4-r8 son registros a ser salvados por la función llamada y son utilizados para variables locales.
- r9 puede ser un registro a ser salvados por la función llamada o no (en algunas variantes de AAPCS es un registro especial)
- r10-r11 son registros a ser salvados por la función llamada
- r12-r15 son registros especiales

Variables globales

```
AREA        globals, CODE, READONLY
EXPORT      asmsubroutine
IMPORT      globvar
asmsubroutine
    LDR     r1, =globvar  ; Cargar r1 con la dirección de
                          ; la variable global
    LDR     r0, [r1]
    ADD     r0, r0, #2
    STR     r0, [r1]
    BX      LR
    END
```

Interfaz C Assembler – Main en C

```
#include <lpc17xx.h>

extern   int        complementa_a_2(int);
int         a;
int         resul;

int main(void)
{
    a=0x55;

    resul = complementa_a_2(a);
}
```

Interfaz C Assembler – Función en Asm

```
; Ejemplo de una suma entre registros

    AREA Code, CODE, READONLY      ; Dar nombre a esta área de código
    ARM                            ; Código en modo ARM (32bits)
    EXPORT Complementa_a_2

Complementa_a_2

    STMFD       sp!, {r4-r12, lr}  ; Pusheamos los registros para
                                   ; despreocuparnos y poder pisar los
                                   ; contenidos

    MVN         r0,r0              ;Luego, se debe devolver el contenido en R0
    ADD         r0,r0,#1

    LDMFD       sp!, {r4-r12, lr}  ; Recuperamos los registros salvados
    BX          lr                 ; Return

    END
```

Inferfaz C Assembler – Main en C

```
// Función que prepara la suma y resta y llama a la función en Assembler

int a;
int b;
int result1;
int result2;
extern int          suma (int,int);
extern int          resta (int, int);

int main (void){

a=0x20;
b=0x20;

    result1 = suma (a,b);
    result2 = resta (a,b);
}
```

Interfaz C Assembler – Función en Asm

```
        AREA    Code,CODE, READONLY
        ARM

        export  suma
        export  resta

suma
        STMFD   sp!, {r4-r12, lr}   ; Pusheamos los registros
        ADD     r0,r0,r1            ; Función suma
        LDMFD   sp!,{ r4-r12,lr}
        BX      lr

resta   STMFD   sp!, {r4-r12, lr}
        SUB     r0,r0,r1
        LDMFD   sp!,{r4-r12,lr}     ; Recuperamos los registros salvados
        BX      lr
```

Universidad Tecnológica Nacional

Facultad Regional Buenos Aires

Ingeniería Electrónica

Técnicas Digitales II