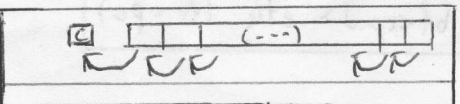
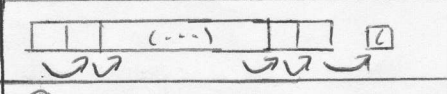
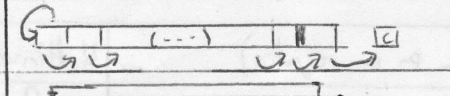
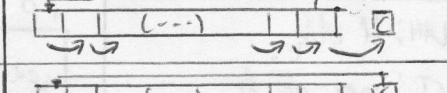
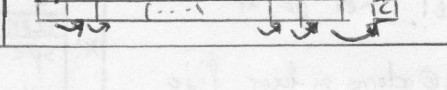
Barrel shifter

r, LSL #n	$r \cdot 2^n$, $n \in [0,31]$, $r \in (\mathbb{N}, \mathbb{Z})$	
r, LSR #n	$r / 2^n$, $n \in [0,32]$, $r \in \mathbb{N}$	
r, ASR #n	$r / 2^n$, $n \in [0,32]$, $r \in \mathbb{Z}$	
r, ROR #n	$n \in [0,31]$, $ROL(n) = ROR(32-n)$	
r, RRX	RLX = adc r,r,r (4 veces)	

Directivas

AREA <etiq>, DATA/CODE	
ENTRY	END
etiq EQU n	
etiq DCB/DCW/DCD/DCQ n	
etiq SPACE <num_bytes>	
ALIGN 2^n	
LDR Rd, =cte (modo inmediato)	
LDR Rd, =etiq (modo directo)	

Modos de direccionamiento

Modo 1	<shifter_op> = R_m / BS(R_m, cte 5bits) / BS(R_m, R_s) / Cte 32 bits (ROR (8bits inmedicto, cte. 4 bits))	
Modo 2/3	Preindexado (con writeback !) [R_n, ± despl] (!)	<modo 2> = R_n ± (R_m / cte 12 bits / BS(R_m, cte 5b))
	Postindexado [R_n], ± despl	<modo 3> = R_n ± (R_m / cte 8 bits)

Lectura / Escritura

ldr/str Rd, <modo 2>	ldr-h/str-h Rd, <modo 3>	b = byte
ldr-b/str-b Rd, <modo 2>	ldr-sh Rd, <modo 3>	h = half-word (2bytes) (alinear en @ par)
	ldr-sb Rd, <modo 3>	- = word (4 bytes) (alinear en @ mult. de 4)
		s = extensión de signo (ca2)

Proceso de datos

Aritméticas	instr[<cond>][s] Rd, Rn, <shifter_op>	s': actualiza N,Z,C,V	
add	Rd = Rn + <shifter_op>	adc	Rd = Rn + <shifter_op> + C
sub	Rd = Rn - <shifter_op>	sbc	Rd = Rn - <shifter_op> - (1-C)
rsb	Rd = <shifter_op> - Rn	rsc	Rd = <shifter_op> - Rn - (1-C)

Lógicas	instr[<cond>][s] Rd, Rn, <shifter_op>	s': actualiza N,Z,C	
and	Rd = AND(Rn, <shifter_op>)	eor	Rd = XOR(Rn, <shifter_op>)
orr	Rd = OR(Rn, <shifter_op>)	bic	Rd = AND(Rn, NOT(<shifter_op>))

Comparación	instr[<cond>] Rn, <shifter_op>	Siempre actualiza flags	
cmp	sub sin resultado	tst	AND sin resultado
cmn	add sin resultado	teq	XOR sin resultado

Movimiento	instr[<cond>][s] Rd, <shifter_op>	s': actualiza N,Z, C según salida BS	
mov	Rd = <shifter_op>	mvn	Rd = NOT <shifter_op>

Multiplicador HW	Rd ≠ Rn ≠ pc		
mul Rd, Rm, Rs	Rd = Rm · Rs	mull RdLo, RdHi, Rm, Rs	$Rd_{HiLo} = Rm \cdot Rs$ (mul 64bits)
mla Rd, Rm, Rs, Rn	Rd = Rm · Rs + Rn	mlal RdLo, RdHi, Rm, Rs	$Rd_{HiLo} = Rm \cdot Rs + Rd_{HiLo}$

Condiciones

Flags

eq	Z=1	mi	N=1	cs	C=1	vs	V=1
ne	Z=0	pl	N=0	cc	C=0	vc	V=0

IN

>	hi	C=1, Z=0	≤	lo	C=0
≥	hs	C=1	≤	ls	C=0, Z=1

ℤ

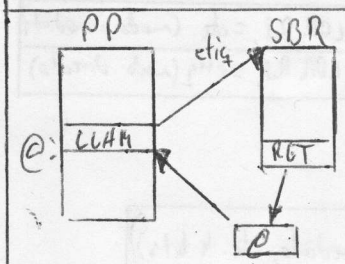
>	gt	Z=0, N=V	≤	lt	N ≠ V
≥	ge	N=V	≤	le	Z=1, N ≠ V

Pila	PUSH {Rd}	str Rd, [sp, #4]!	POP {Rd}	ldr Rd, [sp], #4
Salto	b <cond> etiq	bl <cond> etiq (lr = pc)		

r11 = fp	r13 = sp = @0x40003000
r14 = lr	r15 = pc = @0x40000000

Subrutinas

Unimivel (@ret se almacena en un registro)

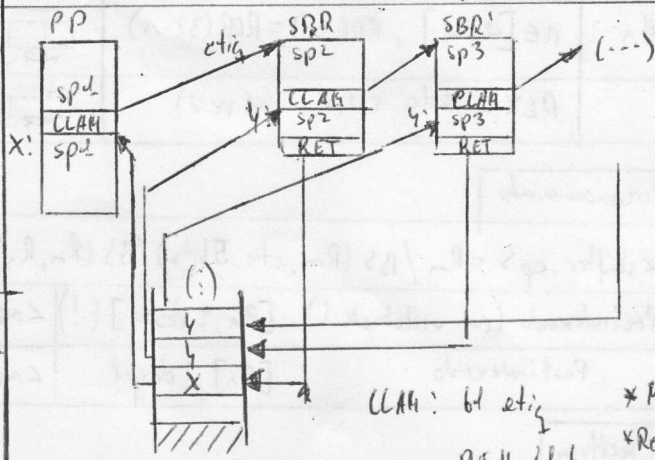


LLAM: bl etiq

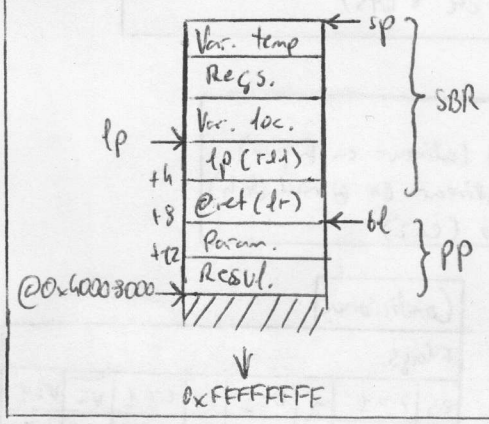
RET: mov pc, lr

* @retorno en lugar fijo
* no anidable

Multinivel (@ret se guarda en la propia SBR / en la pila)



Bloque de activación



Recursividad

- Reservar espacio para resultados
- (Si es necesario) generar nuevos parámetros en regs. auxiliares
- Apilar parámetros
- Salto (bl etiq)

- Apilar @ret, fp (PUSH {lr, r11})
- Generar nuevo fp (mov r11, sp)
- Apilar registros utilizados, recuperar parámetros
- ¿Caso trivial? / Bo / Bi
- ~~Liberar parámetros, recuperar resultado anterior~~
- Realizar operación (res = caso trivial ó rest nuevo caso)

- Desapilar regs., fp {POP r11}, @ret (POP {pc})

- Liberar parámetros, desapilar resultado

Relación ASM - C

Codigo C	Codigo ASM
<tipo> var_global_c	EXTERN var_global_c
<tipo> func_c	EXTERN func_c
extern <tipo> var_s	EXPORT var_s
extern <tipo> sbr_s	EXPORT sbr_s
extern char/int	← DCB / DCD
char/int	ldr(b)/str(b) rd, [rn, =EXTERN]
unsigned / signed	UN / SL (S)

Reglas de paso de parámetros / resultados

- Paso de parámetros de izda. a dcha

```
func(x, y, z) →  mov r2, #x
                 mov r1, #y
                 mov r0, #z
                 PUSH {r0, r2}
                 bl func
```

- Paso por valor en 1 o 2 registros
- Paso por referencia mediante dirección
- Parámetros en regs r0-r3, el resto en pila
- Resultados en regs. r0-r3, si no cabe en memoria por ref.

Registros de controlador

RDAT	De datos
REST	De estado
RCTR	De control

Mapeados en @ memoria

Sincronización por encuesta

1. Comprobar si el periférico puede mandar/recibir nueva información (REST = 1 → Disp)
2. Realizar transferencia de información (RDAT ↔ datos en @/regs.)
3. Si es necesario, enviar señales de control (strobe: RCTR → 1, ..., RCTR → 0)

Sincronización por interrupción

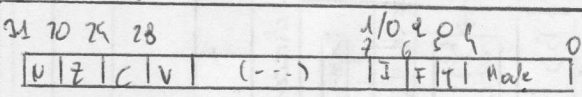
Programa principal

1. Guardar en memoria/pila estado inicial del vector de interrupciones (VICVectAddrn)
2. Cargar en el vector de interrupciones direcciones de las RSI utilizadas
3. Habilitar en máscara VICIntEnable las IRQ que pueden interrumpir
4. Prog. principal → Interrupción → RSI
5. Deshabilitar mediante máscara VICIntEnClr las IRQ utilizadas
6. Recuperar estado inicial del vector de interrupciones

Rutina de servicio (RSI)

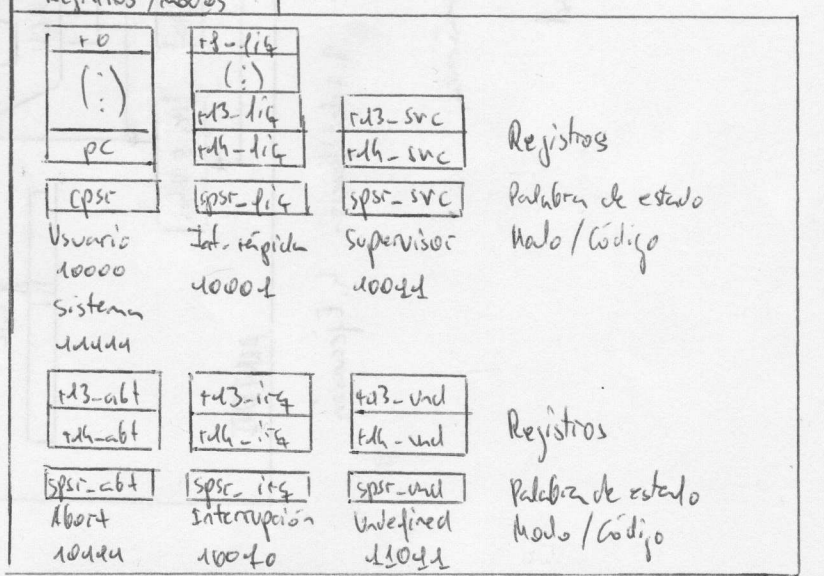
1. Corregir lr (sub lr, lr, #4) y apilar (PUSH {lr}) (PUSH)
2. Cargar palabra de estado del programa interrumpido (MRS rn, spsr) y apilar (PUSH {rn})
3. Cargar nuevo CPSR (cpsr) activando modo IRQ e inhibiendo interrupciones si fuera necesario (msr cpsr_c, #2_11010010)
4. Apilar registros utilizados
5. Transferencia de información
6. Señales de control (si necesario), bajar prioridad (si necesario)
7. ~~Recuperar CPSR anterior (POP {rn})~~ ~~recuperado~~ ⊛ Desactivar interrupciones (msr cpsr_c, #2_11010010)
8. Tratamiento de información (si conveniente)
9. Desapilar registros utilizados, CPSR anterior (POP {rn}) y recuperarlo (msr spsr_fsxc, rn)
10. Eliminar @RSI de VICVectAddr (ldr rn, =VICVectAddr; str in, [rn])
11. Desapilar @retorno (POP {pc}^)

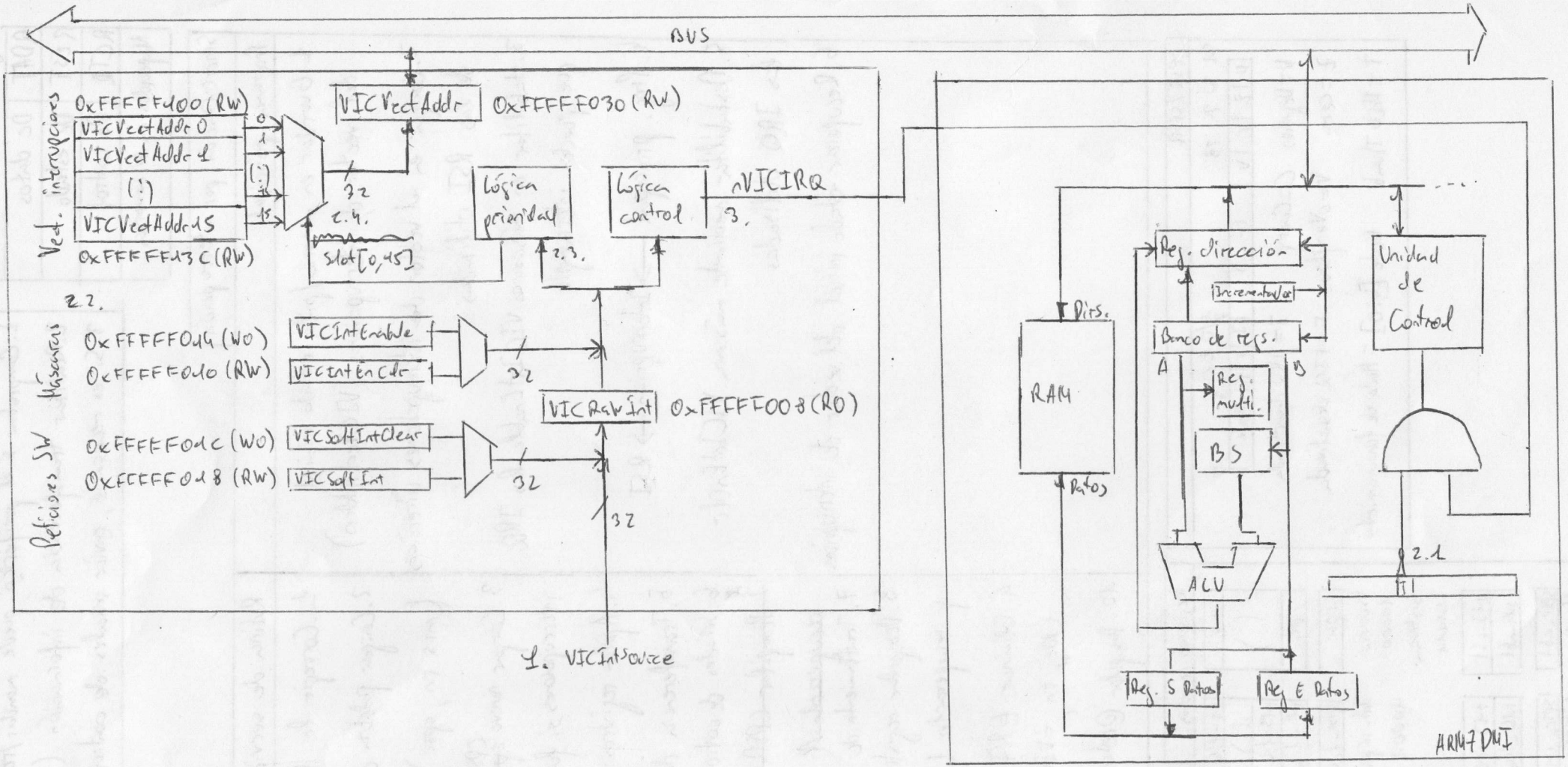
CPSR/SPSR



N = Negativo C = Carry I = IRQ desactivada
Z = Cero V = oVerflow F = FIQ desactivada
T = Modo Thumb Mode [4:0] = Modo de funcionamiento

Registros/modos





Vector de interrupciones	0xFFFFF[...]
100	IRQ 0 Watchdog timer
104	IRQ 1 SW Interrupt
108	IRQ 2 DbgComm Rx
10C	IRQ 3 DbgComm Tx
110	IRQ 4 Timer 0
114	IRQ 5 Timer 1
118	IRQ 6 UART 0
11C	IRQ 7 UART 1
120	IRQ 8 PWM
124	IRQ 9 I2C
128	IRQ 10 SPI
12C	IRQ 11 PLL Lock
130	IRQ 12 RTC
134	IRQ 13 External interrupt 0
138	IRQ 14 External interrupt 1
13C	IRQ 15 External interrupt 2

Vector de excepciones	0x000000[...]
00	Reset
04	Undefined instruction
08	SWI
0C	Prefetch abort
10	Data abort
14	<reservado>
18	IRQ
1C	FIQ

1. Petición 2. Aceptación 3. Identificación 4. Ejecución

2.1. Inhibición global

2.2. Inhibición por máscara

2.3. Detección

2.4. Selección por prioridad