ARM

- En la actualidad, ARM Ltd no hace procesadores, solo los diseña y licencia sus diseños a fabricantes (P. ej: Analog Devices, Atmel, Cirrus Logic, Hyundai, Intel, Oki, Plilips, Samsung, Sharp, Lucent, 3Comp, HP, IBM, Sony, etc.).

Características

- Computadora de 3 direcciones (registros) de 32 bits
- Ciclos de máquina de un solo reloj
- Extensión Thumb
- Excepciones vectorizadas
- Número de transistores: > 74,209 implica bajo consumo.
- Frecuencias de operación: 45 – 133 MHz.
- Bus de 32 bits para datos e instrucciones.
- Elevado rendimiento: hasta 120 MIPS.
- Elevada densidad de código (Diseñado para trabajar en C)

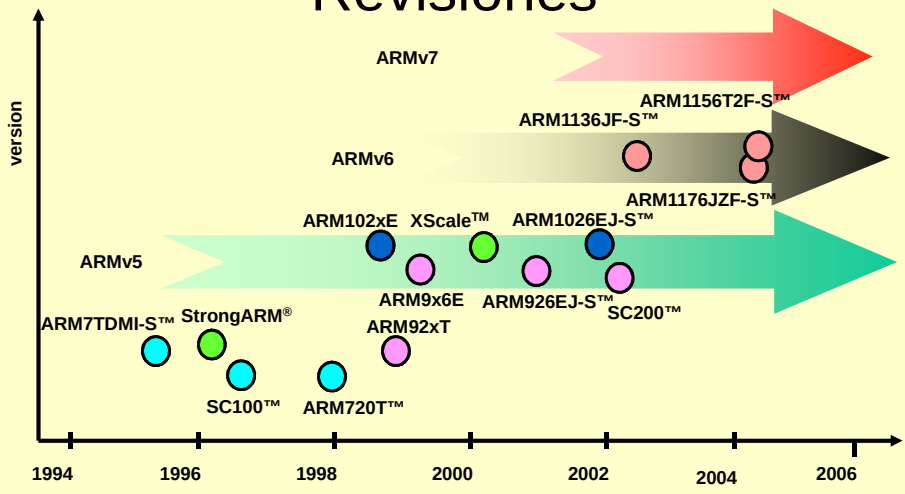
Características

- Se basa en Arquitectura RISC.
- 37 registros de 32 bits (16 disponibles).
- Registros 0 a 7 disponibles en todo momento
- Memoria caché (dependiendo de la aplicación)
- Estructura del bus tipo Von Neuman (ARM7), tipo Harvard (ARM9)

Risc

- Instrucciones de Procesamiento de datos
- Instrucciones de Transferencia de Datos
- Instrucciones de Control de Flujo

Revisiones



Datos e instrucciones

- Tipos de datos de 8/16/32 bits
- Todos las familias de procesadores ARM comparten el mismo conjunto de instrucciones

ARM tiene 7 modos básicos de operación

- **User** : Modo NO privelegiado para la mayoría de las aplicaciones
- **FIQ** : Se ingresa con una interrupción de alta prioridad (fast).
- **IRQ** : Se ingresa con una interrupción de baja prioridad (normal)
- **Supervisor** : Se ingresa en reset y cuando se ejecuta una SWI
- **Abort** : Se emplea para gerenciar violaciones en el acceso a memoria
- **Undef** : Se emplea para gerenciar instrucciones indefinidas
- **System** : Modo privilegiado que emplea los mismos registros que el modo usuario

Tamaño de instrucciones y datos

- ARM es una arquitectura de 32-bits.
  - **Byte** significa 8 bits
  - **Halfword** significa 16 bits
  - **Word** significa 32 bits
- El repertorio de instrucciones
  - 32-bit ARM
  - 16-bit Thumb
- Los que tienen la extensión Jazelle ejecutan código Java

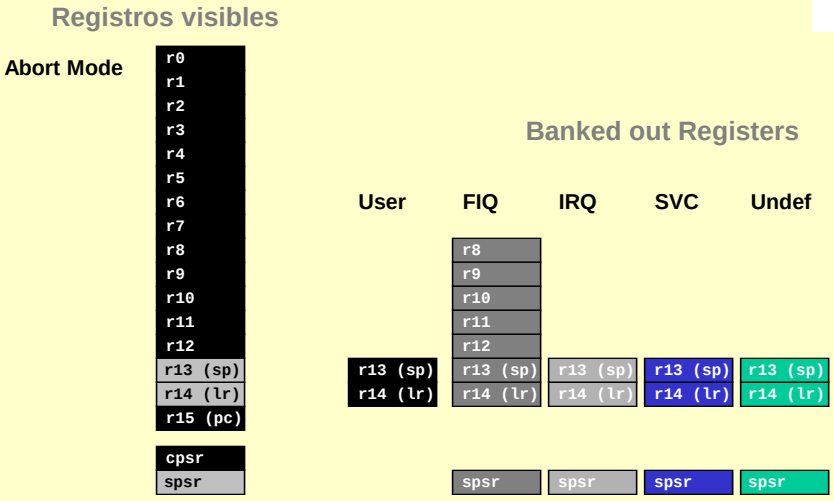
RISC

- Instrucciones conceptualmente simples.
- Transferencias Memoria/Registros exclusivamente LOAD/STORES.
- Las operaciones aritméticas son entre registros.
- Tamaño de instrucciones uniformes.
- Pocos formatos para las instrucciones.
- Conjunto de instrucciones ortogonal: poco o ningún traslape en la funcionalidad de las instrucciones.
- Pocos modos de direccionamiento.

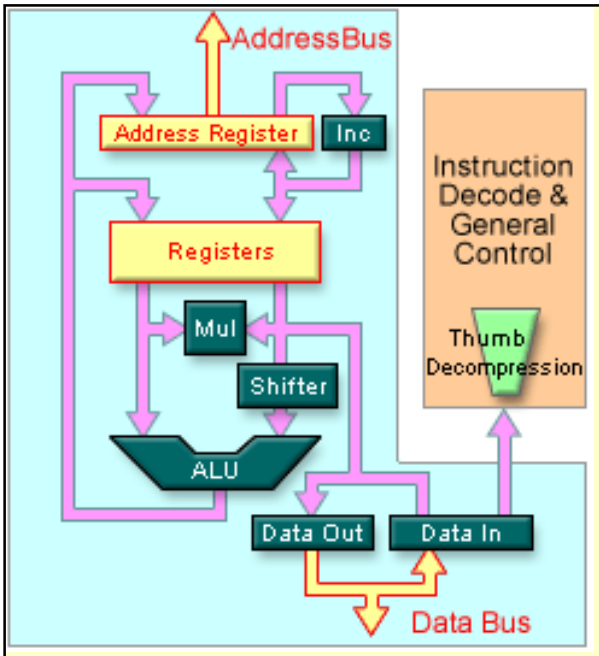
Risc

- Casi todas las instrucciones se ejecutan en un ciclo de reloj.
- Tendencia a tener un gran número de registros.
- Arquitectura RISC predomina en los procesadores de elevado rendimiento.

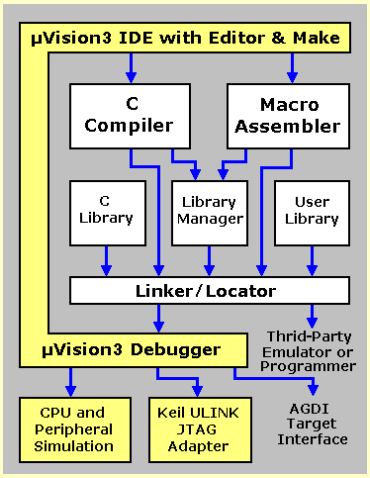
Registros ARM



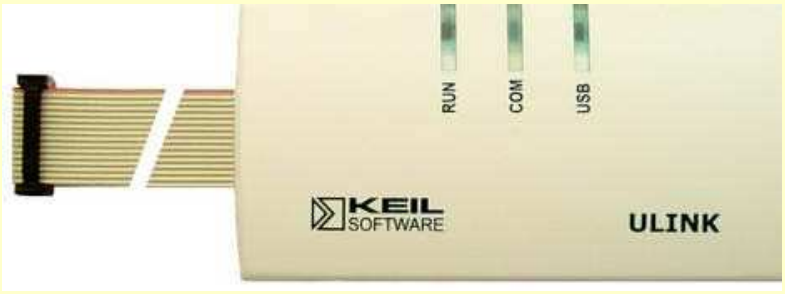
Arquitectura



Ciclo de desarrollo



Herramientas - Jtag



ARM7

- Alimentación: 3.3 V y 5 V.
- Bajo consumo de potencia: 80 mW.
- Tecnología CMOS.
- Extensiones: Thumb, Jazelle.
- Los miembros de ARM7 tienen un coprocesador de interfaz que permite la conexión hasta con 16 coprocesadores más.

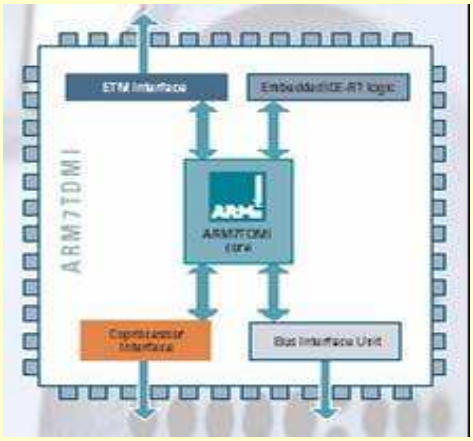
ARM7TDMI

- Es la versión mas utilizada de ARM7.
- ¿ TDMI ?
- T: “Thumb”, soporta esta extensión.
- D: “Debug-interface”.
- M: “Multiplier”, hardware multiplicador.
- I: “Interrupt”, interrupciones veloces.

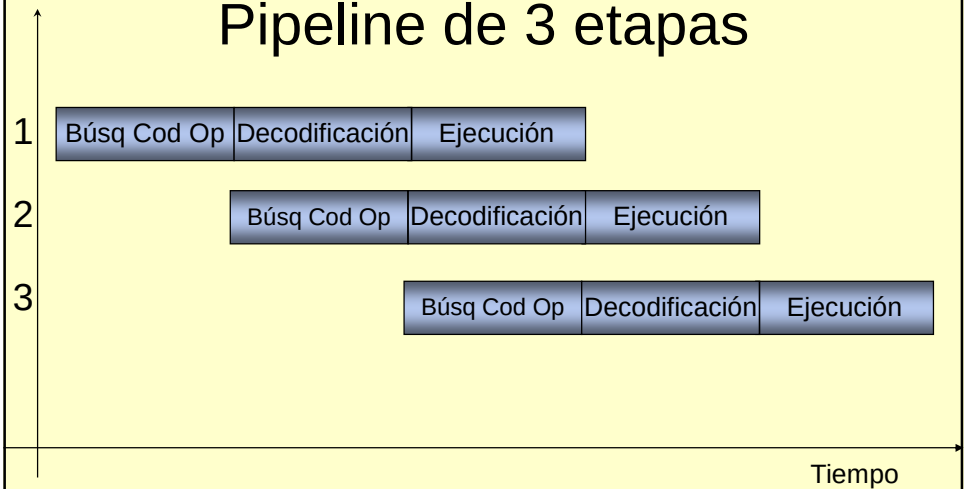
ARM7TDMI

- Arquitectura de bus unificada.
- Lógica de depuración EmbeddedICE-RT.
- Interface ETM (Embedded Trace Macrocell).

ARM7TDMI



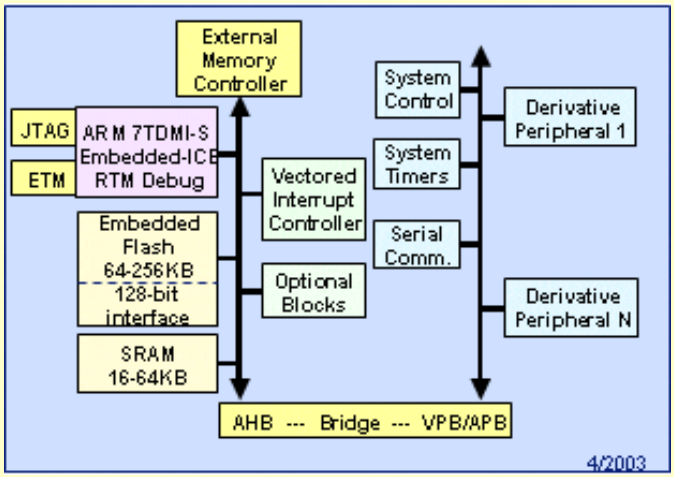
Pipeline de 3 etapas



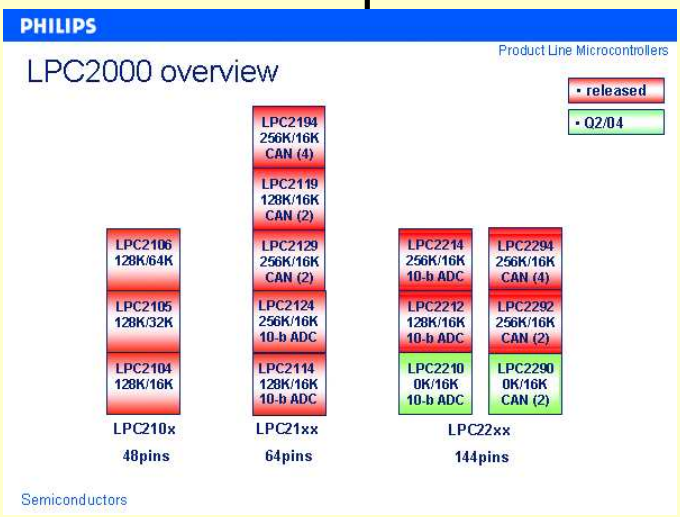
ARM7EJ-S

- Versión sintetizable, incorpora las bondades del ARM7TDMI.
- Soporta ejecución acelerada de Java y operaciones DSP.
- Emplea tecnología ARM Jazelle.

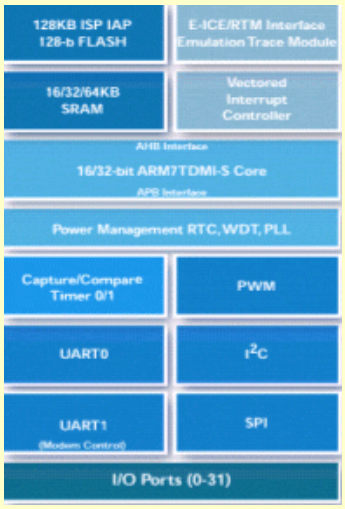
Arquitectura ARM 7



Philips



Características



CAN
ADC
Ethernet
USB

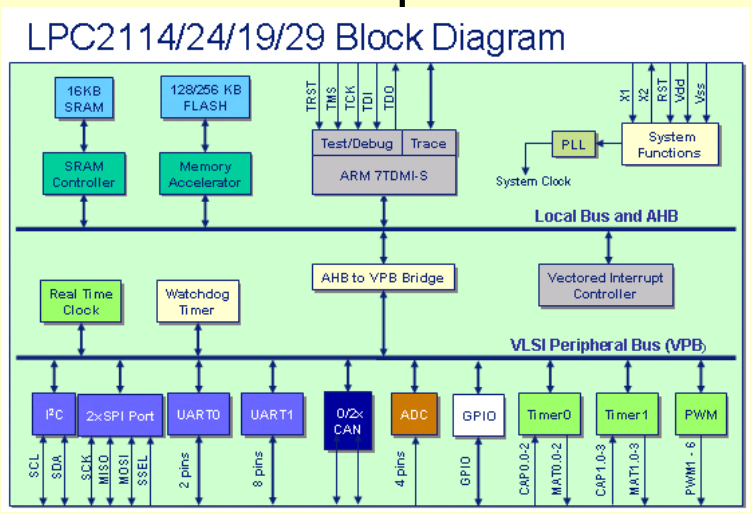
Soporte de depuración
Keil
ICE
JTAG

Philips

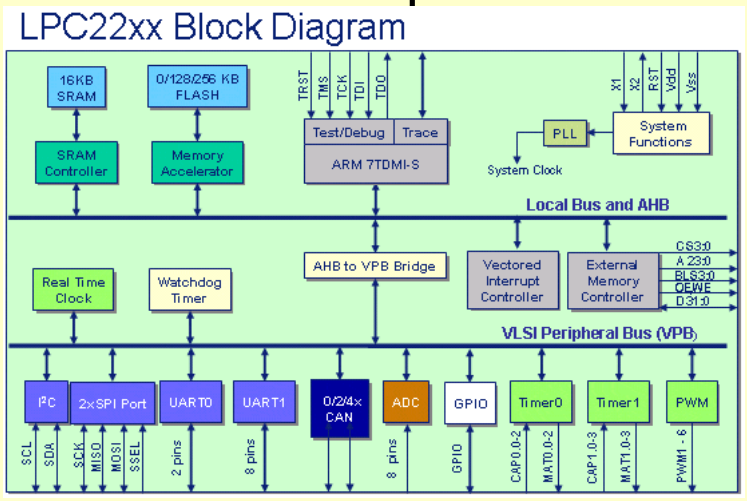
Type number	Package		
	Name	Description	Version
LPC2212FBD144	LQFP144	plastic low-profile quad flat package, 144 leads, body 20 × 20 × 1.4 mm	SOT486-1
LPC2214FBD144	LQFP144	plastic low-profile quad flat package, 144 leads, body 20 × 20 × 1.4 mm	SOT486-1

Type number	Flash memory	RAM	CAN	Temperature range (°C)
LPC2212FBD144	128 kB	16 kB	-	−40 to +85
LPC2214FBD144	256 kB	16 kB	-	−40 to +85

Philips



Philips



LPC22xx

- Núcleo a 1,8 V
- E/S a 3,3 V compatible TTL
- Icc = 35 mA
- Idle = 20 mA
- Sleep = 25 µA

Características

- 3 tipos de Interrupciones
  - FIQ
  - Vectorizadas
  - Interrupciones Generales
  - 32 entradas de interrupción
- PLL
  - Frec in = 10 – 25 MHz
  - Trabaja hasta 60 MHz

Comunicación Serie

- SPI
  - Hasta 2 canales que admiten master-slave
  - Veloc = 1/8 clock
- UART
  - 16550 compatible (con FIFO y flag de fifo llena).
  - Velocidad hasta 1/16 del clock

Comunicación Serie

- I2C
  - Hasta 750 kHz con 7 bits de direccionamiento
  - Bidireccional
  - Sin Maestro (multimaster)
- CAN

Timers

- De 32 bits con 4 registros de captura
- De 32 bits con 4 registros de coincidencia
- Watchdog que para debug no resetea al micro sino que genera excepciones

E/S

- 10 bits – 0 a 3 V y 400 kmuestras/s
- 4 – 8 canales
- PWM de 32 bits con timer específico
- RTC
- 32 bits de E/S

ARM7

- Para sistemas que requieren manejo completo de memoria virtual y espacios de ejecución protegidos.
- Memoria caché de 8K
- MMU: unidad controladora de memoria.
- Para aplicaciones de plataforma abierta como Windows CE, Linux, Palm OS y Symbian OS.

ARM tiene 7 modos básicos de operación

- **User** : Modo NO privelegiado para la mayoría de las aplicaciones
- **FIQ** : Se ingresa con una interrupción de alta prioridad (fast).
- **IRQ** : Se ingresa con una interrupción de baja prioridad (normal)
- **Supervisor** : Se ingresa en reset y cuando se ejecuta una SWI
- **Abort** : Se emplea para gerenciar violaciones en el acceso a memoria
- **Undef** : Se emplea para gerenciar instrucciones indefinidas
- **System** : Modo privilegiado que emplea los mismos registros que el modo usuario

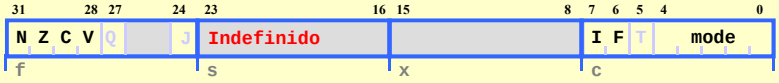
Repertorio de instrucciones

- Comparación
  - CMP r1,r2
  - CMN r1,r2
  - TST r1,r2
  - TEQ r1,r2

REGISTROS ARM7.

- 37 registros de 32 bits, 31 propósito general y 6 registros de estado.
- El número de registros disponibles y su estructura dependen del modo de operación
- 16 registros directamente accesibles (R0 – R15).
- R13: puntero de pila (sp)
- R14: enlace a subrutina (lr)
  - Guarda el valor de R15 cuando se ejecuta una instrucción BL

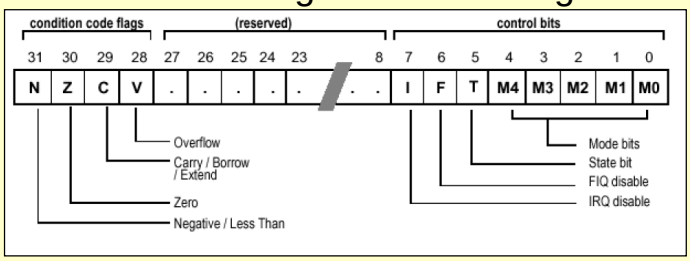
Program Status Register



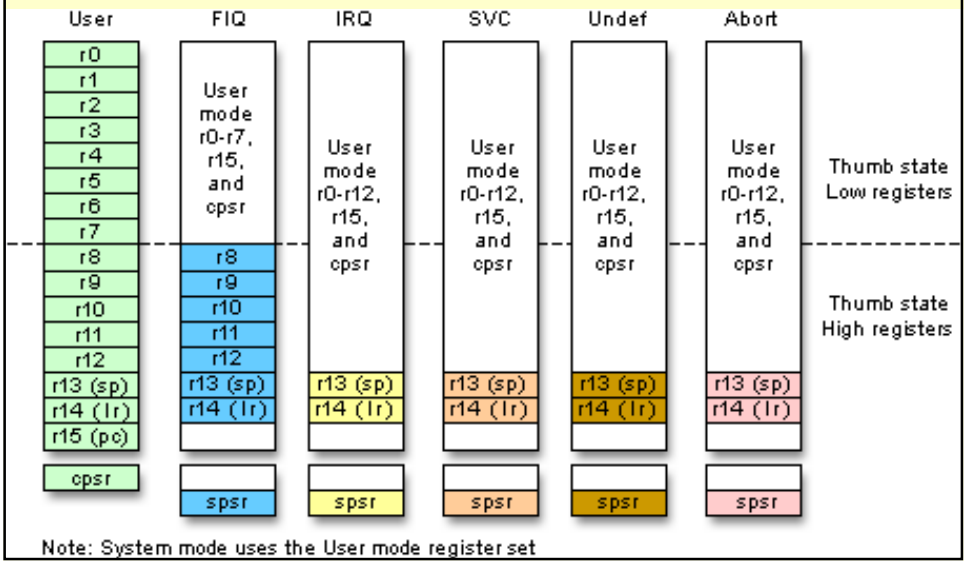
- Condition code flags
  - N = Resultadp Negativo de ALU
  - Z = Ze ro resultado de ALU
  - C = ALU produjo Carry
  - V = ALU produjo oVerflow
- Sticky Overflow flag - Q flag
  - Solo Arquitectura 5TE/J
  - Indica si ocurrió saturación
- J bit
  - Architecture 5TEJ only
  - J = 1: Processor in Jazelle state
- Bits de deshabilitrar Interrupciones.
  - I = 1: Deshabilita IRQ.
  - F = 1: Deshabilita FIQ.
- T Bit
  - Arquitectura xT solamente
  - T = 0: Procesador en estado ARM
  - T = 1: Procesador en estado Thumb
- Bits de Modo
  - Especifican el modo del procesador

REGISTROS ARM7.

- R15: contador de programa
- R16: registro de estado (CPSR, Current Program Status Register)
- SPSR: Saved Program Status Register



Arquitectura



ARM vs. Thumb

- **ARM**
  - Instrucciones fijas de 32bit
  - Instrucciones simples pueden realizar más funciones que una THUMB.
  - Identical execution speed compared to THUMB from Flash/EE (CD > 0) and SRAM (CD >= 0)
  - La tabla de vectores en ARM

- **THUMB**
  - Instrucciones fijas de 16bits
    - Aumentan la densidad de código
    - Aumentan la velocidad de ejecución
  - Repertorio de instrucciones sencillo
  - Conmuta a modo ARM en cada excepción
  - Acceso limitado al banco de registros.

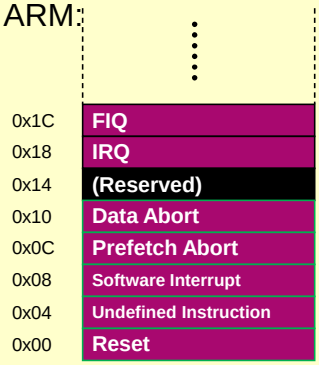
Program Counter (r15)

- When the processor is executing in ARM state:
  - All instructions are 32 bits wide
  - All instructions must be word aligned
  - Therefore the **pc** value is stored in bits [31:2] with bits [1:0] undefined (as instruction cannot be halfword or byte aligned)
- When the processor is executing in Thumb state:
  - All instructions are 16 bits wide
  - All instructions must be halfword aligned
  - Therefore the **pc** value is stored in bits [31:1] with bit [0] undefined (as instruction cannot be byte aligned)
- When the processor is executing in Jazelle state:
  - All instructions are 8 bits wide
  - Processor performs a word access to read 4 instructions at once

Manejo de Excepciones

- Cuando se produce una excepción , el ARM:
  - Copia CPSR en SPSR_<modo>
  - Pone los bits del CPSR apropiados
    - Cambia a modo ARM
    - Cambia a modo excepción
    - Deshabilita interrupciones (Si es apropiado)
  - Almacena la dirección de retorno en LR_<modo>
  - PC = Dirección del vector
- Para retornar, el exception handler necesita:
  - Restaurar CPSR de SPSR_<modo>
  - Restaurar PC de LR_<modo>

  Solo en estadoARM



Vector Table

Vector table can be at **0xFFFF0000** on ARM720T and on ARM9/10 family devices

Excepciones

- Dir Prox Instrucción → LR
  - Si la excepción ocurre en estado ARM, PC+4 ó PC+8
  - Si la excepción ocurre en Thumb PC+2 ó PC+4
- Copia CPSR al correspondiente SPSR
- Fuerza los bits de modo del CPSR (según la excepción
- Fuerza a buscar con el PC la instrucción según el vector

Retorno de excepciones

- PC ← LR

Exception	Saved LR value: ARM	Saved LR value: Thumb	Recommended return instruction	Return point
Reset	-	-	-	After Reset, r14_svc value is Unpredictable.
Data Abort	PC + 8	PC + 8	`SUBS PC, R14_abt, #8`	Returns to aborted instruction.
FIQ	PC + 4	PC + 4	`SUBS PC, R14_fiq, #4`	Returns to interrupted instruction.
IRQ	PC + 4	PC + 4	`SUBS PC, R14_irq, #4`	Returns to interrupted instruction.
Prefetch Abort	PC + 4	PC + 4	`SUBS PC, R14_abt, #4`	Returns to aborted instruction.
Undefined instruction	PC + 4	PC + 2	`MOVS PC, R14_und`	Returns to instruction after Undefined instruction.
SWI instruction	PC + 4	PC + 2	`MOVS PC, R14_svc`	Returns to instruction after SWI instruction.

- CPSR ← SPSR
- Limpia Interrupt Disable

Vectores de excepción

Address	Exception	Mode on entry	I state on entry	F state on entry
`0x00000000`	Reset	Supervisor	Set	Set
`0x00000004`	Undefined Instruction	Undefined	Set	Unchanged
`0x00000008`	SWI	Supervisor	Set	Unchanged
`0x0000000C`	Prefetch Abort	Abort	Set	Unchanged
`0x00000010`	Data Abort	Abort	Set	Unchanged
`0x00000014`	Reserved	Reserved	-	-
`0x00000018`	IRQ	IRQ	Set	Unchanged
`0x0000001C`	FIQ	FIQ	Set	Set

Prioridades de las excepciones

Priority	Exception
Highest	Reset
	Data Abort
	FIQ
	IRQ
	Prefetch Abort
Lowest	Undefined Instruction and SWI

Excepciones

- Dir Prox Instrucción → LR
  - Si la excepción ocurre en estado ARM, PC+4 ó PC+8
  - Si la excepción ocurre en Thumb PC+2 ó PC+4
- Copia CPSR al correspondiente SPSR
- Fuerza los bits de modo del CPSR (según la excepción
- Fuerza a buscar con el PC la instrucción según el vector

Retorno de excepciones

•PC ← LR

Exception	Saved LR value ARM	Saved LR value Thumb	Recommended return instruction	Return point
Reset	-	-	-	After Reset, r14_svc value is Unpredictable.
Data Abort	PC + 8	PC + 8	`SUBS PC, R14_abt, #8`	Returns to aborted instruction.
FIQ	PC + 4	PC + 4	`SUBS PC, R14_fiq, #4`	Returns to interrupted instruction.
IRQ	PC + 4	PC + 4	`SUBS PC, R14_irq, #4`	Returns to interrupted instruction.
Prefetch Abort	PC + 4	PC + 4	`SUBS PC, R14_abt, #4`	Returns to aborted instruction.
Undefined instruction	PC + 4	PC + 2	`MOVS PC, R14_und`	Returns to instruction after Undefined instruction.
SWI instruction	PC + 4	PC + 2	`MOVS PC, R14_svc`	Returns to instruction after SWI instruction.

- CPSR ← SPSR
- Limpia Interrupt Disable

Vectores de excepción

Address	Exception	Mode on entry	I state on entry	F state on entry
`0x00000000`	Reset	Supervisor	Set	Set
`0x00000004`	Undefined Instruction	Undefined	Set	Unchanged
`0x00000008`	SWI	Supervisor	Set	Unchanged
`0x0000000C`	Prefetch Abort	Abort	Set	Unchanged
`0x00000010`	Data Abort	Abort	Set	Unchanged
`0x00000014`	Reserved	Reserved	-	-
`0x00000018`	IRQ	IRQ	Set	Unchanged
`0x0000001C`	FIQ	FIQ	Set	Set

Prioridades de las excepciones

Priority	Exception
Highest	Reset
	Data Abort
	FIQ
	IRQ
	Prefetch Abort
Lowest	Undefined Instruction and SWI

Instrucciones ARM7

- Instrucciones de 32 bits en el modo de operación nativo ARM: longitud de palabra de 32 bits
- Todas las instrucciones son condicionales
- En ejecución normal (incondicional), la condición AL (always) se establece en el campo condición
- En operaciones condicionales se selecciona una de las 14 condiciones
- 36 formatos de instrucciones

Instrucciones ARM7

- 11 tipos básicos de instrucciones.
- Dos de estos tipos emplean la ALU, el desplazador en barril y el multiplicador para ejecutar operaciones a alta velocidad sobre datos en los registros.
- Ejemplos: AND, EOR, SUB, RSB, ADD, ADC, SBC, RSC, TST, TEQ, CMP, CMN, ORR, MOV, BIC, MVN, (Multiplicaciones) MUL, MLA, MULL, MLAL

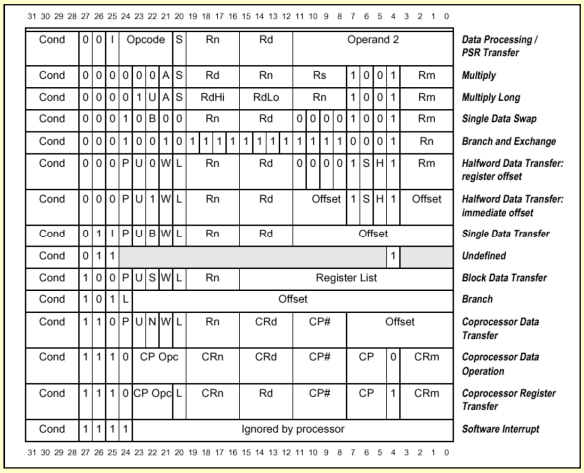
Instrucciones de Procesamiento de Datos

- Consiste de:
  - Aritmeticas: **ADD ADC SUB SBC RSB RSC**
  - Logicas: **AND ORR EOR BIC**
  - Comparaciones: **CMP CMN TST TEQ**
  - Movimeinto Datos: **MOV MVN**
- Estas instrucciones operan sobre registros y NO en memoria.
- Sintaxis:

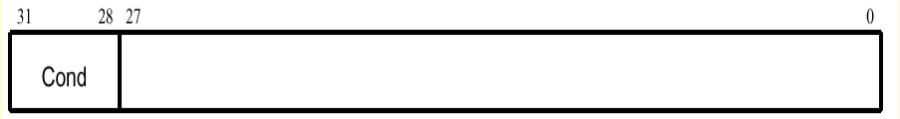
  <Operación>{<cond>}{S} Rd, Rn, Operand2

  - Comparaciones sòlo afectan flags – no especifican Rd
  - Movimiento de datos no especifican Rn
- El segundo operando se envía a la ALU a través del desplazador en barril.

Repertorio de instrucciones



CODIGOS DE CONDICION



Condition field

0000 = EQ - Z set (equal)
0001 = NE - Z clear (not equal)
0010 = CS - C set (unsigned higher or same)
0011 = CC - C clear (unsigned lower)
0100 = MI - N set (negative)
0101 = PL - N clear (positive or zero)
0110 = VS - V set (overflow)
0111 = VC - V clear (no overflow)
1000 = HI - C set and Z clear (unsigned higher)
1001 = LS - C clear or Z set (unsigned lower or same)
1010 = GE - N set and V set, or N clear and V clear (greater or equal)
1011 = LT - N set and V clear, or N clear and V set (less than)
1100 = GT - Z clear, and either N set and V set, or N clear and V clear (greater than)
1101 = LE - Z set, or N set and V clear, or N clear and V set (less than or equal)
1110 = AL - always
1111 = NV - never

Ejemplos condicionales

Fuente C

```
if (r0 == 0)
{
  r1 = r1 + 1;
}
else
{
  r2 = r2 + 1;
}
```

ARM instructions

Incondicional

```
CMP r0, #0
  BNE else
  ADD r1, r1, #1
  B end
else
  ADD r2, r2, #1
end
```

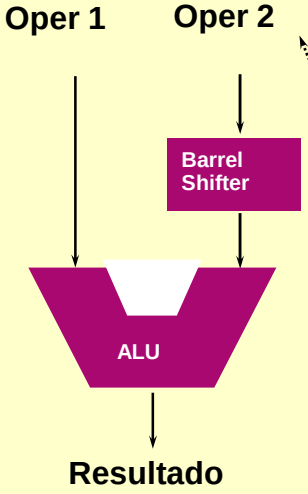
- 5 instrucciones
- 5 palabras
- 5 o 6 ciclos

Condicional

```
CMP   r0, #0
ADDEQ r1, r1, #1
ADDNE r2, r2, #1
...
```

- 3 instrucciones
- 3 palabras
- 3 ciclos

Barrel Shifter: El 2º Operando



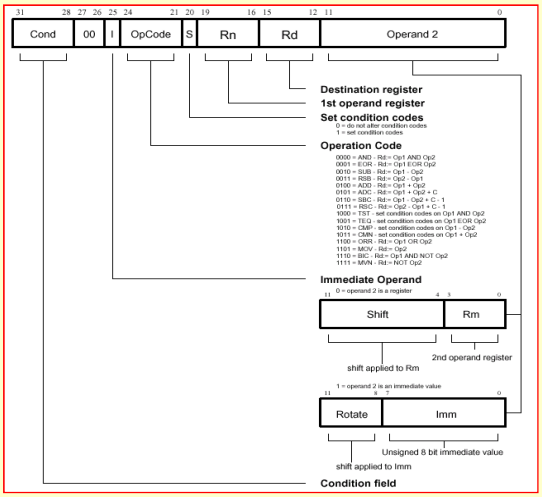
Register, optionally with shift operation

- Shift value can be either be:
  - 5 bit unsigned integer
  - Specified in bottom byte of another register.
- Used for multiplication by constant

Immediate value

- 8 bit number, with a range of 0-255.
  - Rotated right through even number of positions
- Allows increased range of 32-bit constants to be loaded directly into registers

Procesamiento de Datos



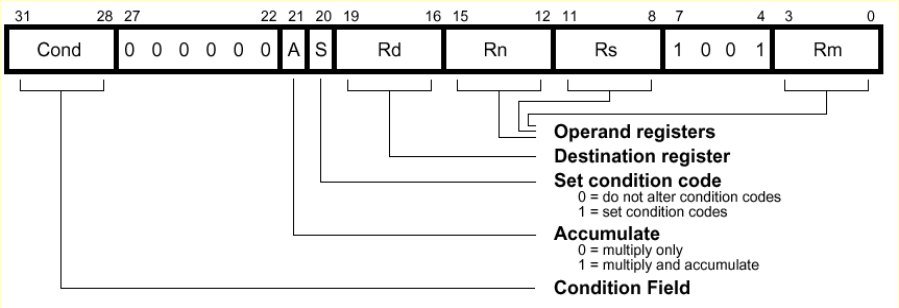
Aritméticas

- ADD r1, r2, r3 ; r1 = r2 + r3
- ADC r1, r2, r3 ; r1 = r2 + r3 + C
- SUB r1, r2, r3 ; r1 = r2 - r3
- SUBC r1, r2, r3 ; r1 = r2 - r3 + C - 1
- RSB r1, r2, r3 ; r1 = r3 - r2 (inversa)
- RSC r1, r2, r3 ; r1 = r3 - r2 + C - 1

Repertorio de instrucciones

- Aritméticas
  - ADD r3,r2,#1
  - ADD r3,r2,r1, lsl #3 (lsr, asl, asr, ror, rrx)
  - ADD r5,r5,r3, LSL r2
  - MUL r4,r3,r2
  - MLA r4,r3,r2,r1 ;r4:=(r3 x r2 + r1)
  - RSB r0,r0,r0, LSL #3
    - » ; Multiplicar por 7

MULTIPLICACIONES



Multiplicar y Dividir

- 32-bit versions on an ARM7TDMI will execute in 2 - 5 cycles

```
– MUL r0, r1, r2              ; r0 = r1 * r2
– MLA r0, r1, r2, r3          ; r0 = (r1 * r2) + r3
```

- 64-bit multiply instructions offer both signed and unsigned versions
  - For these instruction there are 2 destination registers

```
– [U|S]MULL r4, r5, r2, r3    ; r5:r4 = r2 * r3
– [U|S]MLAL r4, r5, r2, r3    ; r5:r4 = (r2 * r3) + r5:r4
```

- Most ARM cores do not offer integer divide instructions
  - Division operations will be performed by C library routines or inline shifts

Repertorio de instrucciones

- Lógicas
  - AND r0,r1,r2
  - ORR r0,r1,r2
  - EOR r0,r1,r2 ; r0:= r1 xor r2
  - BIC r0,r1,r2 ; r0:= r1 and not r2
  - AND r8,r7,#&ff

Solo afectan los Flags

```
CMP     r1, r2      ; cc por r1 - r2
CMN     r1, r2      ; cc por r1 + r2
TST     r1, r2      ; cc por r1 and r2
TEQ     r1, r2      ; cc por r1 xor r2
```

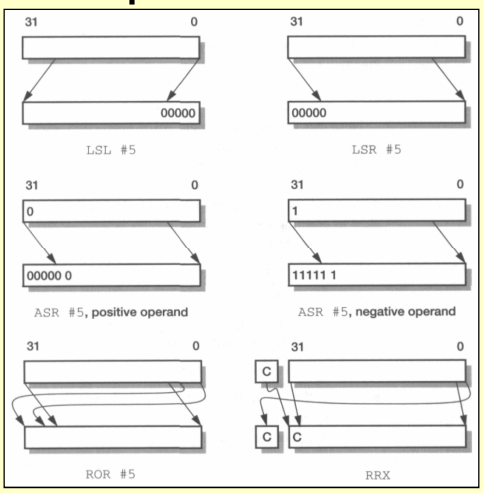

Inmediatas

- ADD r3, r3,#1 ; r3 := r3 + 1
- AND r8, r7,#&ff ; r8 := r7[7:0]

Desplazamientos

- ADD r3, r2, r1, LSL #3 ; r3 := r2 + 8 x r1
- ADD r5, r5, r3, LSL r2 ; r5 : = r5 + r3 x 2^{r2}

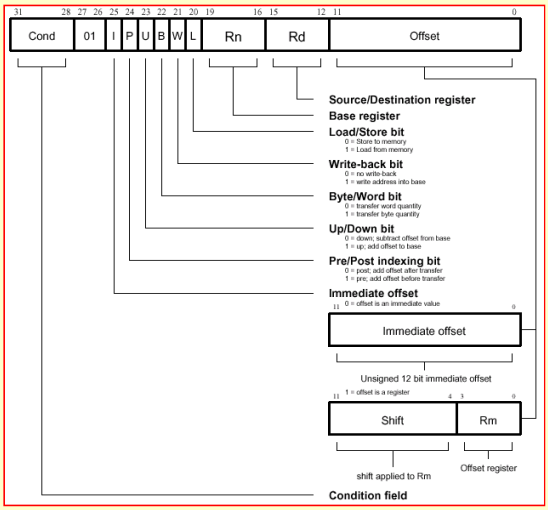
Desplazamientos



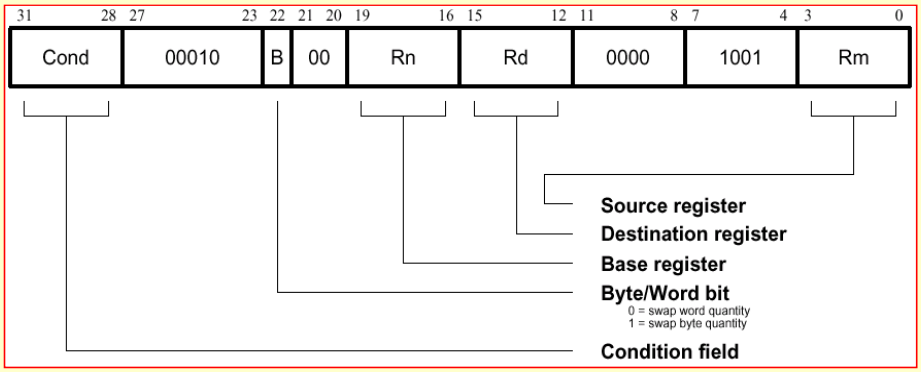
Modificando el CCR

```
ADDS    r2, r2, r0 ; 32-bit carry out -> C..
ADC     r3, r3, r1 ; .. and added into high word
```


Transferencias de Datos



INTERCAMBIO SIMPLE DE DATOS



Movimiento de Datos

- Movimiento
  - MOV r0,r2
  - MVN r0,r2 ; r0:= not r2
  - LDR r0,[r1]
  - STR r0,[r1]
  - LDR r0,[r1], #4
  - LDR r0,[r1,#4]!

Pre y post indexado

```
LDR r0,[r1,#4]      ; r0 := men32[r1+ 4]
LDR r0,[r1,#4]!     ; r0 := mem32[r1+ 4] ; r1 := r1 + 4
```


Carga de constantes de 32 bits

– **Se Genera una Seudoinstrucción**

– **LDR r0,=0xFF**	=>	**MOV r0,#0xFF**
– **LDR r0,=0x55555555**	=>	**LDR r0,[PC,#Imm12]**
		…
		…
		DCD 0x55555555

Load/Store

Arreglo de 25 palabras.

El compilador asocia y con r1. La base del array esta apuntado por r2. Implementar en assembler

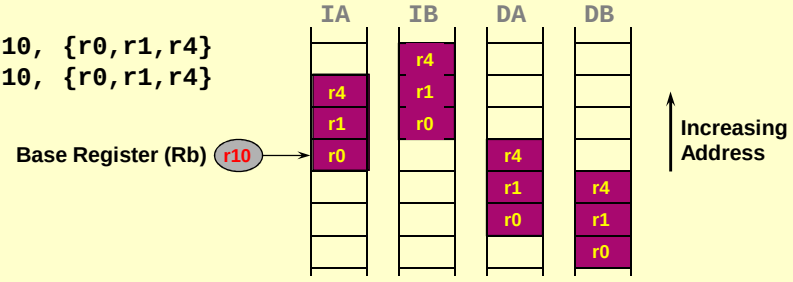
array[10] = array[5] + y;

```
LDR   r3, [r2, #5]   ; r3 = array[5]
ADD   r3, r3, r1     ; r3 = array[5] + y
STR   r3, [r2, #10]  ; array[5] + y = array[10]
```

Multiples Load y Store

- 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}
```



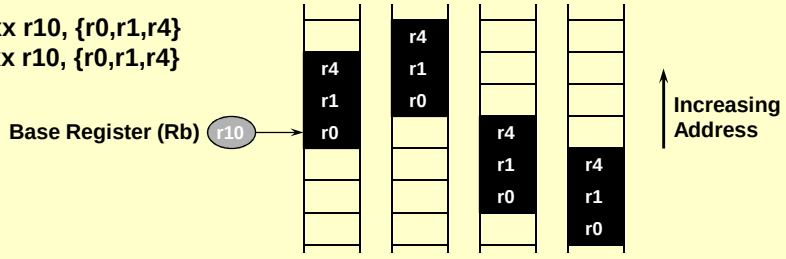
Copia de tablas

```
COPY    ADR     r1, TABLE1          ; r1 points to TABLE1
        ADR     r2, TABLE2          ; r2 points to TABLE2
LOOP    LDR     r0, [r1]            ; get TABLE1 1st word
        STR     r0, [r2]            ; copy into TABLE2
        ADD     r1, r1, #4          ; step r1 on 1 word
        ADD     r2, r2, #4          ; step r2 on 1 word
        ???                         ; if more go back to LOOP
        ..
TABLE1                              ; < source of data >
        ..
TABLE2                              ; < destination >
```


Load/Store

- **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}



Tablas y registros

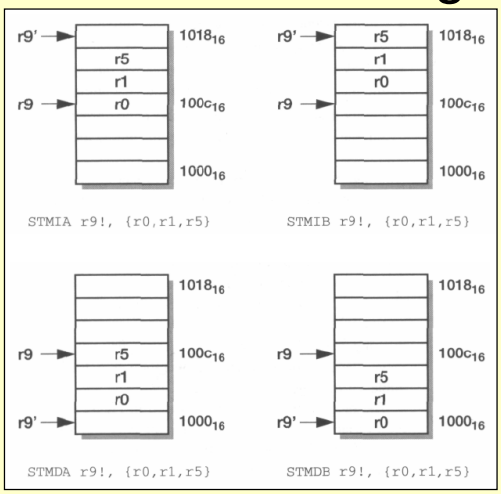
```
LDMIA   r1, {r0,r2,r5}      ; r0 := mem32[r1]
                            ; r2 := mem32[r1 + 4]
                            ; r5 := mem32[r1 + 8]
```


Uso de la Pila

```
LDMIA  r0!, {r2-r9}
STMIA  r1,  {r2-r9}
```

```
STMFD   r13!,   {r2-r9}
LDMIA   r0!,    {r2-r9}
STMIA   r1,     {r2-r9}
LDMFD   r13!,   {r2-r9}
```

Transferencia de Registros



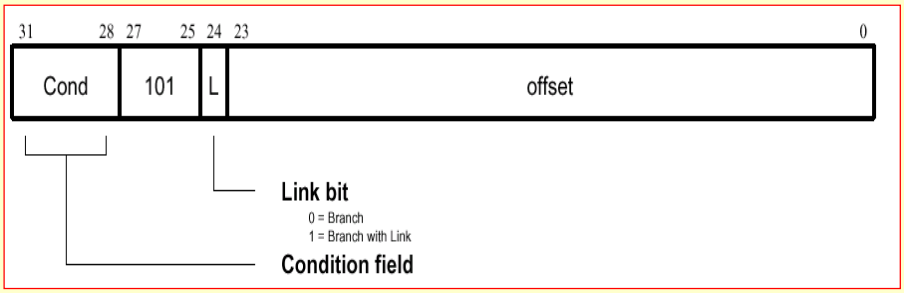
Copia de bloques y stack

		Ascending		Descending	
		Full	**Empty**	**Full**	**Empty**
Increment	**Before**	STMIB STMFA			LDMIB LDMED
	After		STMIA STMEA	LDMIA LDMFD	
Decrement	**Before**		LDMDB LDMEA	STMDB STMFD	
	After	LDMDA LDMFA			STMDA STMED

Instrucciones ARM7

- Instrucciones de salto (Branching): BX, B, BL
- BX: Branch and eXchange, salto con cambio de conjunto de instrucciones ARM < -- > Thumb
- B: salto con desplazamiento de 24 bits con signo
- BL: enlace (link) PC -> R14
- Instrucciones de transferencia de datos: LDR, STR, LDRH, STRH, LDRSB, LDRSH, LDM, STM, SWP.

INSTRUCCIONES DE SALTO: B, BL



Control de flujo

Branch	Interpretation	Normal uses
B BAL	Unconditional Always	Always take this branch Always take this branch
BEQ	Equal	Comparison equal or zero result
BNE	Not equal	Comparison not equal or non-zero result
BPL	Plus	Result positive or zero
BMI	Minus	Result minus or negative
BCC BLO	Carry clear Lower	Arithmetic operation did not give carry-out Unsigned comparison gave lower
BCS BHS	Carry set Higher or same	Arithmetic operation gave carry-out Unsigned comparison gave higher or same
BVC	Overflow clear	Signed integer operation; no overflow occurred
BVS	Overflow set	Signed integer operation; overflow occurred
BGT	Greater than	Signed integer comparison gave greater than
BGE	Greater or equal	Signed integer comparison gave greater or equal
BLT	Less than	Signed integer comparison gave less than
BLE	Less or equal	Signed integer comparison gave less than or equal
BHI	Higher	Unsigned comparison gave higher
BLS	Lower or same	Unsigned comparison gave lower or same

Ejemplo

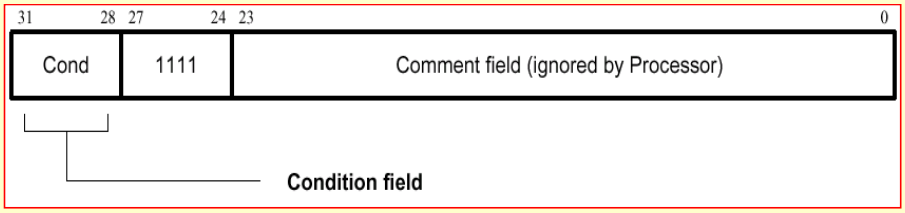
```
   CMP    r0,#5
   BEQ    SALTO
   ADD    r1,r1,r0
   SUB    r1,r1,r2        ; r0:=r1 + r0 – r2
Salto:

   CMP        r0,#5
   ADDNE      r1,r1,r0
   SUBNE      r1,r1,r2     ;r0:=r1 + r0 – r2
```

Subrutinas

```
  BL      subru
…
Subru: ….
  mov     pc,r14
```


INTERRUPCIONES DE SOFTWARE



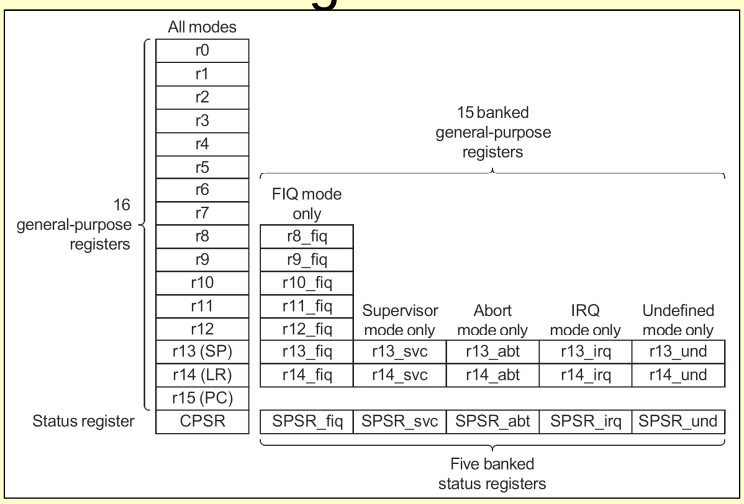
Instrucciones ARM7

- Instrucciones de excepciones: SWI, SoftWare Interrupt.
- Instrucciones del Coprocesador: CDP, LDC, STC, MRC, MCR.
- ARM no ejecuta estas instrucciones pero deja al coprocesador la manipulación de ellas.

Modos

Modo	Descripción
User	Para ejecución normal de aplicación
FIQ	Transferencia de datos de alta velocidad
IRQ	Manejo general de Interrupciones
Supervisor	Modo protegido para el sistema operativo
Abort	Para implementar memoria virtual o protección de memoria
Abort	Emulación por software de coprocesadores
System	Para correr tareas privilegiadas del sistema operativo

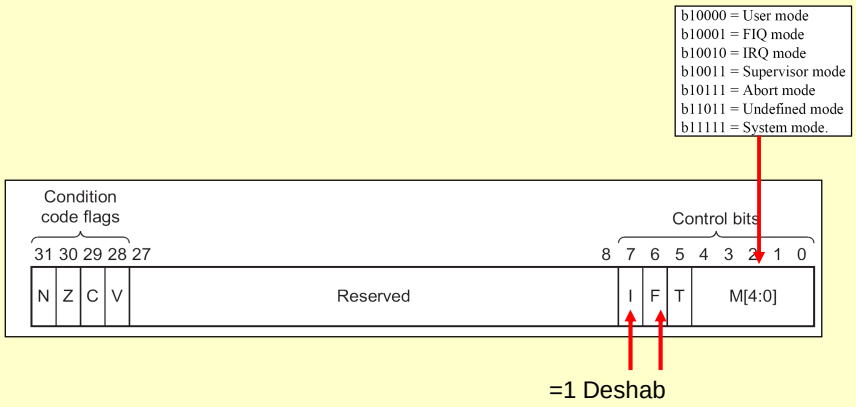
Registros



Registros

Modo	Indentificador de modo de bancos de registros
User	_usrb
Fast interrupt	_fiq
Interrupt	_irq
Supervisor	_svc
Abort	_abt
System	_usr
Undefined	_und

Program Status Registers



TIPOS DE DATOS.

- Emplea un bus de datos y un bus de direcciones de 32 bits.
- Tipos de datos que soporta el procesador: bytes (8 bits) y palabras (32 bits).
- Las instrucciones son exclusivamente palabras.
- Las transferencias pueden ser bytes o palabras.

TIPOS DE CICLOS DE TRANSFERENCIA A MEMORIA.

- Transferencia de registro del coprocesador: ARM7 desea usar el bus de datos para comuicarse con un coprocesador pero no requiere ninguna acción del sistema de memoria.
- Estos 4 tipos de ciclos son visibles a la memoria a través de las lineas de control nMREQ y SEQ.

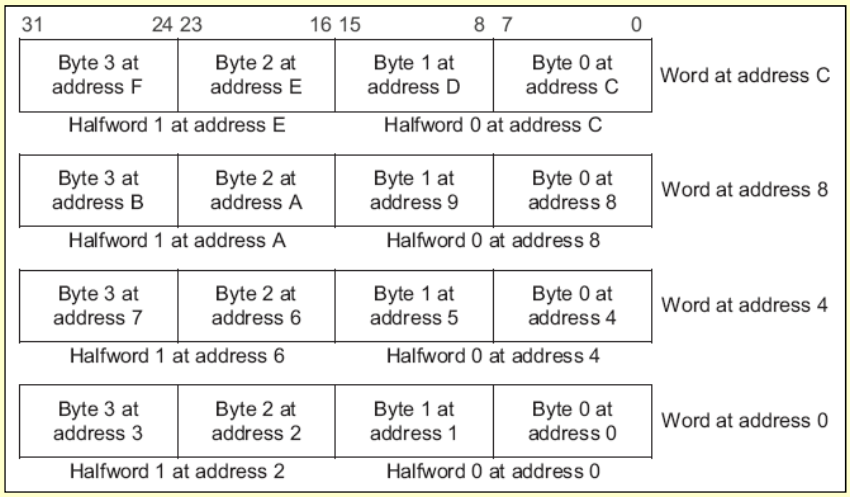
COPROCESADORES.

- La funcionalidad de ARM7 se incrementa agregando hasta 16 coprocesadores externos.
- nCPI, CPA (coporocesador ausente), CPB (coporocesador ocupado), señales que controlan la interfaz con los coprocesadores; CPA y CPB siempre activas excepto cuando el coprocesador está en “handshaking”.
- Cada coprocesador tiene un único número dentro de un sistema y recibe una copia de la instrucción.
- Inspeccionando el campo CP#, cada coprocesador determina si la instrucción le corresponde.

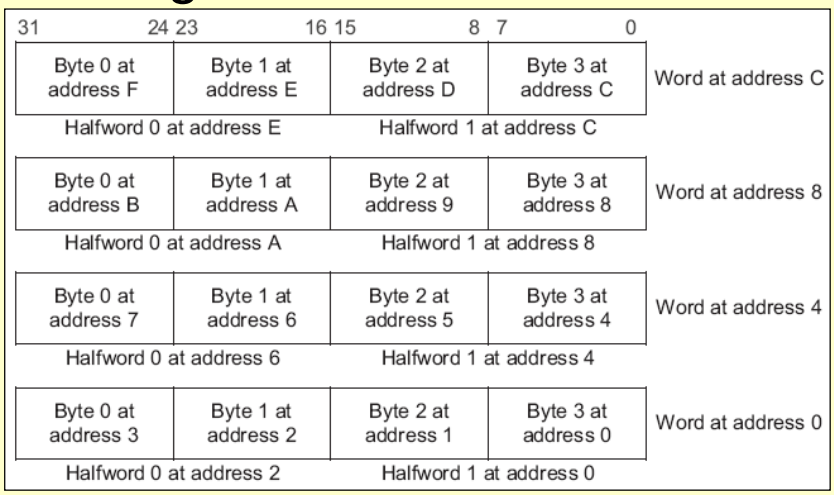
COPROCESADORES.

- Cada coprocesador tiene hasta 16 registros privados.
- Emplean aquitectura LOAD/STORE.
- Acoplados al bus de memoria ARM.
- “Observan” el tráfico de instrucciones del bus.

Little endian data format



Big Endian data format



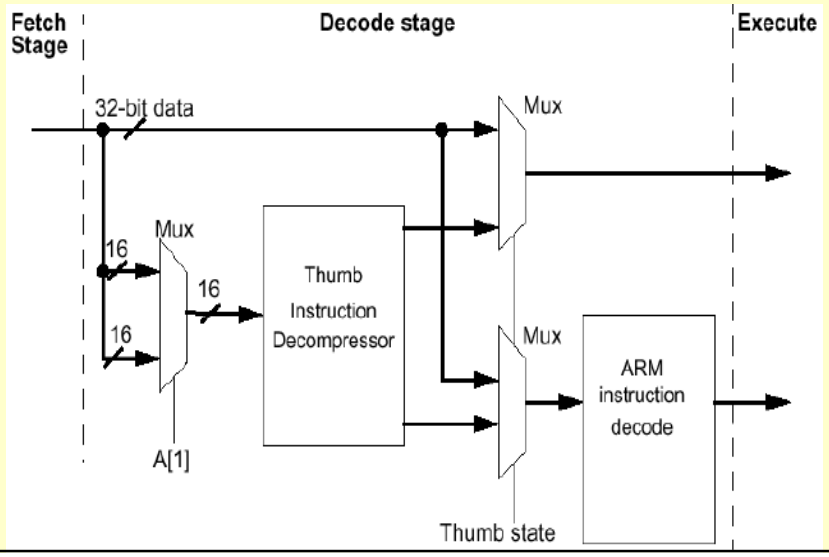
ARM7

- Para sistemas que requieren manejo completo de memoria virtual y espacios de ejecución protegidos.
- Memoria caché de 8K
- MMU: unidad controladora de memoria.
- Para aplicaciones de plataforma abierta como Windows CE, Linux, Palm OS y Symbian OS.

EXTENSION “THUMB”

- Representación comprimida de 16 bits del ISA de ARM.
- Con el objetivo de incrementar: densidad de código y rendimiento, en algunos casos.
- No es una arquitectura completa, es un subconjunto del ISA de ARM.

DECOMPRESOR THUMB.



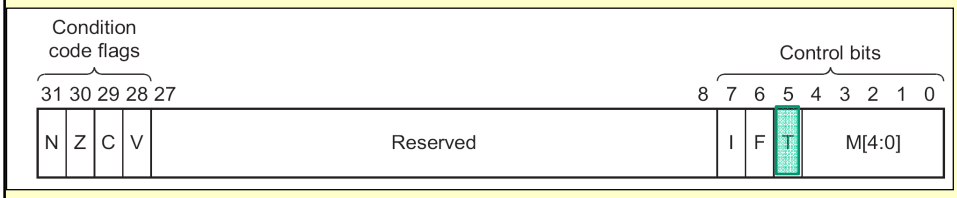
EXTENSION “THUMB”.

- Thumb comprime las instrucciones de ARM a una longitud de palabra de 16 bits, ahorrándose del 35 al 40 % en memoria, comparado con un conjunto de instrucciones de 32 bits.
- Los registros se mantienen como de 32 bits, pero solo la mitad de ellos es empleado.
- Se emplea un decodificador de instrucciones Thumb en el pipeline.
- Solo los registros de la parte mas baja se emplean, los registros de la parte alta se establecen a cero.

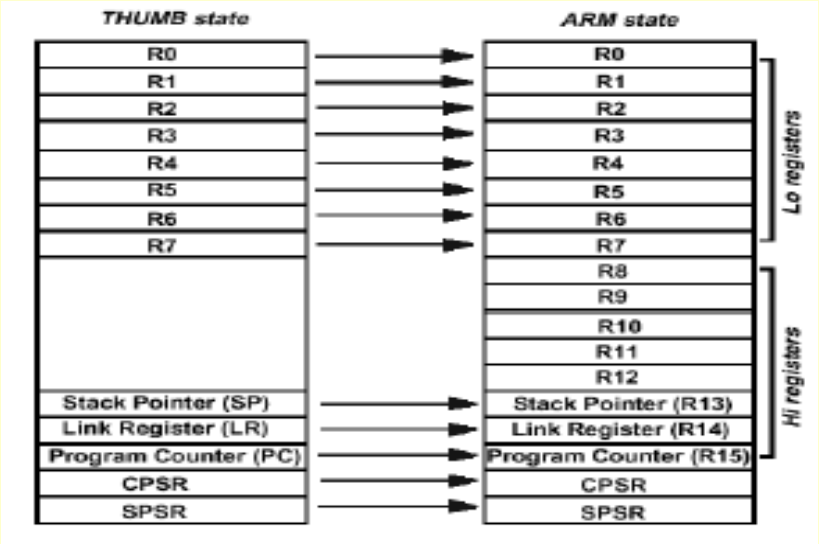
CAMBIANDO DE MODO: ARM < -- > THUMB

- Si solo se emplea el bus de datos de 16 bits, la velocidad de ejecución de código ARM nativo disminuye significativamente.
- Para retornar de Thumb a ARM nativo, se reestablece T-flag y se ejecuta BX a la dirección deseada.

Program Status Registers



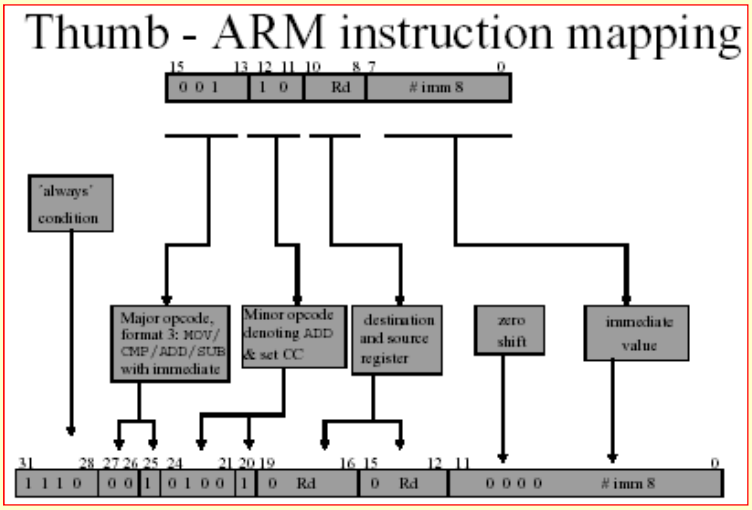
ESTADO DE REGISTROS THUMB.



CONJUNTO DE INSTRUCCIONES THUMB.

- La longitud de palabra se reduce a 16 bits.
- Las instrucciones siguen su propia sintaxis, pero cada instrucción tiene su contraparte en ARM nativo.
- Debido a la reducción de bits, se pierde cierta funcionalidad.
- Existen 19 formatos diferentes de instrucción Thumb.

INSTRUCCIONES THUMB – ARM



SUMARIO: CONJUNTO DE INSTRUCCIONES THUMB.

Mnemonic	Operation	Mnemonic	Operation
MOV	Move	MVN	Move Not
ADD	Add	ADC	Add with Carry
SUB	Subtract	SBC	Subtract with Carry
RSB	Reverse Subtract	RSC	Reverse Subtract with Carry
CMP	Compare	CMN	Compare Negated
TST	Test	NEG	Negate
AND	Logical AND	BIC	Bit Clear
EOR	Logical Exclusive OR	ORR	Logical (inclusive) OR
LSL	Logical Shift Left	LSR	Logical Shift Right
ASR	Arithmetic Shift Right	ROR	Rotate Right
MUL	Multiply	BKPT	Breakpoint
B	Unconditional Branch	Bcc	Conditional Branch
BL	Branch and Link	BLX	Branch and Link and Exchange
BX	Branch and Exchange	SWI	Software Interrupt
LDR	Load Word	STR	Store Word
LDRH	Load Halfword	STRH	Store Halfword
LDRB	Load Byte	STRB	Store Byte
LDRSH	Load Signed Halfword	LDRSB	Load Signed Byte
LDMIA	Load Multiple	STMIA	Store Multiple
PUSH	Push Registers to stack	POP	Pop Registers from stack

APLICACIONES DE THUMB.

- Para optimizar el costo y el consumo de potencia.
- Para rutinas de control largas y no críticas.

CONJUNTO DE INSTRUCCIONES THUMB.

- La longitud de palabra se reduce a 16 bits.
- Las instrucciones siguen su propia sintaxis, pero cada instrucción tiene su contraparte en ARM nativo.
- Debido a la reducción de bits, se pierde cierta funcionalidad.
- Existen 19 formatos diferentes de instrucción Thumb.

EXTENSION JAZELLE.

- El uso de aplicaciones basadas en tecnología Java se está incrementando.
- ARM implementó su extensión Jazelle.
- Las soluciones típicas en Java descansan en implementaciones de software o hardware.
- Implementaciones de software requieren el uso de más componentes de memoria => mayor consumo de potencia.
- Soluciones de hardware requieren acoplar dispositivos al procesador => costo adicional en silicio.

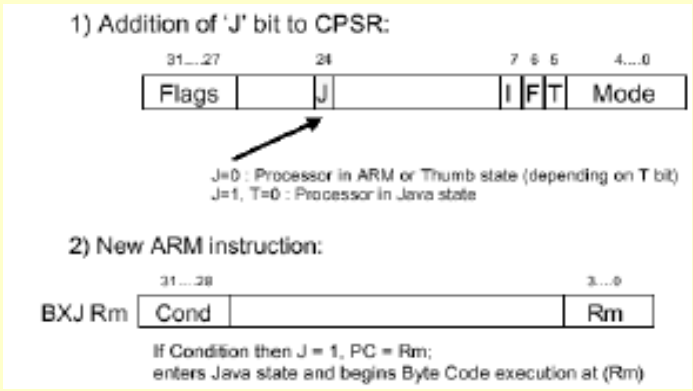
EXTENSION JAZELLE.

- La extensión Jazelle implementa ambas soluciones sin requerir hardware o memoria adicional.
- Mejora el rendimiento hasta 8 veces de soluciones basadas en software y hasta el doble de soluciones basadas en co-procesadores => poder correr aplicaciones Java complejas en sistemas basados en nucleos ARM de bajo consumo de potencia.
- Permite a los desarrolladores correr en un solo procesador aplicaciones Java, junto con los sistemas operativos establecidos y código de aplicación.

EXTENSION JAZELLE.

- En la práctica, es agregar un tercer conjunto de instrucciones al núcleo del procesador ARM (Jazelle, ARM y Thumb).
- Se crea un nuevo estado del procesador: estado Java, el procesador se comporta como un procesador Java.
- El cambio entre estados: Java y ARM/Thumb, es por medio del control del sistema operativo.
- 140 intrucciones son ejecutadas directamente en hardware, el resto (94) se ejecutan por simulación por múltiples instrucciones ARM.

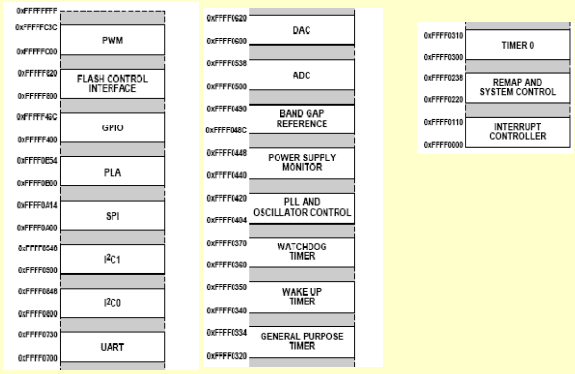
NUEVO MODO DE OPERACIÓN JAVA.



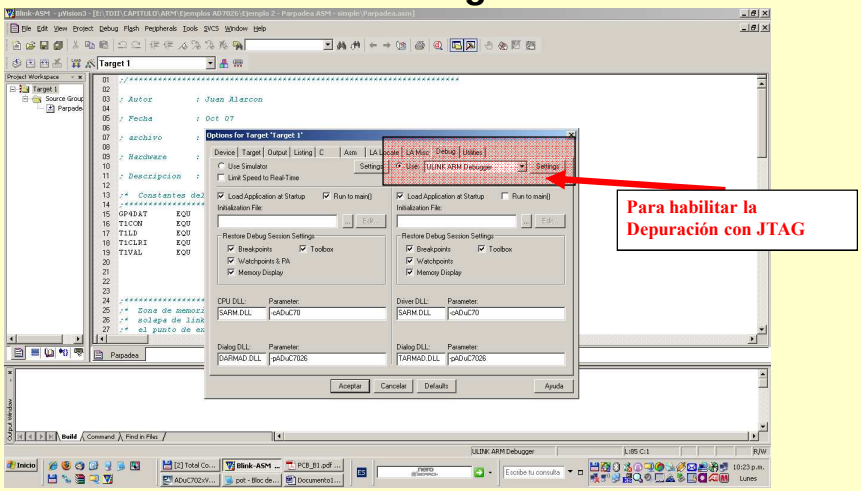
Program Counter (r15)

- When the processor is executing in ARM state:
  - All instructions are 32 bits wide
  - All instructions must be word aligned
  - Therefore the pc value is stored in bits [31:2] with bits [1:0] undefined (as instruction cannot be halfword or byte aligned)
- When the processor is executing in Thumb state:
  - All instructions are 16 bits wide
  - All instructions must be halfword aligned
  - Therefore the pc value is stored in bits [31:1] with bit [0] undefined (as instruction cannot be byte aligned)
- When the processor is executing in Jazelle state:
  - **All instructions are 8 bits wide**
  - Processor performs a word access to read 4 instructions at once

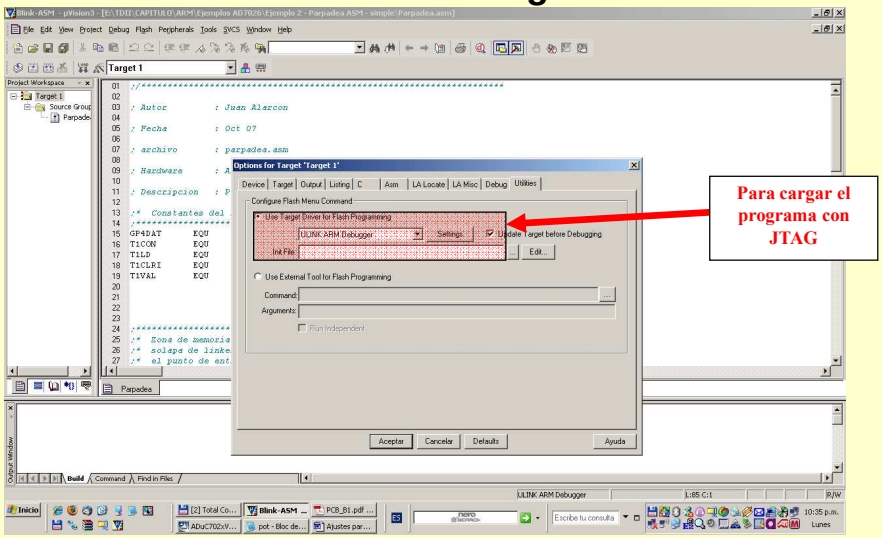
MMRs



Menú: Keil → Flash → Configure Flash Tools → Debug



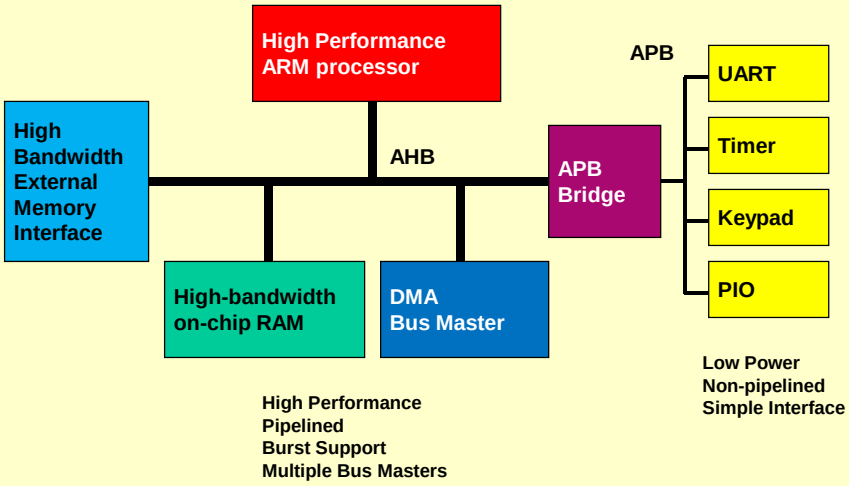
Menú: Keil → Flash → Configure Flash Tools



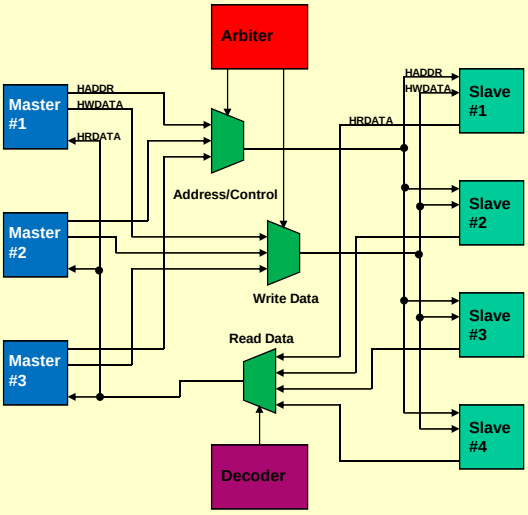
ESTANDAR AMBA.

- Estándar de especificación de buses de conexión.
- Detalla la estrategia para interconexión y manejo de bloques funcionales que conforman un SoC.
- ASB: AMBA System Bus.
- AHB: Advanced High performance Bus, bus de ancho de banda grande para la conexión de bloques funcionales de alta velocidad, tales como la memoria.
- APB: Advanced Peripherial Bus, bus más simple para conectar periféricos de propósito general.

Un ejemplo de AMBA



Estructura AHB



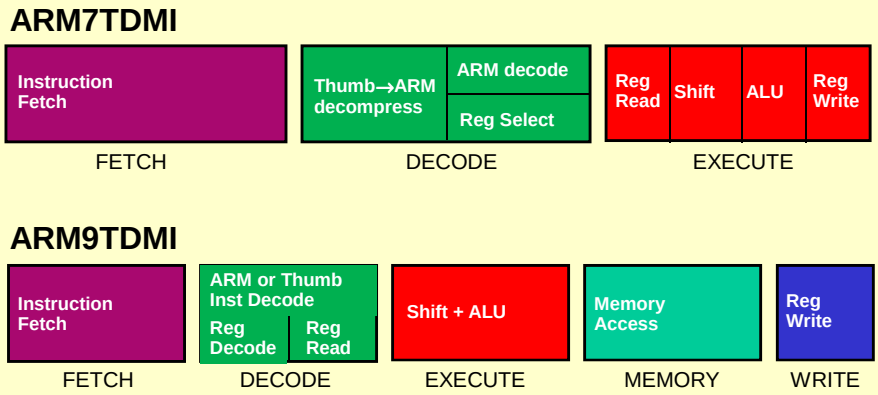
ARM9

- Es una familia constituida por los procesadores ARM920T, ARM922T Y ARM940T.
- Construida en base al procesador ARM9TDMI.
- Set de instrucciones de 16 Bits.
- El procesador es RISC de 32 Bits.
- Buffer de escritura de 8 entradas.

ARM9

- Pipeline de 5 estados que alcanza 1.1 MIPS/MHz, expandible a 300 MIPS.
- Bus de interface AMBA de 32 Bits.
- MMU (Memory Management Unit) que soporta Windows CE, Symbian OS, Linux, Palm OS.
- MPU (Memory Protection Unit) soportando una amplia gama de sistemas operativos en tiempo real, incluyendo VxWORKS.

Pipeline para ARM9TDMI



ARM920T Y ARM922T

- Macrocelulas basadas en el ARM9TDMI RISC de 32 Bits convenientes para una gama de aplicaciones basadas en plataforma OS, ofrecen caches para instrucciones y datos, son idénticos pero se diferencian en que uno es de 16k y el otro de 8k.
- MMU permitiendo soporte para otros sistemas operativos importantes.

APLICACIONES

- En las próximas generaciones de Teléfonos, comunicadores y PDA'S.
- En procesadores 3G.
- En dispositivos basados en Plataforma OS.
- Cámaras digitales.
- Decodificadores de Audio y video.
- En la industria automotriz.