École Supérieure Privé d'Ingénierie et de Technologies

Concepts et Pratique des Processeurs Numériques des Signaux (DSP)

Chiheb Rebai, Nadia Khouja

chiheb.rebai@supcom.rnu.tn, nadia.khouja@supcom.rnu.tn

Objectifs du cours

Mise en évidence de:

- Fonctionnalités de base des processeurs numériques des signaux (DSP)
- Architectures des cores et des périphériques des DSPs
- Méthodologies et Outils de développement, de vérification et de mise au point

Compétences à acquérir:

- Maîtrise des architectures matérielles des DSPs
- Maîtrise du jeu d'instructions pour la programmation des DSP
- Spécification des applications d'implantation sur DSP
- Règles de structuration et de développement de codes DSP
- Maîtrise des outils de développement et de test des applications DSP

Programme du module

Séance 1	*Concepts de base des DSP: définition et architecture générique*	C. Rebai
Séance 2	*Étude des DSP AD-Blackfin: architecture et jeu d'instructions*	C. Rebai
Séance 3	*Initiation à l'environnement de développement intégré Visual DSP++*	C. Rebai
Séance 4	*Application 1: Familiarisation avec le jeu d'instructions du DSP Blackfin*	C. Rebai M. Attia
Séance 5	*Application 2: Implantation d'un filtre à réponse impulsionnelle finie (FIR)*	C. Rebai M. Attia
Séance 6	*Application 3: Chargement des porteuses d'un modulateur DMT*	C. Rebai M. Attia
Séance 7	*Application 4: Fonctions de traitement d'images*	C. Rebai M. Attia

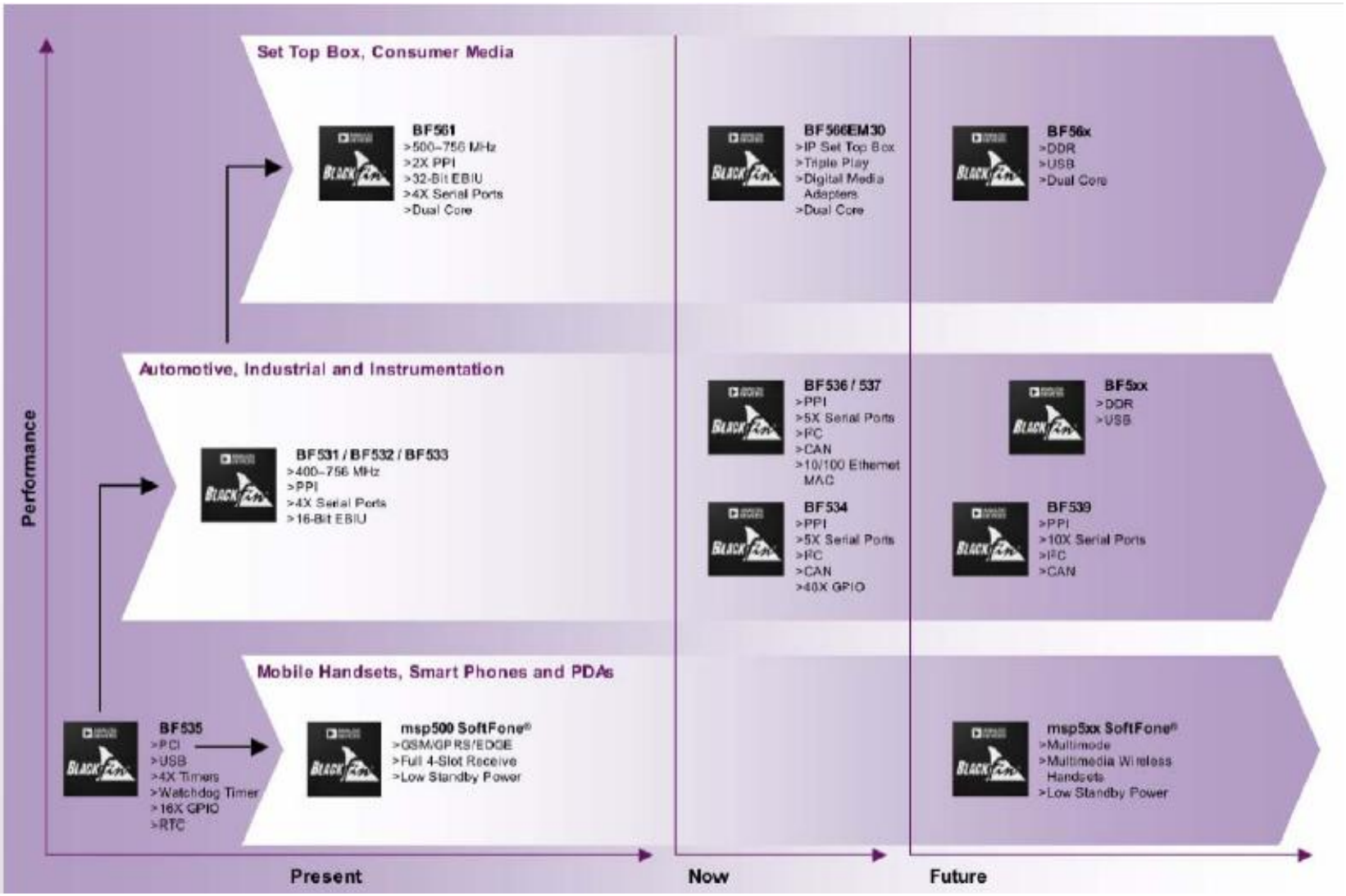
Partie 2

Etude des DSP AD-Blackfin

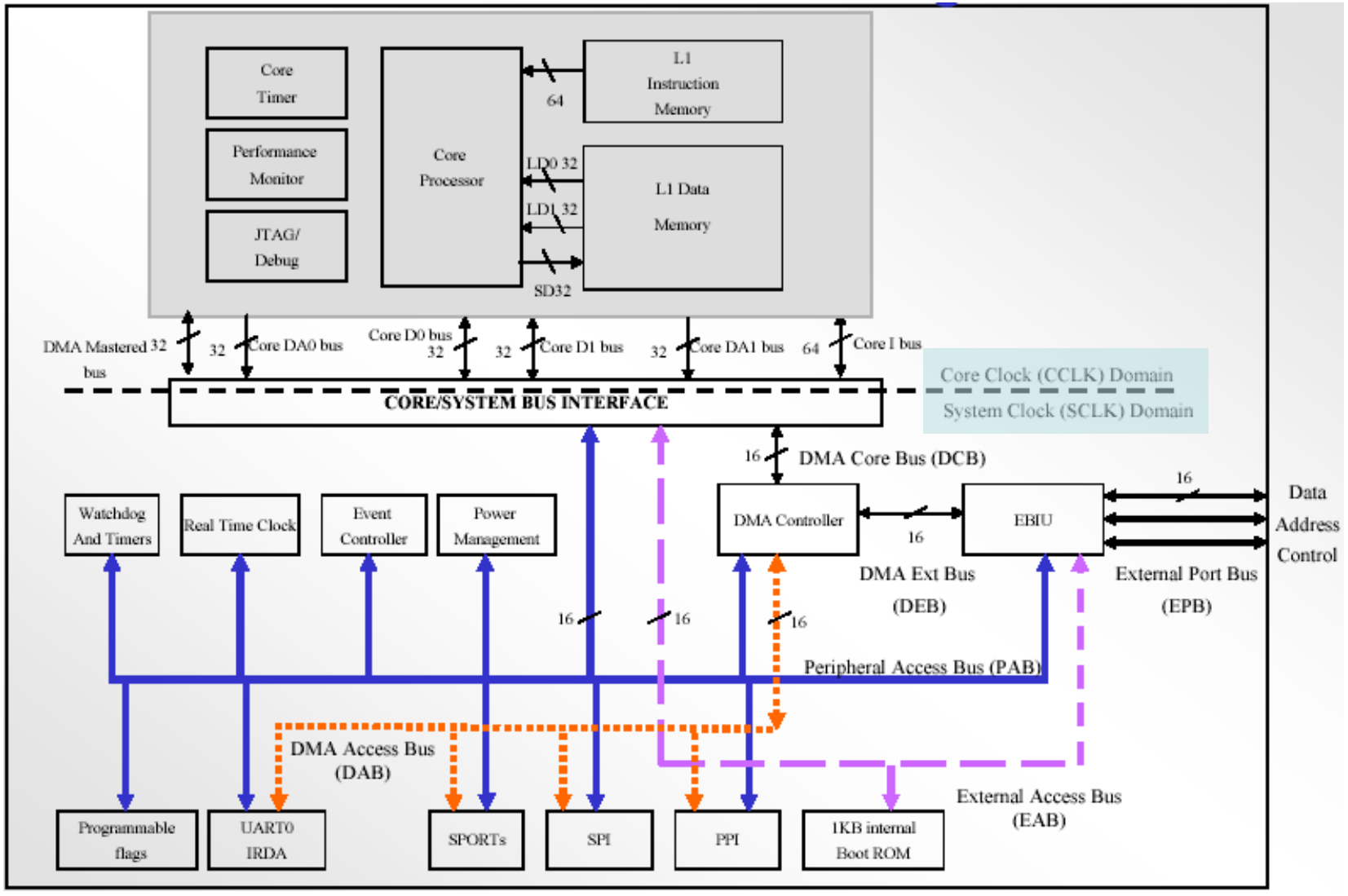
- Architecture du core ADSP Blackfin
- Unités périphériques
- Jeu d'instructions et exemples

Architecture du core ADSP Blackfin

Road Map DSP AD-Blackfin



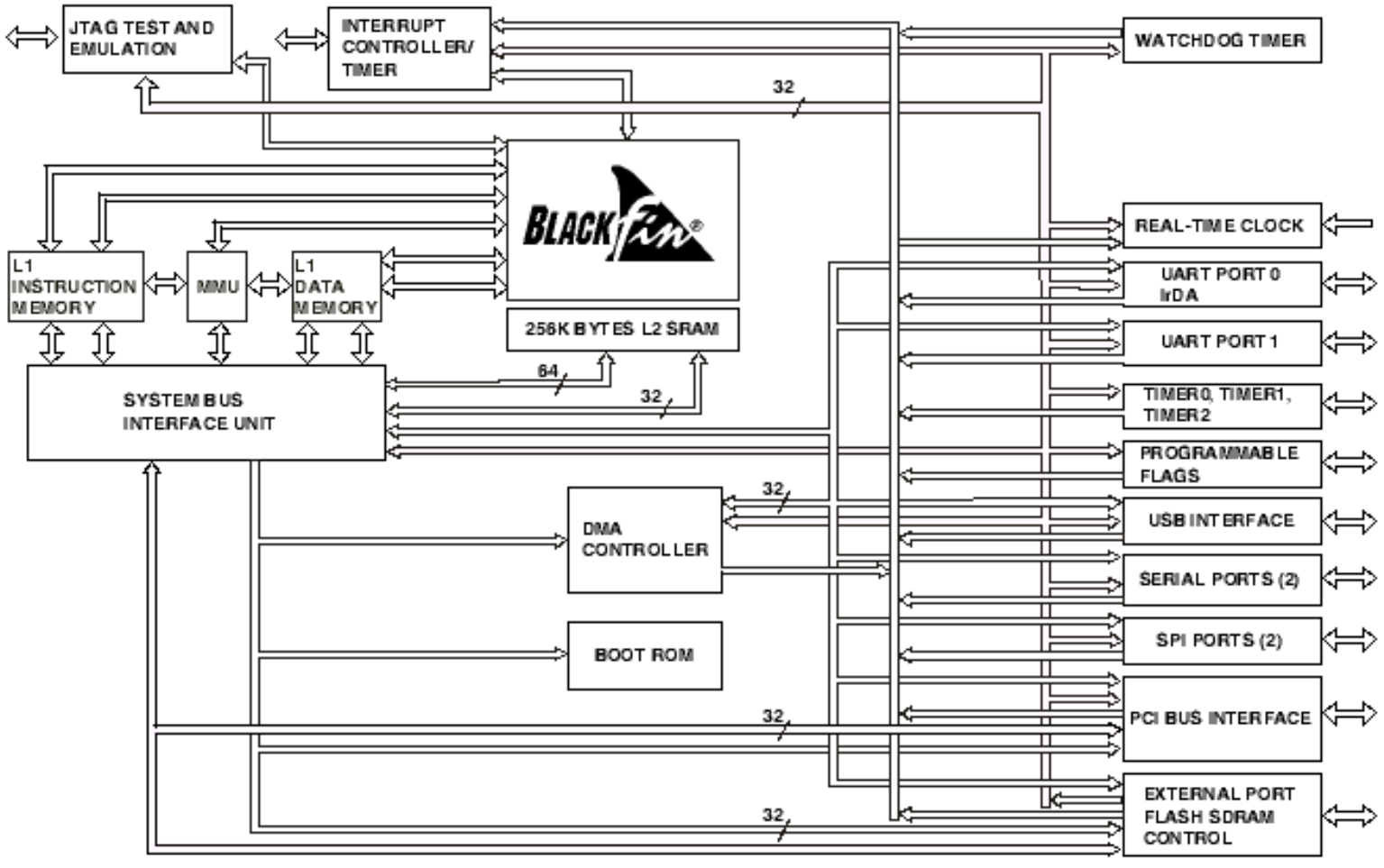
Architecture ADSP BF533



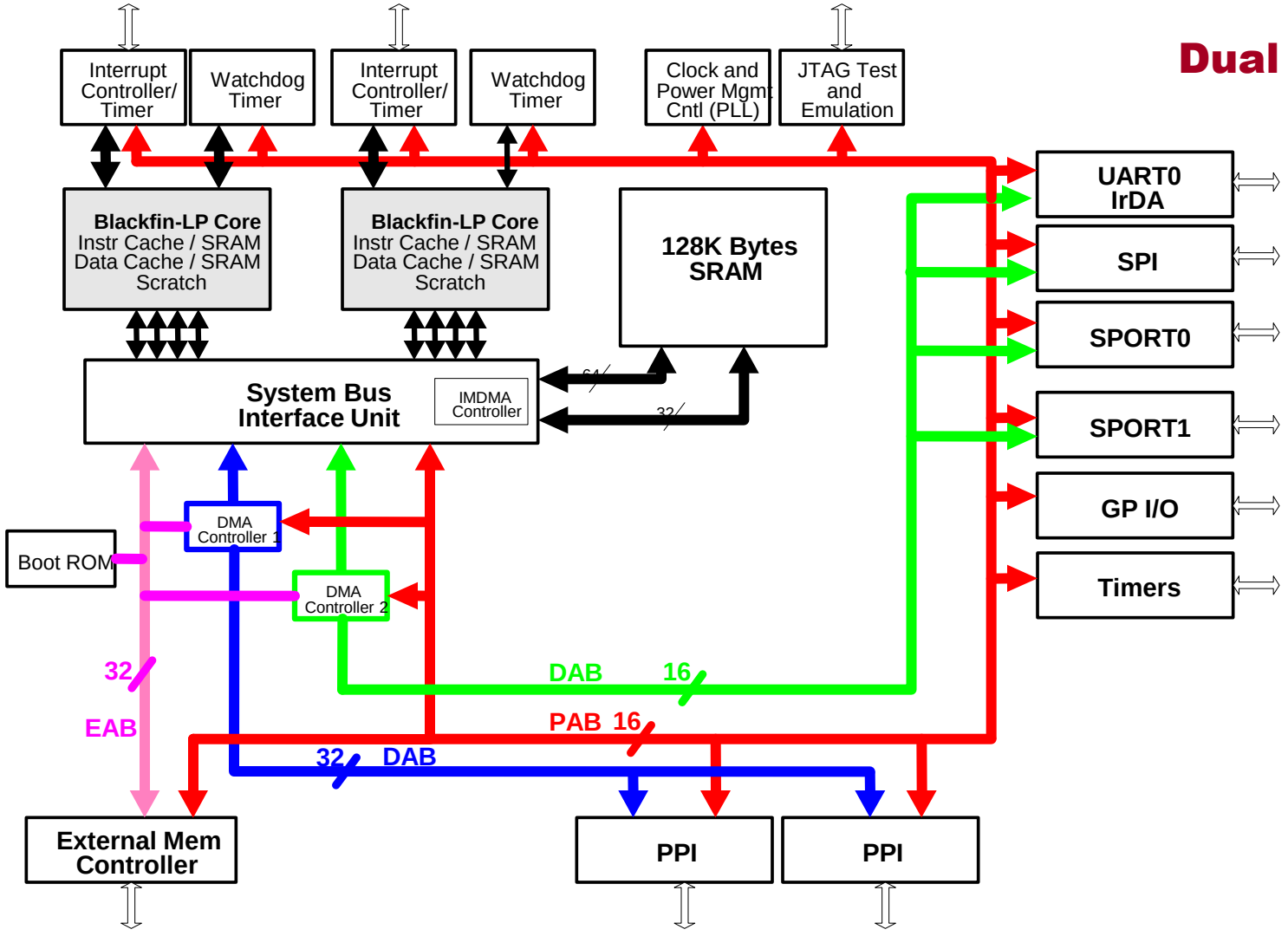
Architecture ADSP BF535

esprit
Ecole Supérieure Privée d'Ingénierie et de Technologies

FUNCTIONAL BLOCK DIAGRAM



Architecture ADSP BF561



Modes opératoires

Les DSP Blackfin supportent 3 modes:

Supervisor Mode

- Permet un accès complet à toutes les ressources périphériques
- Utilisé pour les OS kernel et les device derivers
- Permet de servir les interruptions et les exceptions

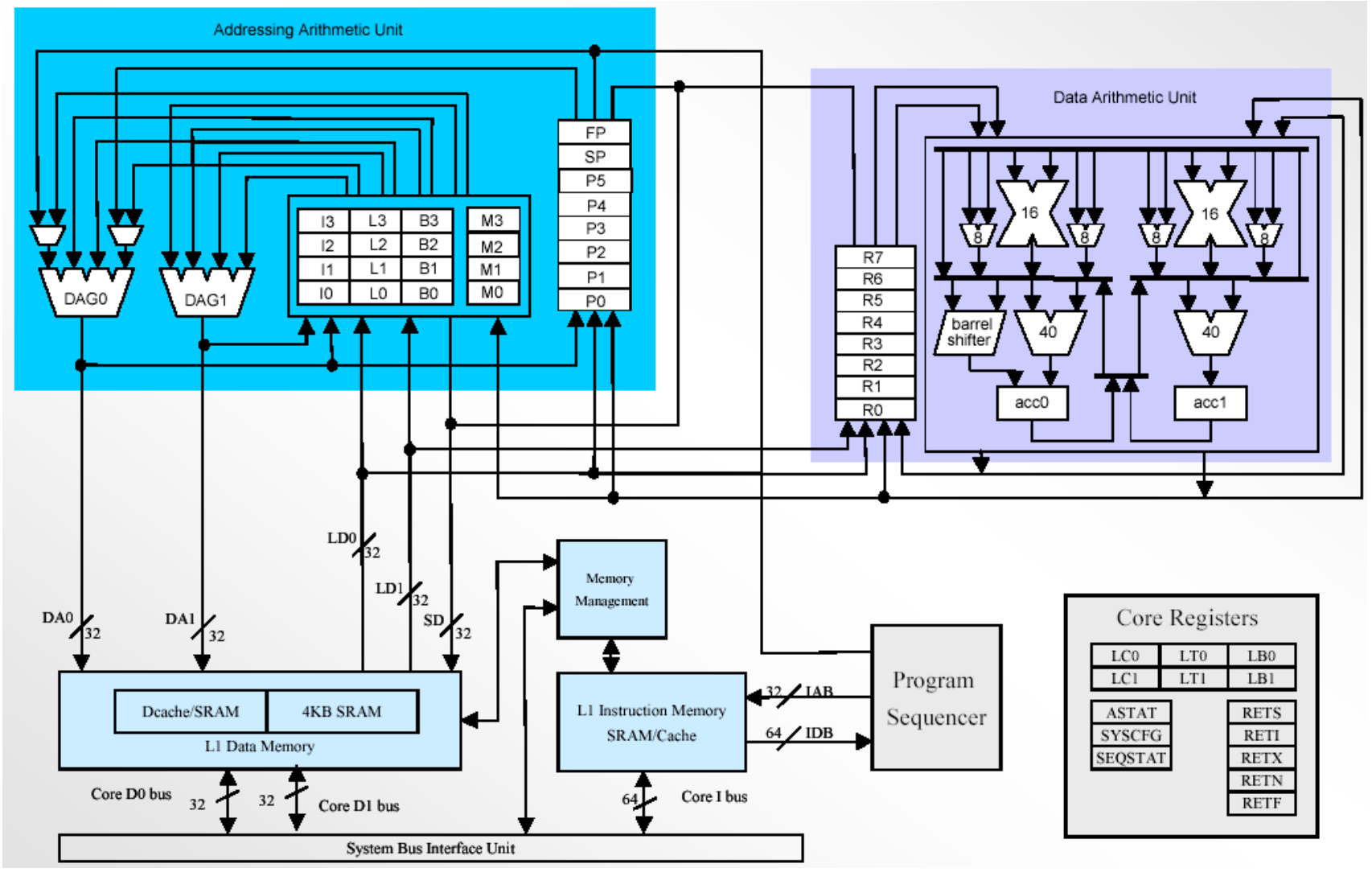
User Mode

- Code d'algorithmes ne permettant pas un accès aux ressources périphériques
- Ne permet pas de servir les interruptions et les exceptions

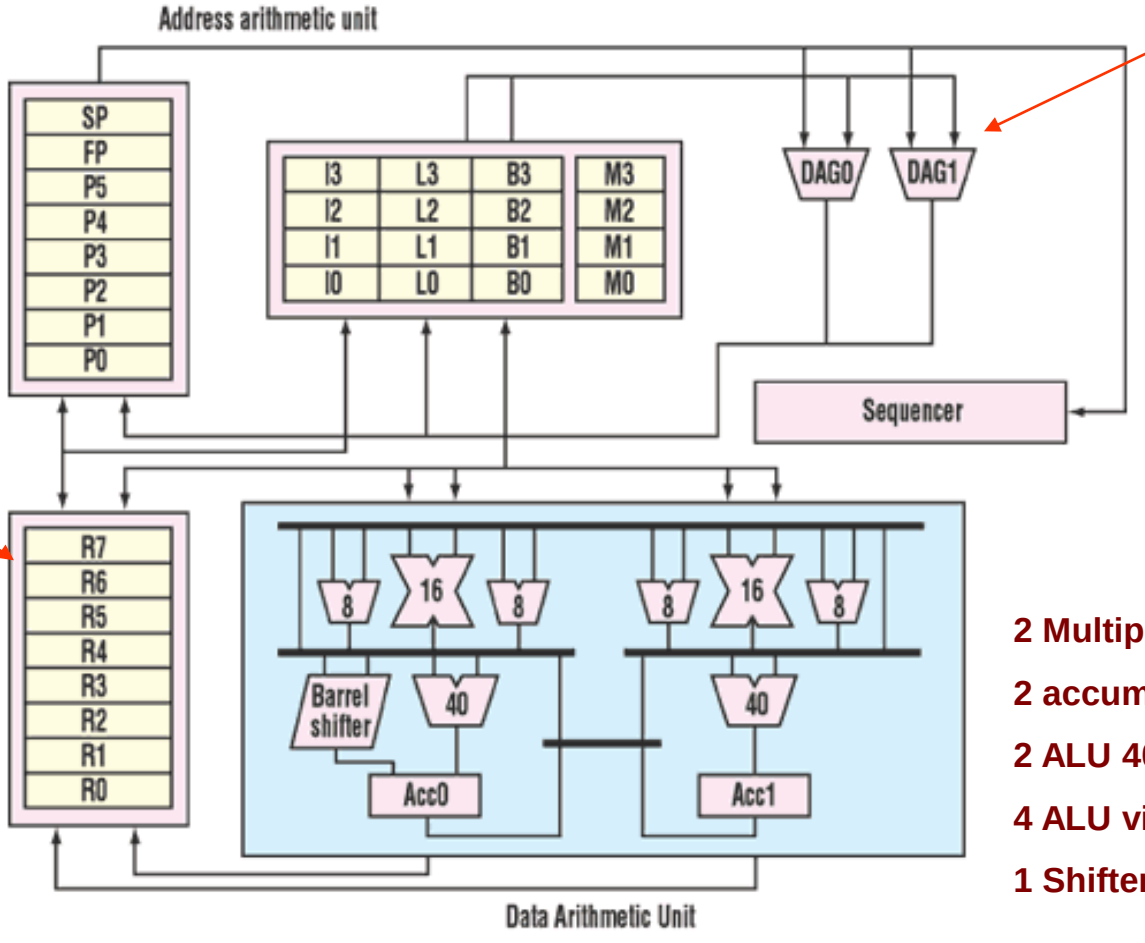
Emulator (or Debug) Mode

- Permet les mêmes fonctionnalités que le **Supervisor mode** et accessible via JTAG

Architecture du core



Unités arithmétiques et d'adressage



2 DAGs

Data Registers

Manipulation de données 8, 16, ou 32 bits à partir du banc de registres de calcul

2 Multiplieurs 16 bits

2 accumulateurs 40 bits

2 ALU 40 bits

4 ALU video 8 bits

1 Shifter

Registres du core

- Tous les traitements sont effectués sur des données contenues dans des registres
- Tous les périphériques sont configurés avec des registres
- L'accès mémoire se fait à travers des pointeurs dans les registres d'adresses
- L'accès aux registres se fait par nom ou registre mappé en mémoire (MMRs : Memory Mapped registers)

Registres du core accessibles par nom

– Data Registers:	R0-R7
– Accumulator Registers:	A0, A1
– Pointer Registers:	P0-P5, FP, SP,USP
– DAG Registers:	I0-I3, M0-M3, B0-B3, L0-L3
– Cycle Counters:	Cycles, cycles2
– Program Sequencer:	SEQSTAT
– System Configuration Register:	SYSCFG
– Loop Registers:	LT[1:0], LB[1:0], LC[1:0]
– Interrupt Return Registers:	RETI, RETX, RETN, RETE

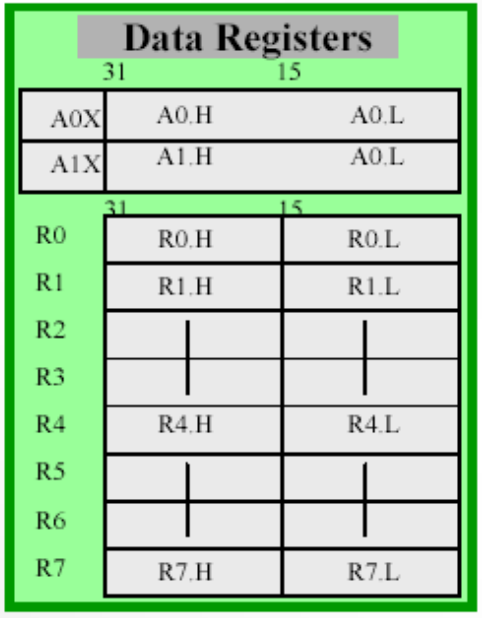
Registres du core

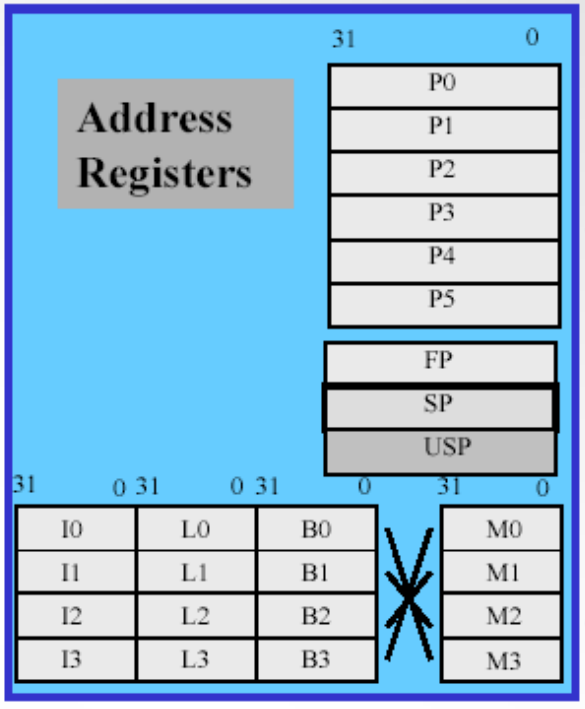
Data Registers:

R0-R7 are referred to as "dreg"

_lo refers to .L and

_hi refers to .H



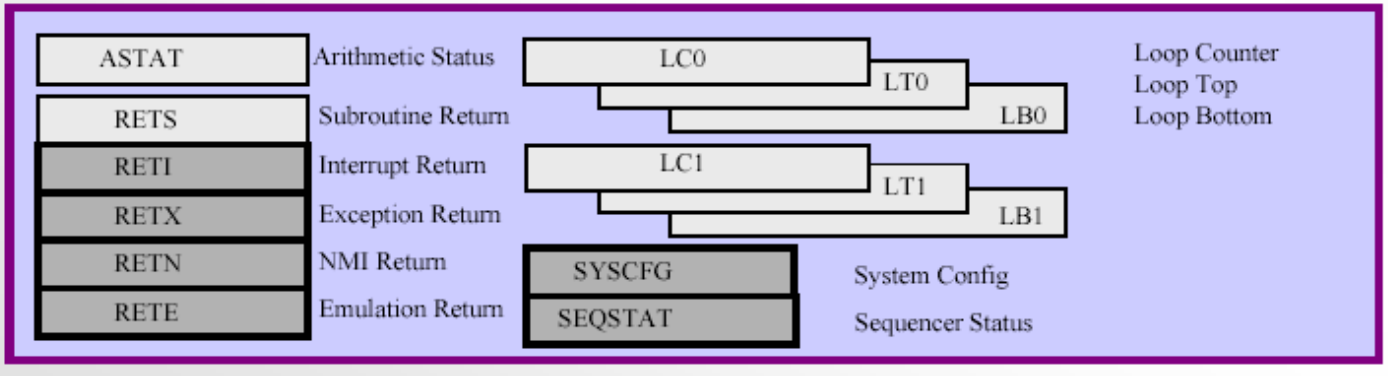


Pointer Registers: *P0-P5* are referred to as "preg"

Frame Pointer

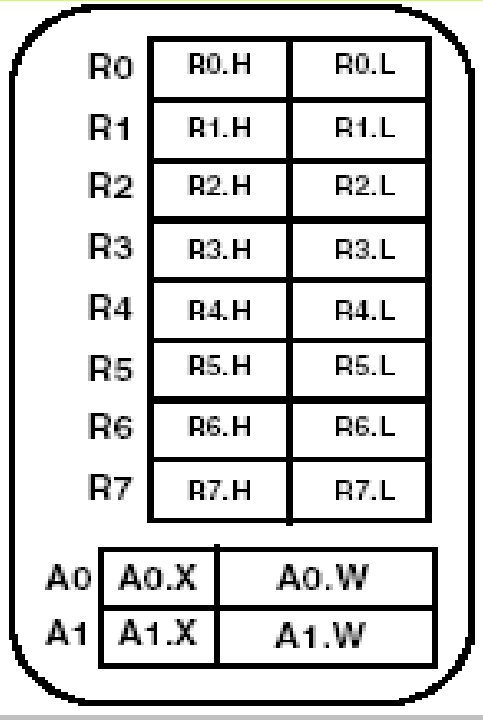
Stack Pointer

Index Registers: *I0-I3* are referred to as "ireg"



Shaded registers only accessible in Supervisor mode

Registres de données



R[7:0] : ***32 bits***

R[7:0].H : ***16 bits***

R[7:0].L : ***16 bits***

A[1:0] : ***40 bits***

A[1:0].X : ***8 bits***

A[1:0].W : ***32 bits***

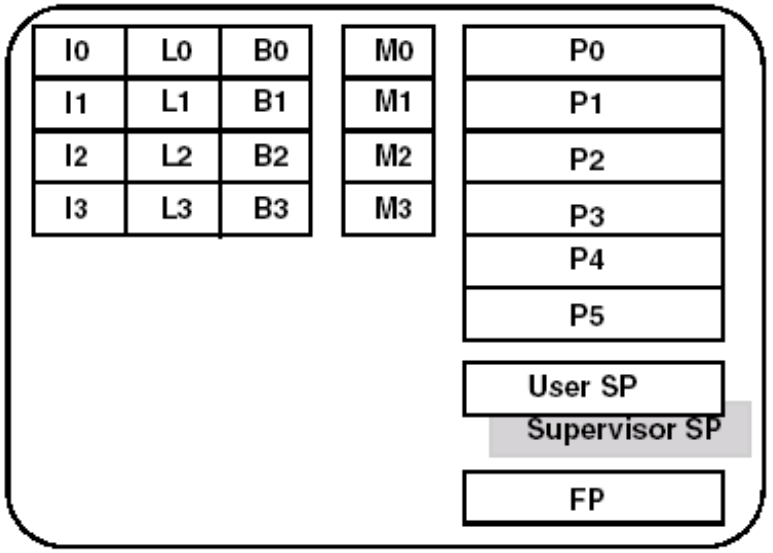
A[1:0].H : ***16 bits***

A[1:0].L : ***16 bits***

Exemple: *accès par nom*

```
R0 = SYSCFG;  // Load data register with contents of SYSCFG register
```

Registres des DAG



Caractéristiques

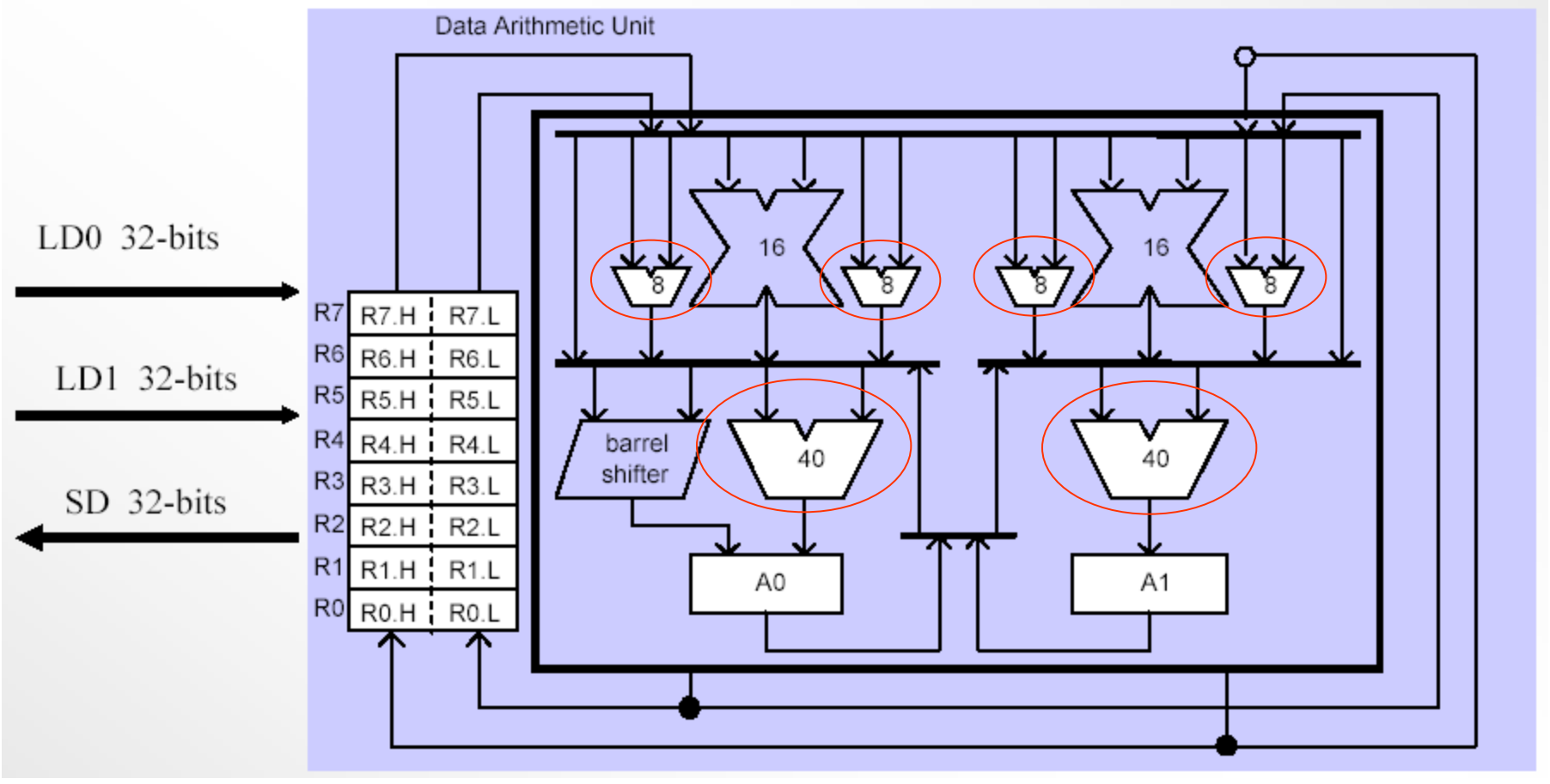
- **Supply address and post-modify**
- **Supply address with offset**
- **Modify address**
- **Bit-reversed carry address**

Index registers:	**I[3:0]**	Unsigned 32-bit
Modify registers:	**M[3:0]**	Signed 32-bits
Base and Length registers:	**B[3:0] and L[3:0]**	Unsigned 32-bits
Pointer registers:	**P[5:0], FP, USP, and SP**	Unsigned 32-bits

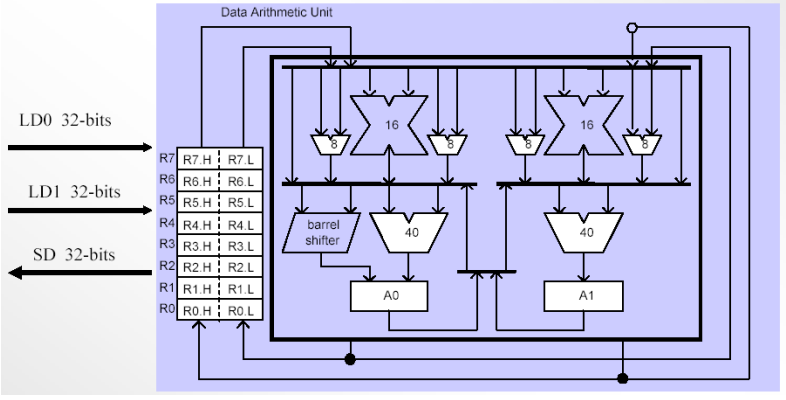
Instructions DAG

```
R0 = [ I2 ] ;          /*loads a 32-bit value from an address pointed to by I2 and stores it in the
                         destination register R0*/
R0 = [ I2++M1 ] ;      /*loads a 32-bit value from an address pointed to by I2 and stores it in the
                         destination register R0, I2 is then updated by M1 content*/
R0.H = W [ I2 ] ;      /* loads a 16-bit value from an address pointed to by I2 and stores it in the
                         16-bit destination register R0.H */
[ P1 ] = R0 ;          /* 32-bit store operation.*/
B [ P1++] = R0 ;       /* stores the 8-bit value from the R0 register in the address pointed to by
                        P1 register, then increments P1 register Byte transfer only with pointers */
R0 = W [ P1++ ] (Z) ;  /* loads a 16-bit word into a 32-bit destination register from an address
                         pointed  to by the P1 Pointer register. The Pointer is then incremented by
                         2 and the word is zero-extended to fill the 32-bit destination register  */
I1 += M2 ;             /* adds M2 to I1 and updates I1 with the new value, no memory access*/
```

Architecture ALU



Fonctions ALU



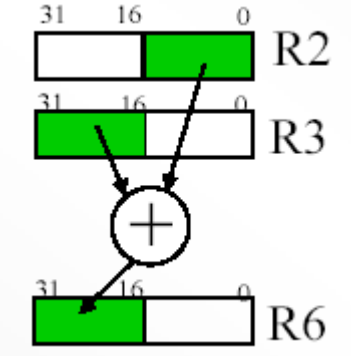
- Single 16-Bit Operations
- Dual 16-Bit Operations
- Quad 16-Bit Operations
- Single 32-Bit Operations
- Dual 32-Bit Operations

- Two 40-bit ALUs operating on 16-bit, 32-bit, and 40-bit input operands and output 16-bit, 32-bit, and 40-bit results.
- Functions
  - Fixed-point addition and subtraction
  - Addition and subtraction of immediate values
  - Accumulator and subtraction of multiplier results
  - Logical AND, OR, NOT, XOR, bitwise XOR (LFSR), Negate
  - Functions: ABS, MAX, MIN, Round, division primitives
  - Supports conditional instructions
- Four 8-bit video ALUs

Opérations ALU 16 bits

Single 16-bit addition

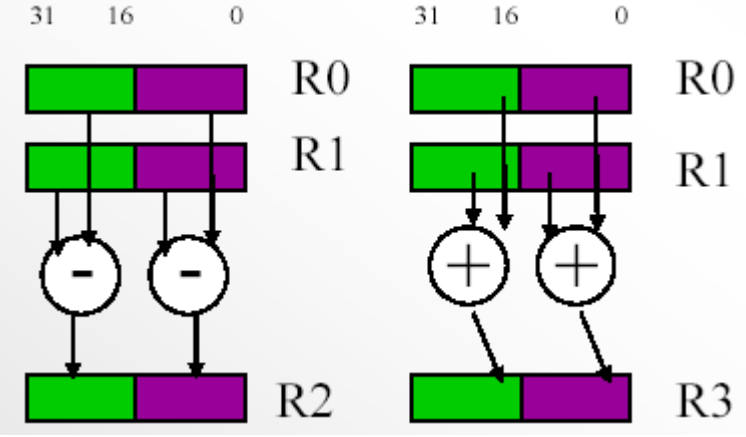
```
R6.H = R3.H + R2.L (s);
```



with saturation

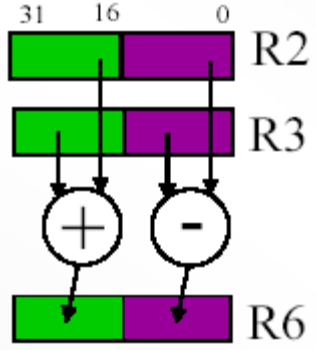
Quad 16-bit addition

```
R3 = R0 + | + R1, R2 = R0 - | - R1;
```



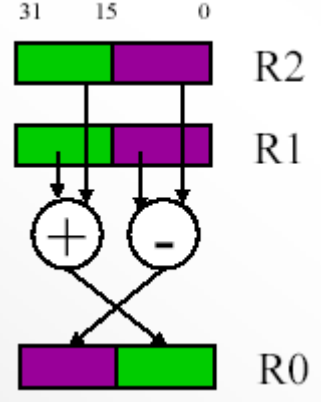
Dual 16-bit addition

```
R6 = R2 + | - R3;
```



Dual 16-bit Cross addition

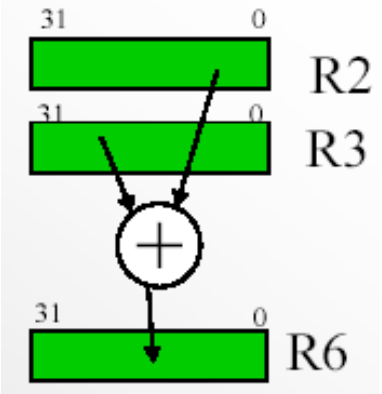
```
R0 = R2 +|- R1 (CO);
```



Opérations ALU 32 bits

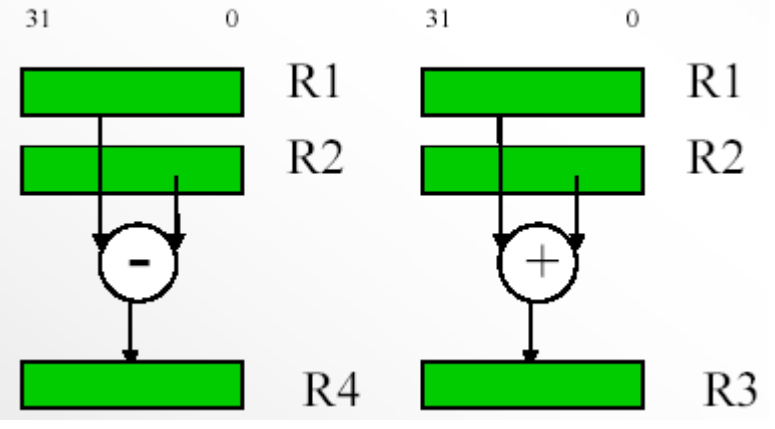
32-bit addition

```
R6 = R2 + R3;
```



Dual 32-bit addition

```
R3 = R1 + R2, R4 = R1 - R2;
```



Opérations ALU logiques

- **AND**

  General Form:

  Dreg = Dreg & Dreg;

  Example:

  R4 = R4 & R3;

- **OR**

  General Form:

  Dreg = Dreg | Dreg;

  Example:

  R4 = R4 | R3;

- **NOT**

  General Form:

  Dreg = ~Dreg;

  Example:

  R3 = ~ R4;

- **XOR**

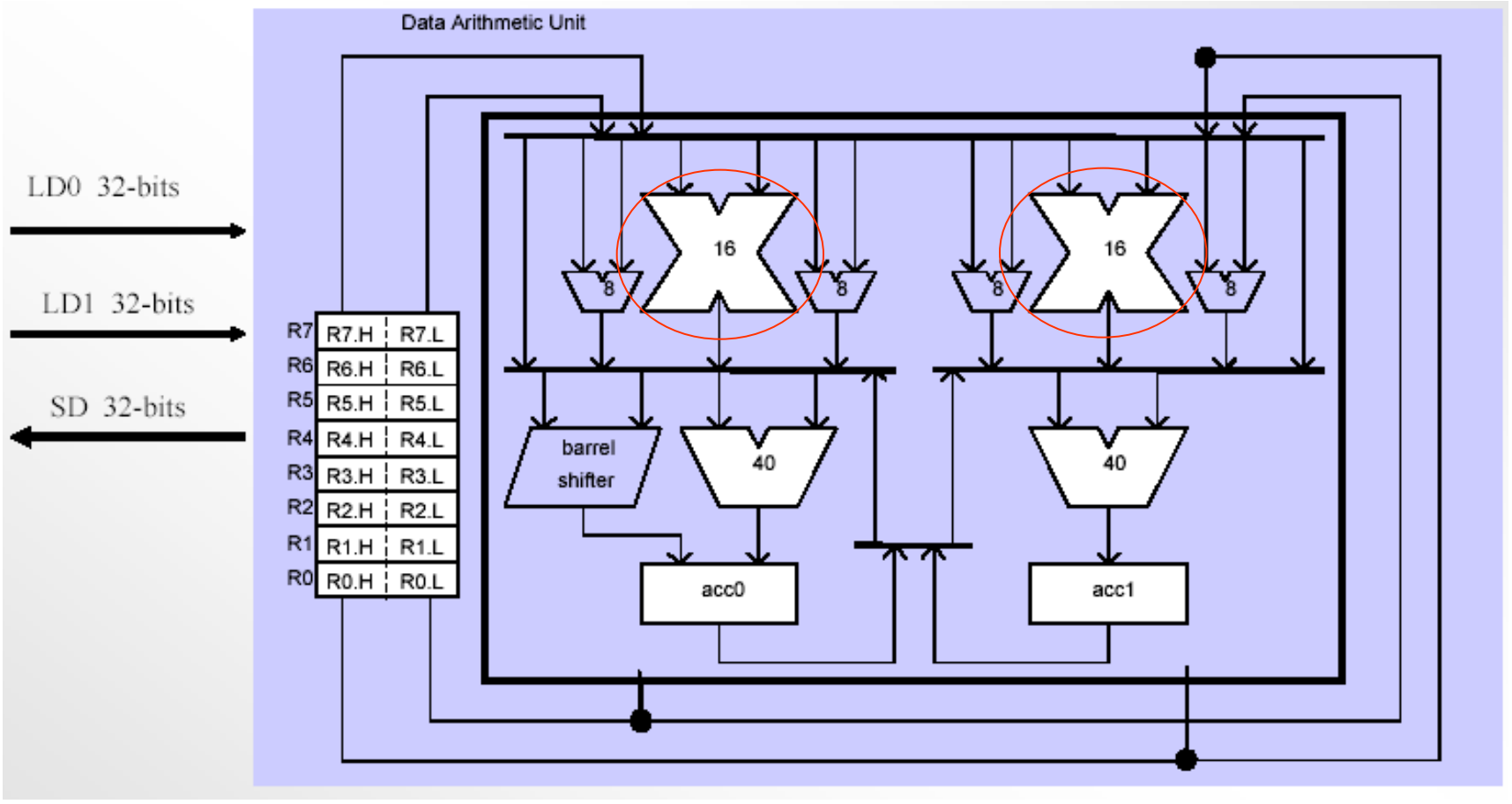
  General Form:

  Dreg = Dreg ^ Dreg;

  Example:

  R4 = R4 ^ R3;

Architecture MAC

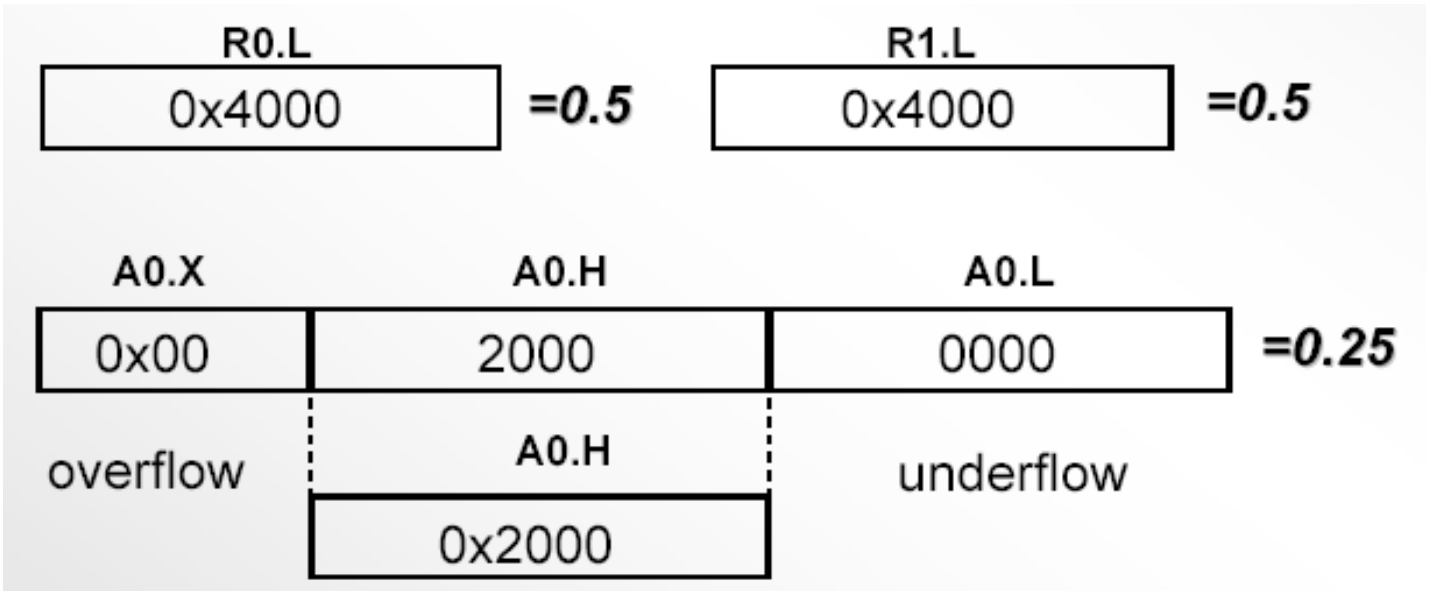


Fonctions MAC

- Two identical MACs
  - Each can perform fixed point multiplication and multiply-and-accumulate operations on 16-bit fixed point input data and outputs 32-bit or 40-bit results depending the destination.
- Functions
  - Multiplication
  - Multiply-and-accumulate with addition (optional rounding)
  - Multiply-and-accumulate with subtraction (optional rounding)
  - Dual versions of the above
- Features
  - Saturation of accumulator results
  - Optional rounding of multiplier results

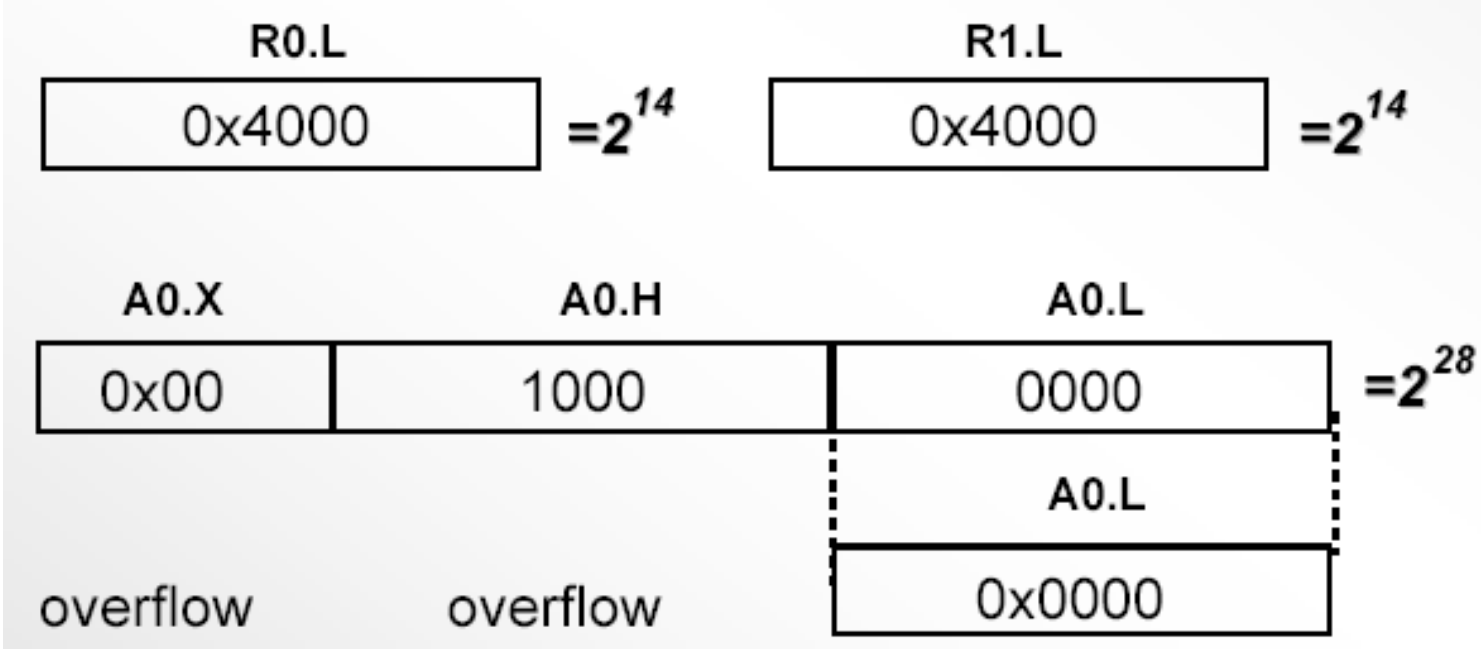
Mode 1: Fractional mode

- Multiplier assumes all numbers in a 1.15 format
- Multiplier automatically shifts product 1-bit left before accumulation (Result forced to 1.31 format)
- Example: A0 = R0.L * R1.L;



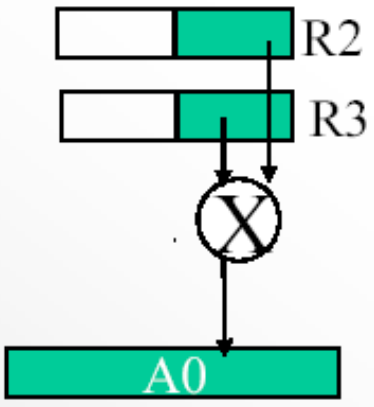
Mode 2: Integer mode

- Multiplier assumes all numbers in a 16.0 format
- No automatic left-shift necessary
- Example: A0 = R0.L * R1.L (IS);

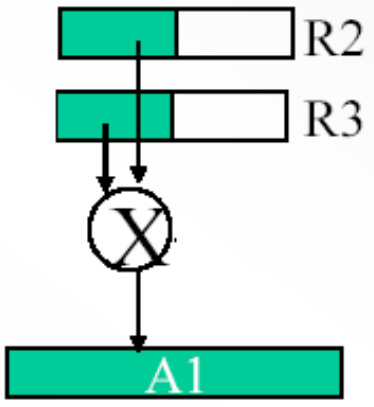


Single MAC 16-bit operation

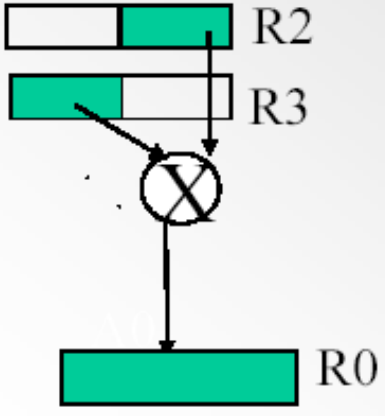
A0 = R2.L * R3.L;



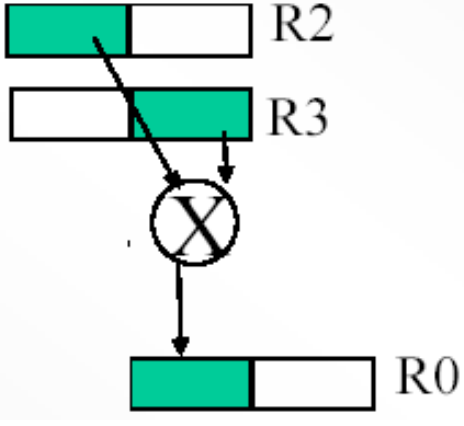
A1 = R2.H * R3.H;



R0 = R2.L * R3.H;

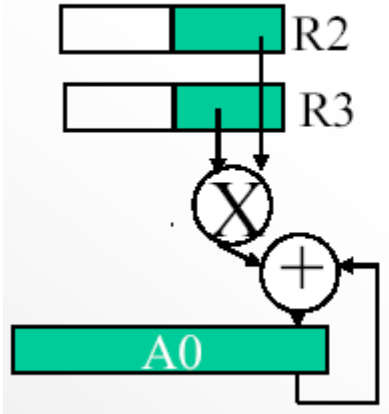


R0.H = R2.H * R3.L;

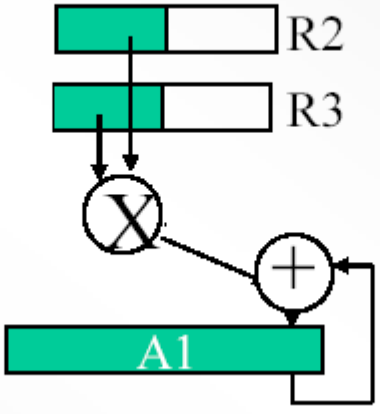


Multiplication + accumulation

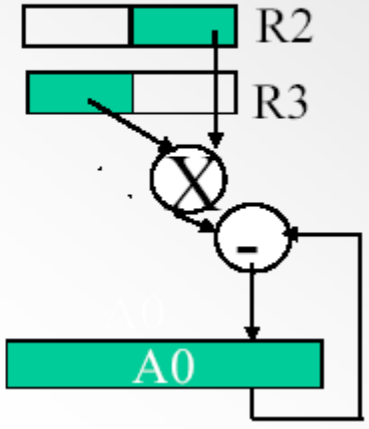
A0 += R2.L * R3.L;



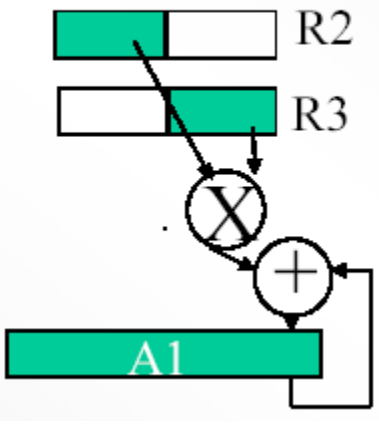
A1 += R2.H * R3.H;



A0 -= R2.L * R3.H;

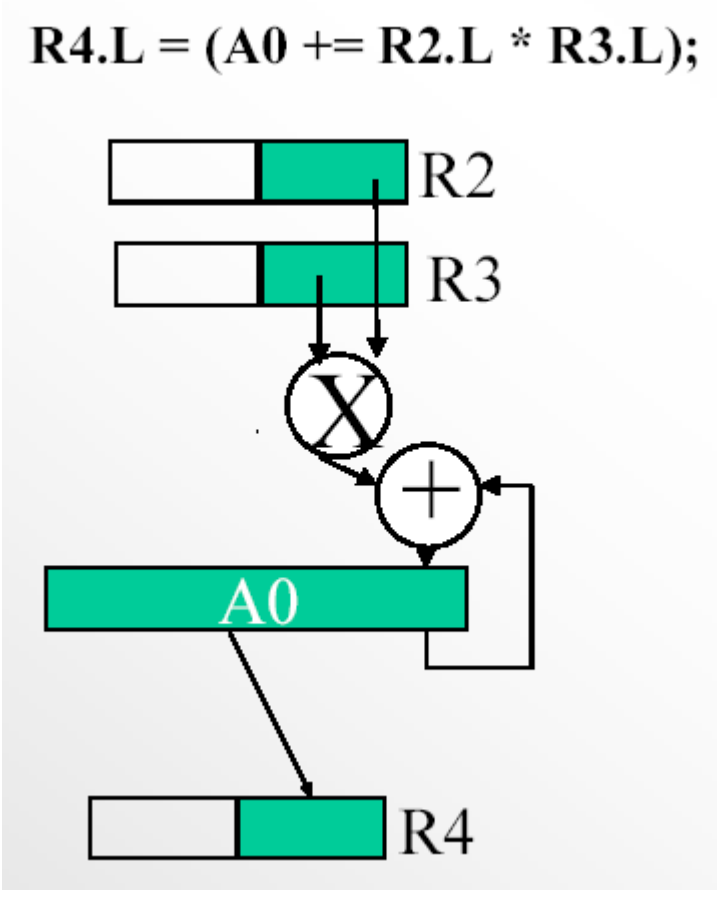


A1 += R2.H * R3.L;

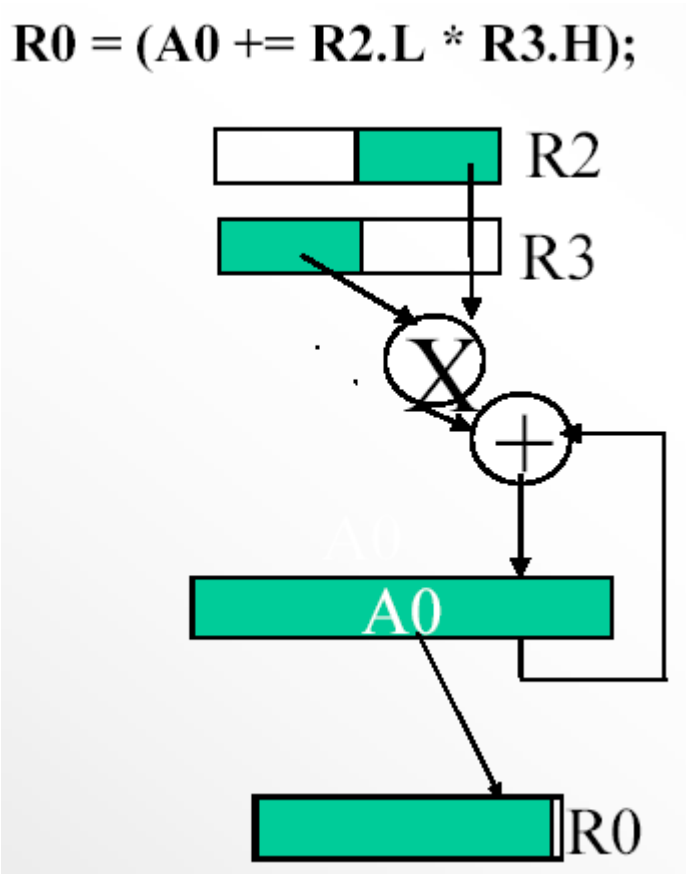


MAC avec transfert DReg

R4.L = (A0 += R2.L * R3.L);

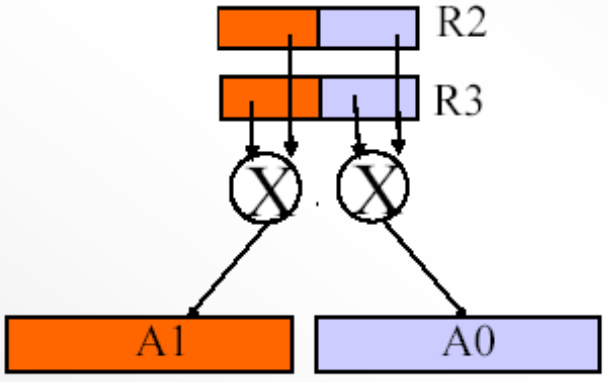


R0 = (A0 += R2.L * R3.H);

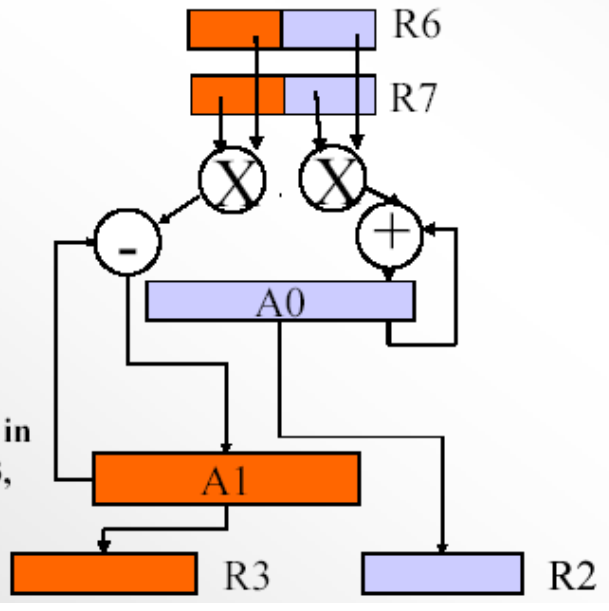


Dual MAC operations

A1 = R2.H * R3.H, A0 = R2.L * R3.L;



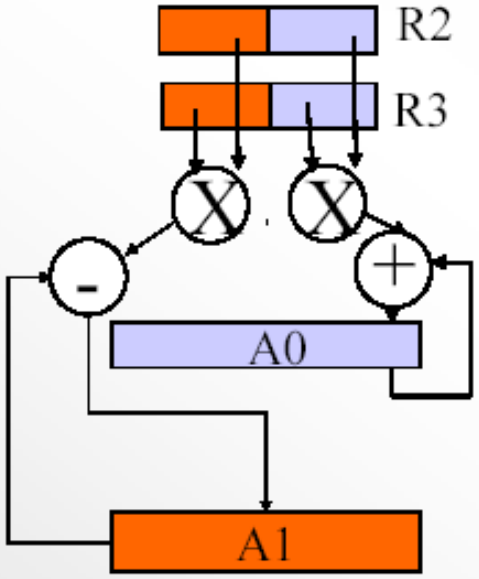
R3 = (A1 += R7.H * R6.H), R2 = (A0 += R7.L * R6.L);



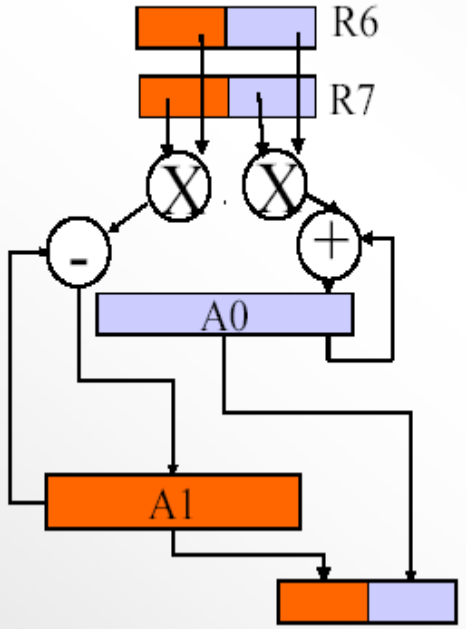
32-bit Data Register Destinations must be used in pairs, e.g. R0:R1 or R2:R3, R4:R5 or R6:R7

Dual MAC operations

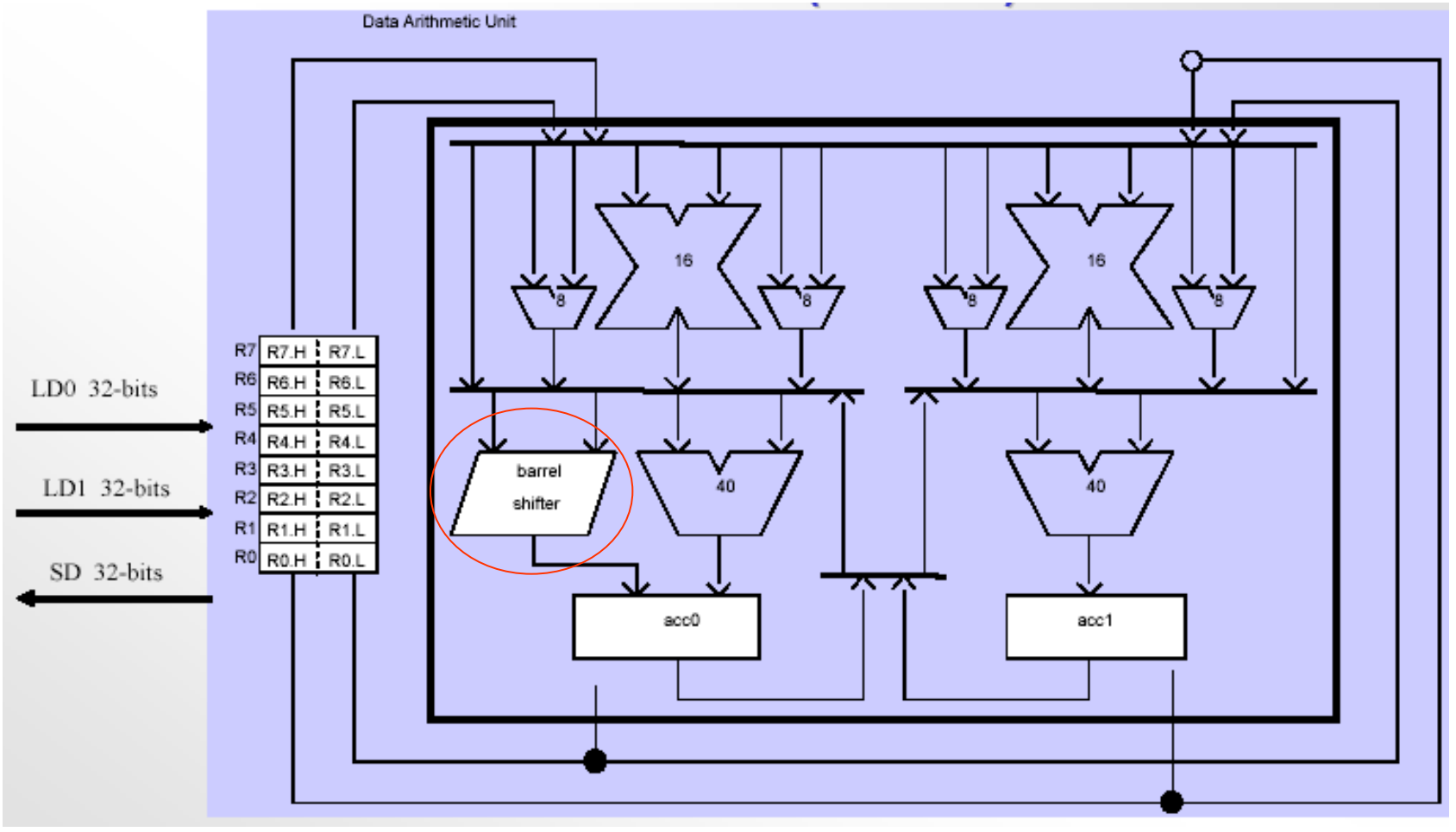
A1 -= R2.H * R3.H, A0 += R2.L * R3.L;



R2.H = (A1 += R7.H * R6.H), R2.L = (A0 += R7.L * R6.L);



Architecture du SHIFTER



Fonctions SHIFTER

- The shifter performs bitwise shifting for 16-bit, 32-bit or 40-bit inputs and yields 16-bit, 32-bit, or 40-bit outputs.
- Functions
  - Arithmetic Shift: The Arithmetic Shift instruction shifts a registered number a specified distance and direction while preserving the sign of the original number. The sign bit value back-fills the left-most bit positions vacated by the arithmetic right shift.
  - Logical Shift: The Logical Shift instruction logically shifts a registered number a specified distance and direction. Logical shifts discard any bits shifted out of the register and backfill vacated bits with zeros.

Opérations SHIFTER

esprit
Ecole Supérieure Privée d'Ingénierie et de Technologies

Décalage arithmétique

- Immediate Shift Magnitude

```
R3.L = R0.H >>> 7;          /* arithmetic right shift, half word */
R5 = R2 << 24 (S);          /* arithmetic left shift */
```

- Registered Shift Magnitude

```
R3.L = ashift R0.H by R7.L;   /* arithmetic shift, half-word */
A0 = ashift A0 by R7.L;       /* arithmetic shift, accumulator */
```

Rotation

- Immediate Rotate Magnitude

```
R4 = rot R1 by 8;        /* rotate left  by 8 */
A0 = rot A0 by –5;       /* rotate right by 5 */
```

- Registered Rotate Magnitude

```
R4 = rot R1 by R2.L       /* rotate by value in R2.L */
A1 = rot A1 by R7.L       /* rotate by value in R7.L */
```

Décalage logique

- Pointer shift, fixed magnitude

```
P3 = P2 >> 1;          /* pointer right shift by 1 */
P0 = P1 << 2;          /* pointer left shift by 2 */
```

- Data shift, immediate shift magnitude

```
R3.L = R0.L >> 4;      /* data right shift, half word register */
R3 = R0 << 12;         /* data left shift, 32-bit word */
A0 = A0 << 7;          /* accumulator left shift */
```

- Data shift, registered shift magnitude

```
R3.H = lshift R0.L by R2.L;    /* logical shift, half word register */
A1 = lshift A1 by R7.L;        /* logical shift, accumulator */
```

Program Sequencer

Fonctionnalités Program Sequencer

- Maintains Loops, Subroutines, Jumps, Idle, Interrupts and Exceptions
- Contains a 10-stage instruction pipeline
- Includes Zero-Overhead Loop Registers

Register Name	Description
SEQSTAT	Sequencer Status register
RETX RETN RETI RETE RETS	Return Address registers: See "Events and Sequencing" on page 4-18. Exception Return NMI Return Interrupt Return Emulation Return Subroutine Return
LC0, LC1 LT0, LT1 LB0, LB1	Zero-Overhead Loop registers: Loop Counters Loop Tops Loop Bottoms
FP, SP	Frame Pointer and Stack Pointer: See "Frame and Stack Pointers" on page 5-5
SYSCFG	System Configuration register
CYCLES, CYCLES2	Cycle Counters: See "CYCLES and CYCLES2 Registers" on page 19-25
PC	Program Counter

Instructions Program Sequencer

Program Flow Instruction	*Instruction Function*
JUMP	Unconditional Branch
IF CC JUMP IF !CC JUMP	Conditional Branch
CALL	Subroutine call
RTS,RTI,RTX,RTN,RTE	Return from Flow interrupter
LSETUP	Set up Hardware Loop

- Jump (P5); /* indirect jump instruction */
- Jump (PC + P3); /* indirect jump with offset (PC-relative) */
- Call (P5); /* RETS register is loaded with address of instruction after call */
- Call (PC + P3); /* RETS register is loaded with address of instruction after call */
- IF CC Jump <label>; /* jump on condition cc=1 */
- Call <label>; /* OK within 24-bit offset from PC */

Instructions Pipeline

Pipeline Stage	Description
Instruction Fetch 1 (IF1)	Start instruction memory access.
Instruction Fetch 2 (IF2)	Intermediate memory pipeline.
Instruction Fetch 3 (IF3)	Finish L1 instruction memory access.
Instruction Decode (DEC)	Align instruction, start instruction decode, and access Pointer register file.
Address Calculation (AC)	Calculate data addresses and branch target address.
Execute 1 (EX1)	Start access of data memory.
Execute 2 (EX2)	Register file read.
Execute 3 (EX3)	Finish accesses of data memory and start execution of dual cycle instructions.
Execute 4 (EX4)	Execute single cycle instructions.
Write Back (WB)	Write states to Data and Pointer register files and process events.

Exécution Pipeline

	Inst Fetch1	Inst Fetch2	Inst Fetch3	Inst. Decode	Address Calc	Ex1	Ex2	Ex3	Ex4	WB
Inst Fetch1	Inst Fetch2	Inst Fetch3	Inst. Decode	Address Calc	Ex1	Ex2	Ex3	Ex4	WB	

	Pipeline Stage									
		IF1	IF2	IF3	DC	AC	EX1	EX2	EX3	EX4	WB
T	1	Insta	Inst9	Inst8	Inst7	Inst6	Inst5	Inst4	Inst3	Inst2	Inst1
I	2		Insta	Inst9	Inst8	Inst7	Inst6	Inst5	Inst4	Inst3	Inst2
M	3			Insta	Inst9	Inst8	Inst7	Inst6	Inst5	Inst4	Inst3
E	4				Insta	Inst9	Inst8	Inst7	Inst6	Inst5	Inst4
↓	5					Insta	Inst9	Inst8	Inst7	Inst6	Inst5
	6						Insta	Inst9	Inst8	Inst7	Inst6
	7							Insta	Inst9	Inst8	Inst7
	8								Insta	Inst9	Inst8
	9									Insta	Inst9
	10										Insta

Program Flow Control

(If CC) jump *get_new_sample* **;** /* assembler resolved target, abstract offsets */

(If !CC) jump (p5) ; /* P5 contains the absolute address of the target */

call (p5) ;

call *get_next_sample* **;**

P5 = 0x20;

LSETUP (*lp_start, lp_end***) LC0 = P5;** /* loop setup instruction */

lp_start: **R5 = R0 + R1(ns) || R0 = [P2++] || R1 = [I1++];**

lp_end: **R5 = R5 + R2;**

Two sets of loop registers are used to manage two nested loops:

- LC[1:0] – the Loop Count registers
- LT[1:0] – the Loop Top address registers
- LB[1:0] – the Loop Bottom address registers

Interruptions vs Exceptions

esprit
Ecole Supérieure Privée d'Ingénierie et de Technologies

INTERRUPTS

- Hardware-generated
  - Asynchronous to program flow
  - Requested by a peripheral
- Software-generated
  - Synchronous to program flow
  - Generated by RAISE instruction
- All instructions preceding the interrupt in the pipeline are killed

EXCEPTIONS

- Service Exception
  - Return address is the address following the excepting instruction
  - Never re-executed
  - EXCPT instruction is in this category
- Error Condition Exception
  - Return address is the address of the excepting instruction
  - Excepting instruction will be re-executed

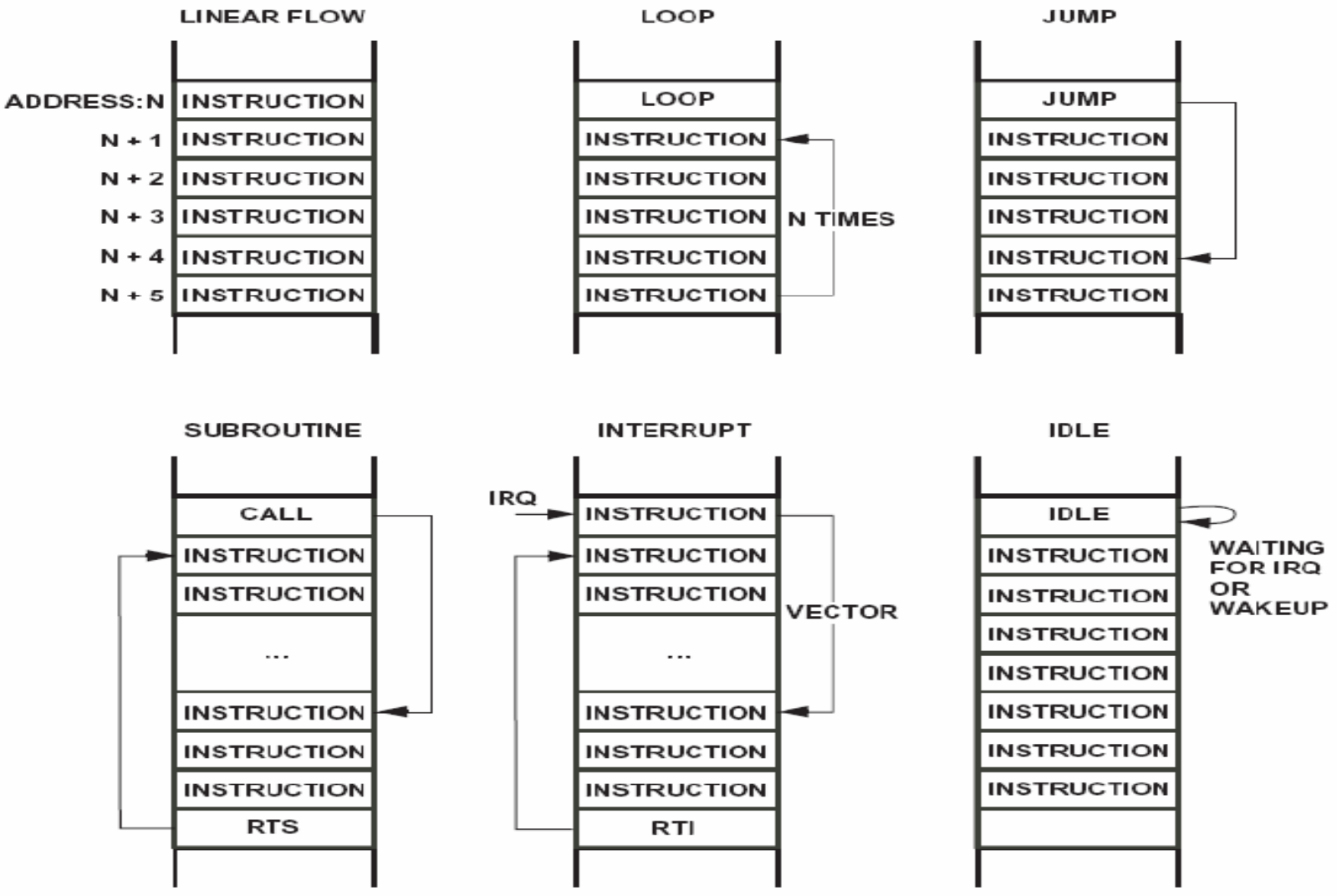
Classification des interruptions

System Interrupt Source	IVG #[1]
PLL Wakeup interrupt	IVG7
DMA error (generic)	IVG7
PPI error interrupt	IVG7
SPORT0 error interrupt	IVG7
SPORT1 error interrupt	IVG7
SPI error interrupt	IVG7
UART error interrupt	IVG7
RTC interrupt	IVG8
DMA 0 interrupt (PPI)	IVG8
DMA 1 interrupt (SPORT0 RX)	IVG9
DMA 2 interrupt (SPORT0 TX)	IVG9
DMA 3 interrupt (SPORT1 RX)	IVG9
DMA 4 interrupt (SPORT1 TX)	IVG9
DMA 5 interrupt (SPI)	IVG10
DMA 6 interrupt (UART RX)	IVG10
DMA 7 interrupt (UART TX)	IVG10
Timer0 interrupt	IVG11
Timer1 interrupt	IVG11
Timer2 interrupt	IVG11
PF interrupt A	IVG12
PF interrupt B	IVG12
DMA 8/9 interrupt (MemDMA0)	IVG13
DMA 10/11 interrupt (MemDMA1)	IVG13
Watchdog Timer Interrupt	IVG13

Event Source	IVG #	Core Event Name
Emulator	0	EMU
Reset	1	RST
Non Maskable Interrupt	2	NMI
Exceptions	3	EVSW
Reserved	4	-
Hardware Error	5	IVHW
Core Timer	6	IVTMR
General Purpose 7	7	IVG7
General Purpose 8	8	IVG8
General Purpose 9	9	IVG9
General Purpose 10	10	IVG10
General Purpose 11	11	IVG11
General Purpose 12	12	IVG12
General Purpose 13	13	IVG13
General Purpose 14	14	IVG14
General Purpose 15	15	IVG15

[1] Note: Default IVG configuration shown.

Variation Program Flow



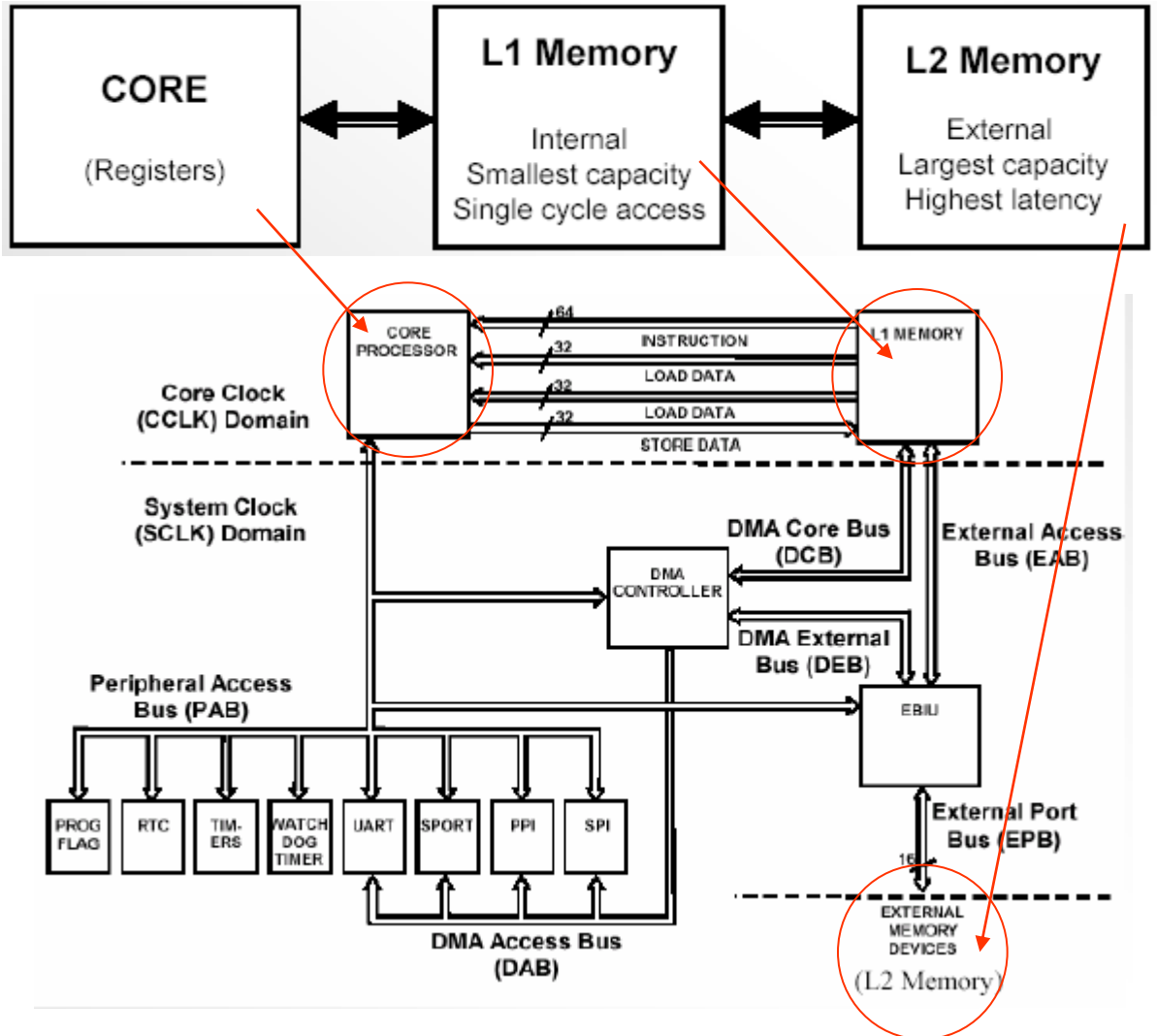
Caractéristiques de la mémoire

32 bits d'adresses ⇒ **Capacité maximale d'adressage = 4 Goctets**

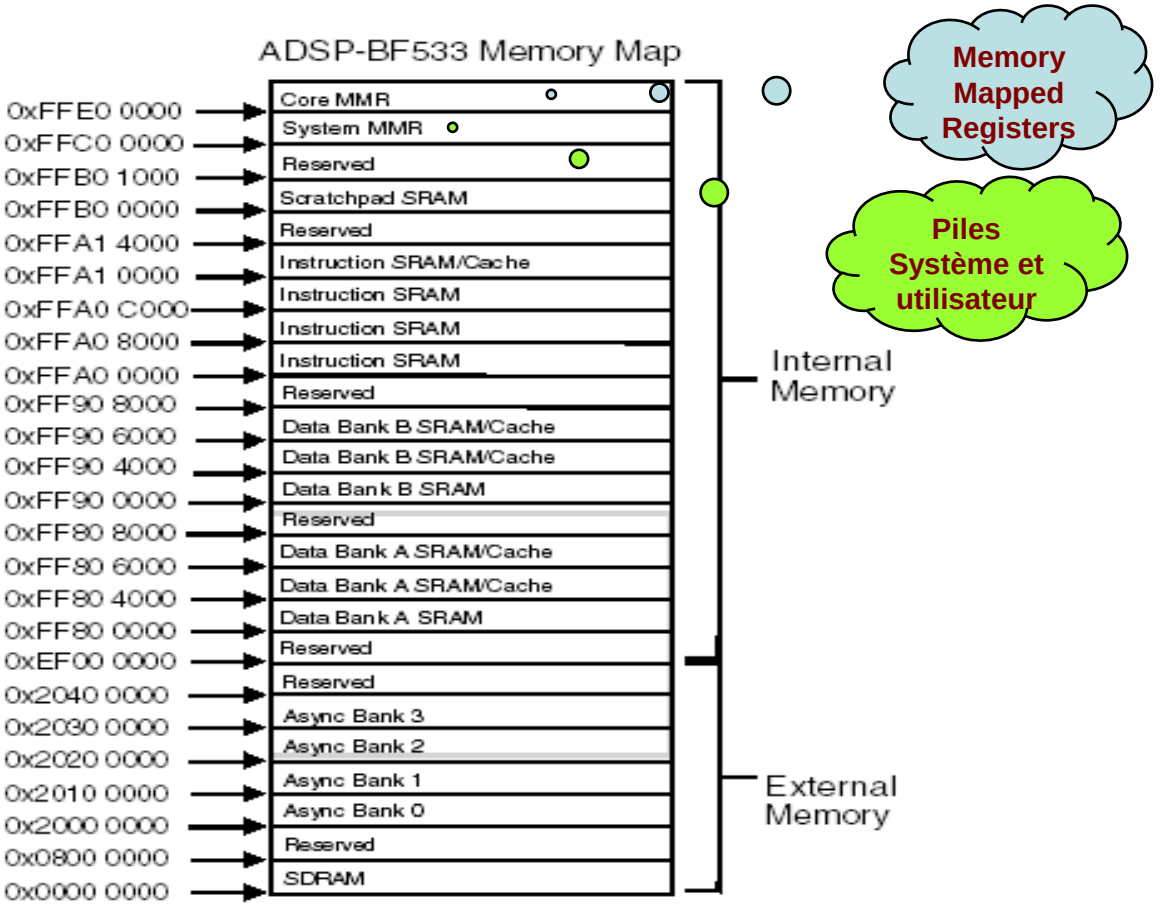
- **L1 Static Random Access Memories (SRAM)**
- **A set of memory-mapped registers (MMRs)**
- **A boot Read Only Memory (ROM)**

Type of Memory	ADSP-BF531	ADSP-BF532	ADSP-BF533
Instruction SRAM/Cache, lockable by Way or line	16 KB	16 KB	16 KB
Instruction SRAM	16 KB	32 KB	64 KB
Instruction ROM	32 KB	32 KB	-
Data SRAM/Cache	16 KB	32 KB	32 KB
Data SRAM	-	-	32 KB
Data Scratchpad SRAM	4 KB	4 KB	4 KB
Total	84 KB	116 KB	148 KB

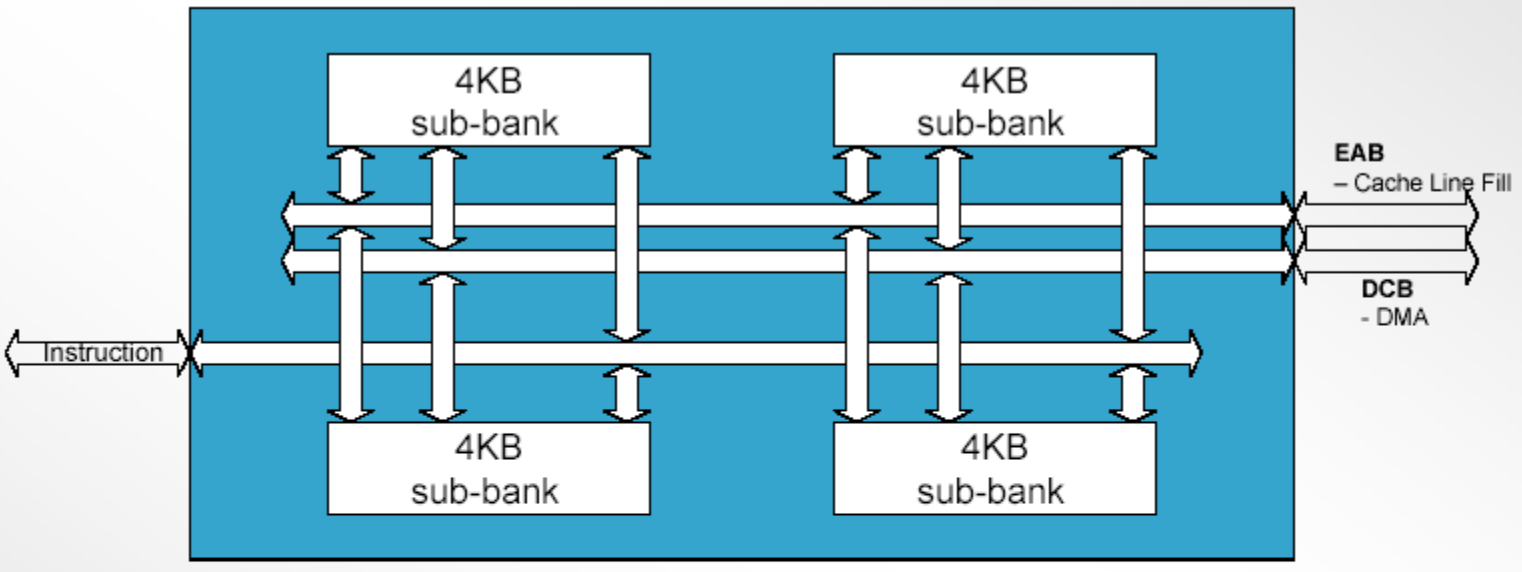
Hiérarchie mémoire



Mapping de la mémoire



Mémoire L1 - Instructions



16 KB SRAM

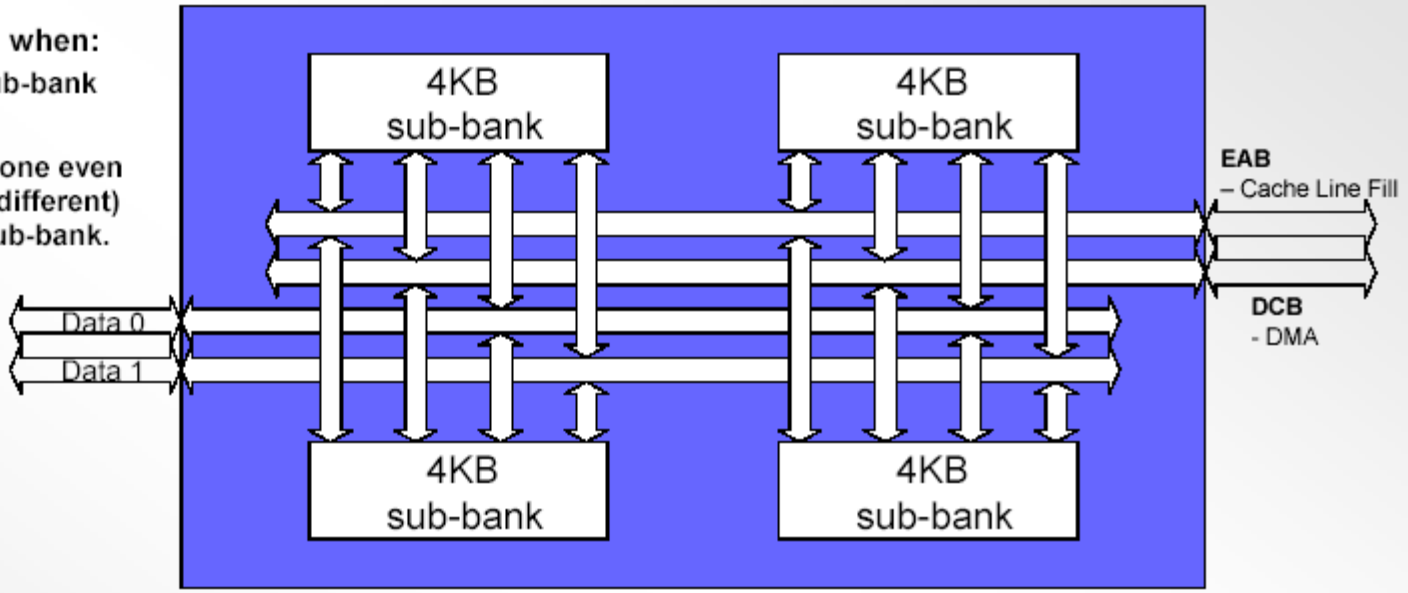
- Four 4KB single-ported sub-banks
- Allows simultaneous core and DMA accesses to different banks

16 KB cache

- 4-way set associative with arbitrary locking of ways and lines
- LRU replacement
- No DMA access

Mémoire L1 - Data

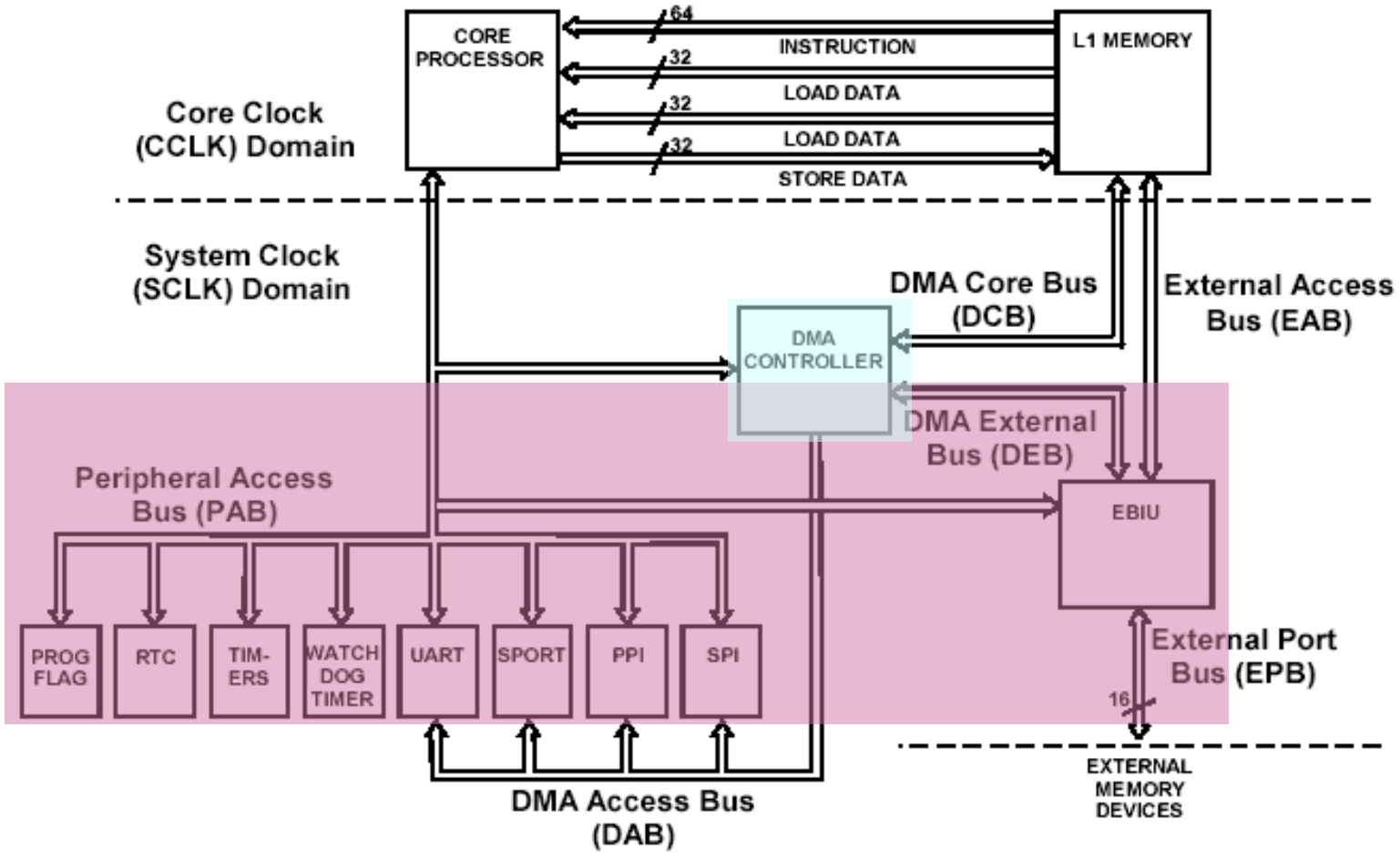
Block is Multi-ported when:
Accessing different sub-bank
OR
Accessing one odd and one even access (Addr bit 2 different) within the same sub-bank.



- When Used as SRAM
  - Allows simultaneous dual DAG and DMA access
- When Used as Cache
  - Each bank is 2-way set-associative
  - No DMA access
  - Allows simultaneous dual DAG access

Périphériques

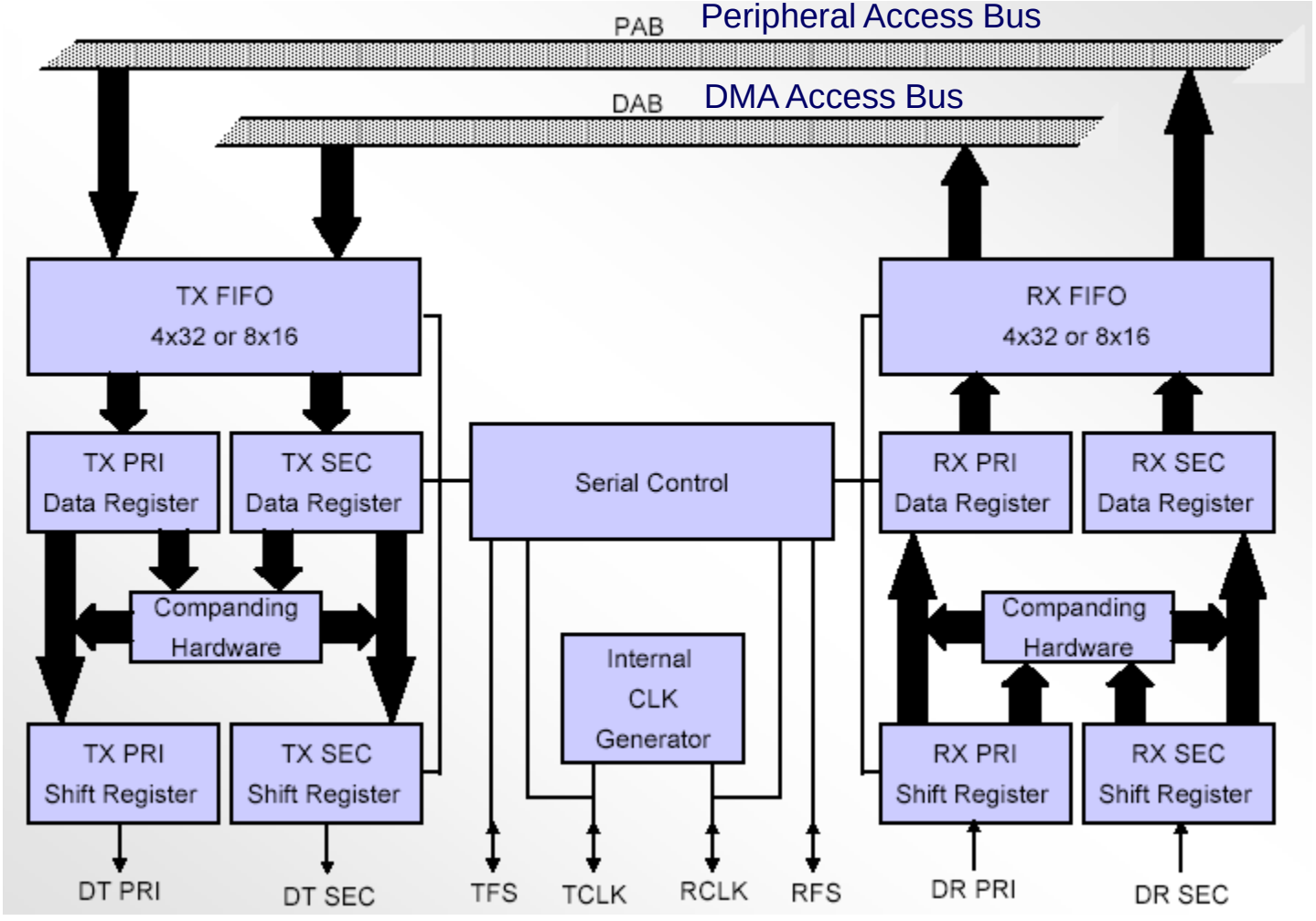
Architecture des périphériques



Périphériques communication série

- SPORTs (synchronous Serial PORTs)
  - High Speed (up to SCLK/2)
  - Two SPORTs (SPORT0 and SPORT1)
  - Typically used for interfacing with CODEC's and TDM data streams
- SPI (Serial Peripheral Interface)
  - Single High Speed SPI port (up to SCLK/4)
  - Typically used to interface with serial EPROMS, other CPUs, data converters, and displays
- UART (Universal Asynchronous Receiver/Transmitter)
  - Single PC-style UART port (baud rate up to SCLK/16)
  - Typically used for maintenance port, and interfacing with slow serial peripherals

Architecture ports série (SPORTs)



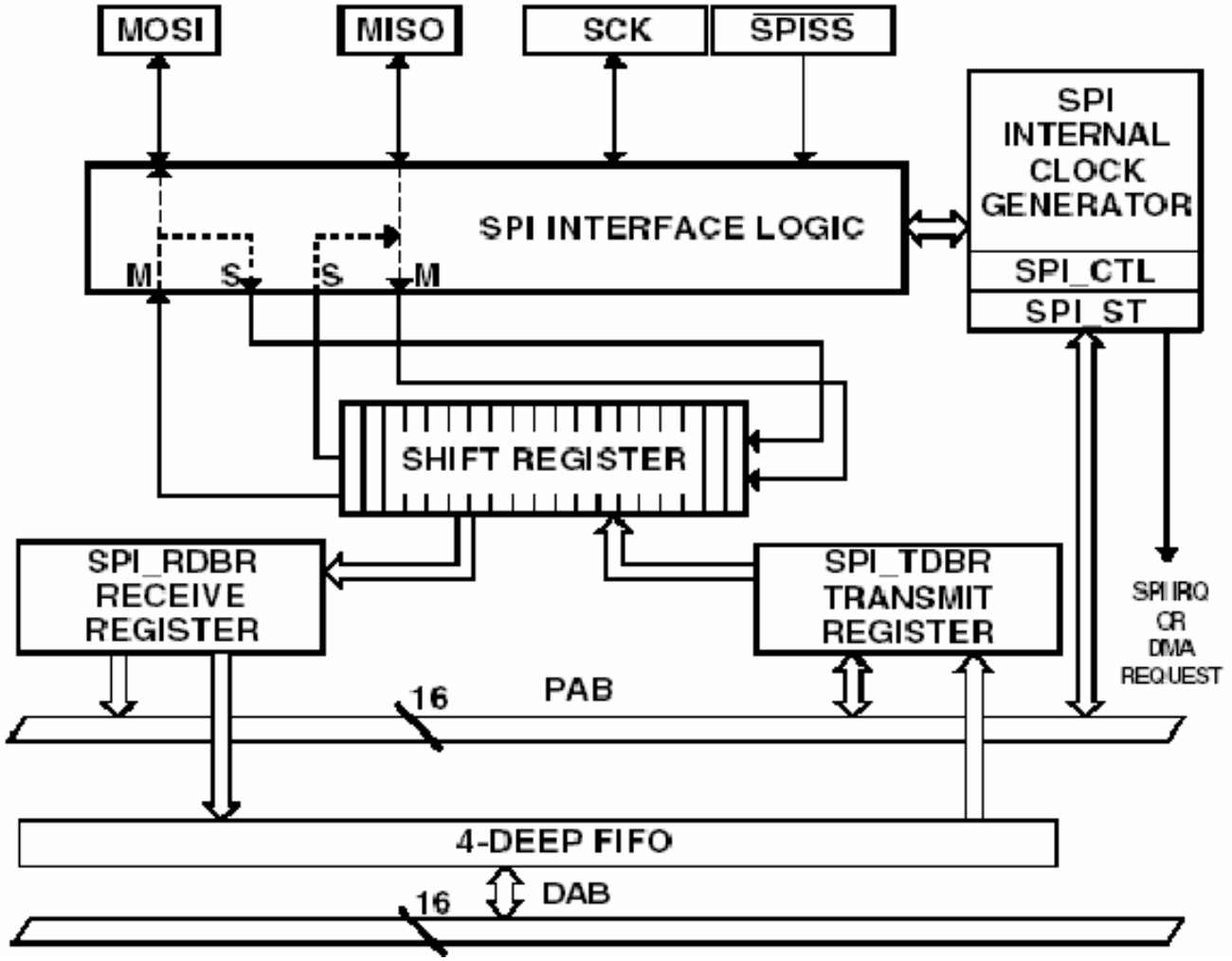
Fonctionnalités SPORTs

- Fully independent receive and transmit channels - double buffered
- Primary and Secondary Data RX/TX pins
- Support up to 32-bit serial words
- Internal or externally generated serial clocks and frame syncs
- Programmable internal/external frame syncs
- Built in hardware for u-law & A-law companding
- Support for multichannel interfaces
- I^2S signaling support
- Generates optional interrupts
- Separate Data and Error Interrupts
- Operates up to ½ System bus clock rate (SCLK)

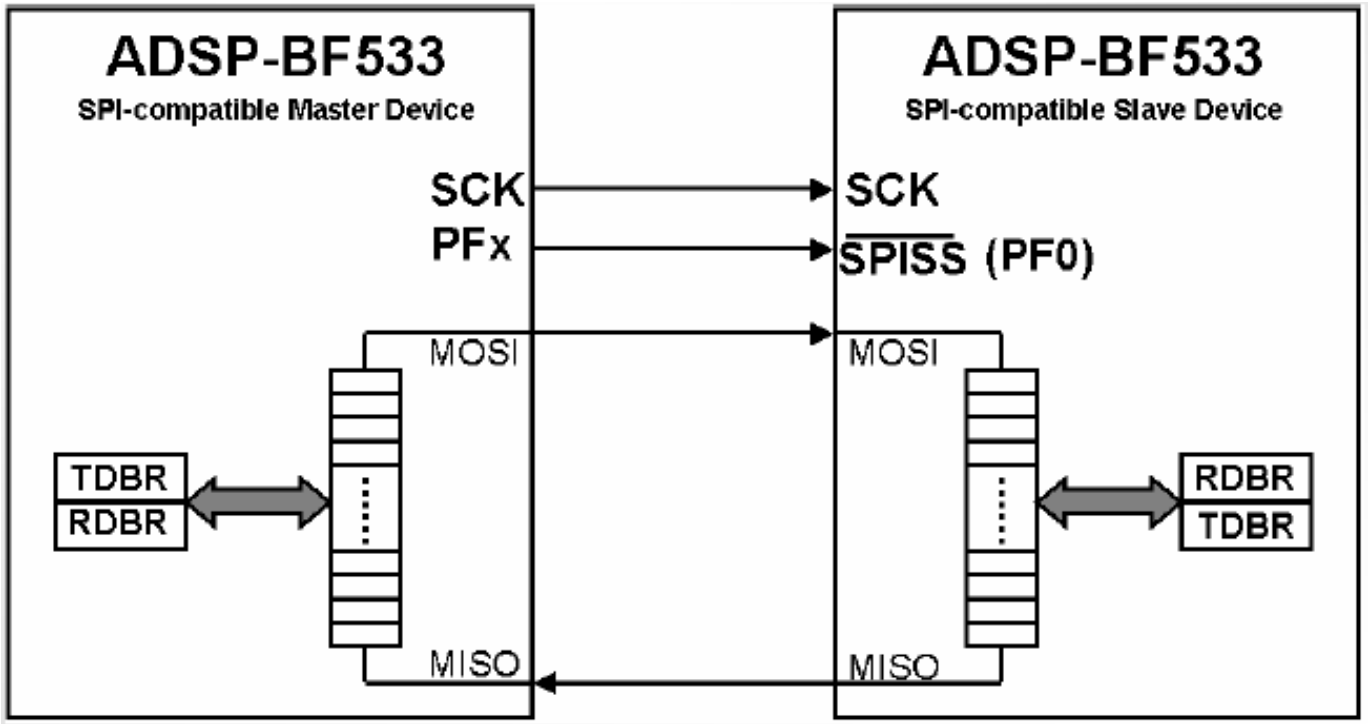
Fonctionnalités SPI

- One SPI-Compatible Port
- 4 Pin Interface (MOSI, MISO, ~SPISS, SCK)
- Master and Slave Mode Operation
  - Supports Multimaster Environments
- Can Use 8 GP Flag Pins As Slave-Select Lines
  - 1 Slave Select *Input* Pins
  - 7 Slave Select *Output* Pins
- Gated SPI Clock (Only Active During Transfers)
- DMA Support
  - One DMA Channel (Transmit or Receive)
- Programmable Baud Rate
- Programmable Clock Polarity and Phase
- Programmable Serial Word Length (8 or 16 Bits)

Architecture SPI



Application SPI



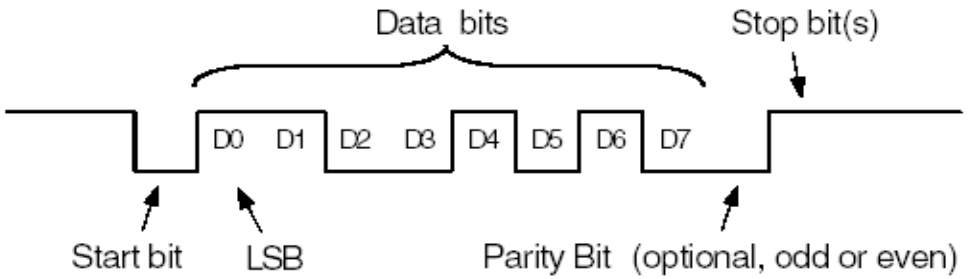
MOSI: Master Output Slave In

MISO: Master Input Slave Out

SPISS: Serial Peripheral Interface Slave Select

Fonctionnalités UART

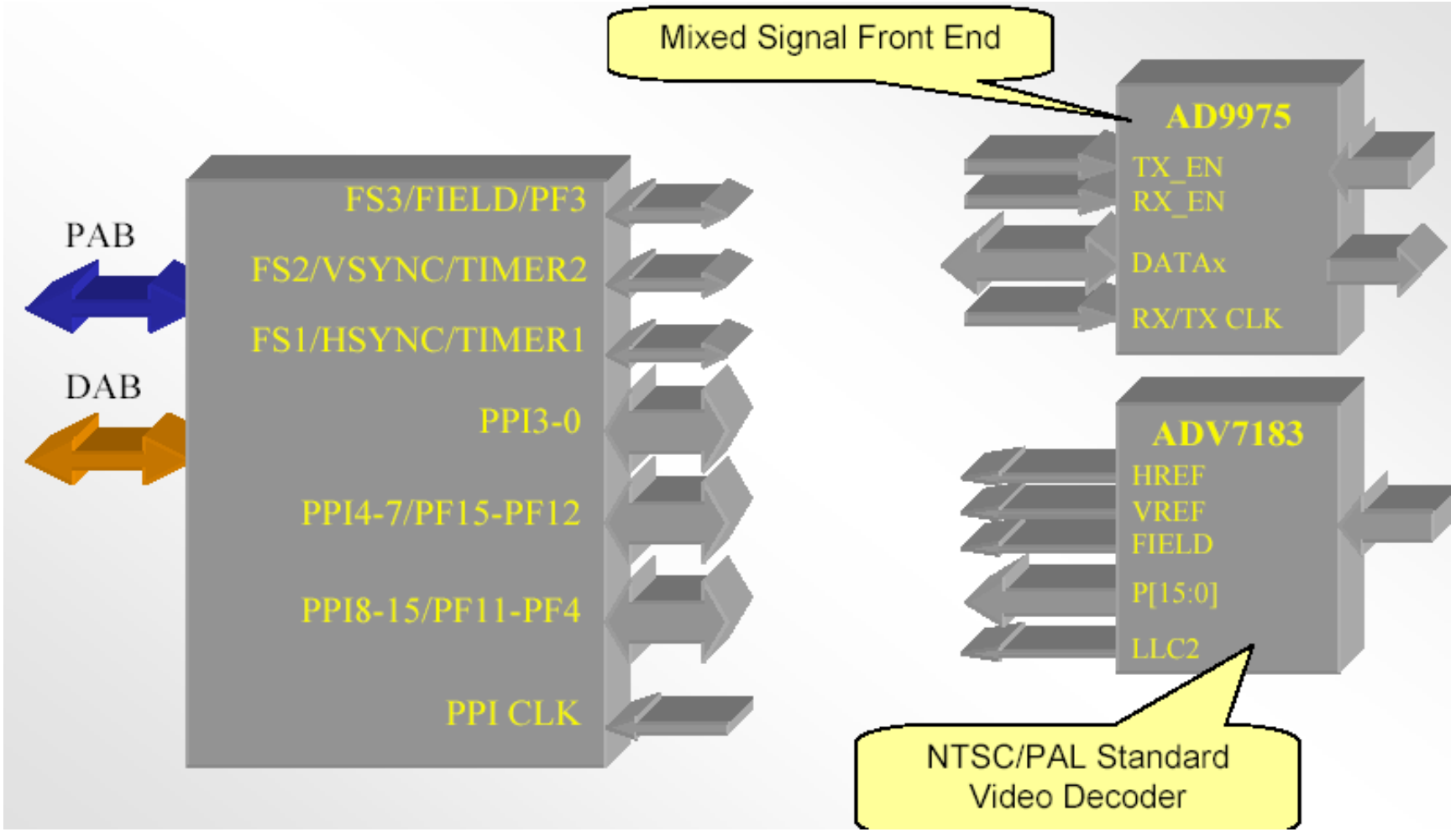
- One UART module
- Industrial Standard 16450 compliant
  - 5-8 data bits
  - 1, 1½ or 2 stop bits
  - None, even or odd parity
  - Baud rate = SCLK/(16*DIVISOR)
  - Loopback mode
- Supports half-duplex IrDA SIR (9.6/115.2 Kbps rate)
- Autobaud detection support through the use of the Timers
- Separate TX and RX DMA support (Register-based and Descriptor-based)



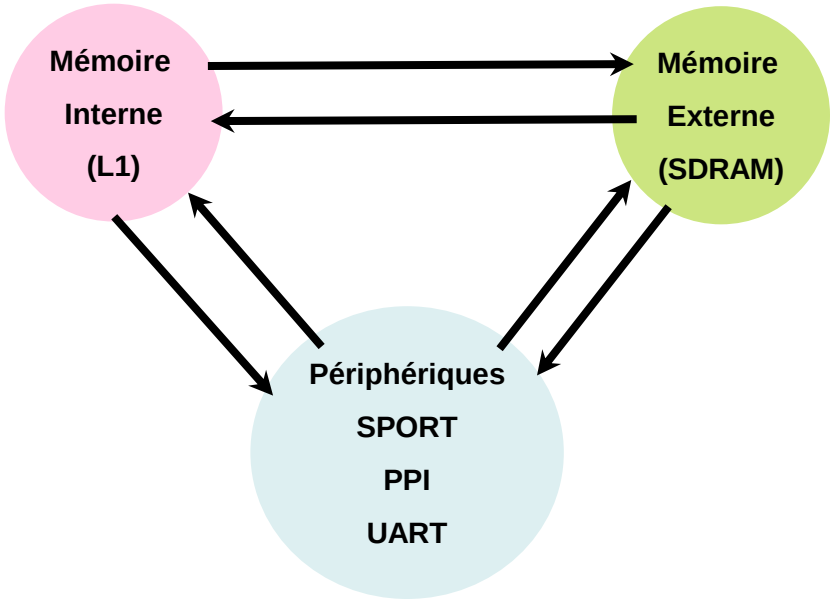
Fonctionnalités PPI

- Parallel Peripheral Interface
  - Programmable bus width (from 8 – 16 bits in 1-bit steps)
  - Bidirectional (half-duplex) parallel interface
  - Synchronous Interface
    - Interface is driven by an external clock (“PPI_CLK”)
    - Up to 66MHz rate (SCLK/2)
    - Asynchronous to SCLK
  - Includes three frame syncs to control the interface timing
  - Applications
    - High speed data converters
    - Video CODECs
- Used in conjunction with a DMA channel
  - Can setup 2D DMA (e.g., for video)
  - Can pack 8-bit bytes into 16-bit words for efficient I/O

Applications PPI



DMA: Direct Memory Access



- Le contrôleur DMA permet le transfert de données sans l'intervention du processeur
- Le core met à jour les registres et répond aux interruptions quand les données sont prêtes
- Transfert 1-D
- Transfert 2-D avec différentes tailles de colonnes et lignes et différents pas

Types of data transfers

Internal or External Memory	←→	Internal or External Memory
Internal or External Memory	←→	Serial Peripheral Interface (SPI)
Internal or External Memory	←→	Serial Port
Internal or External Memory	←→	UART Port
Internal or External Memory	←→	Parallel Port Interface (PPI)

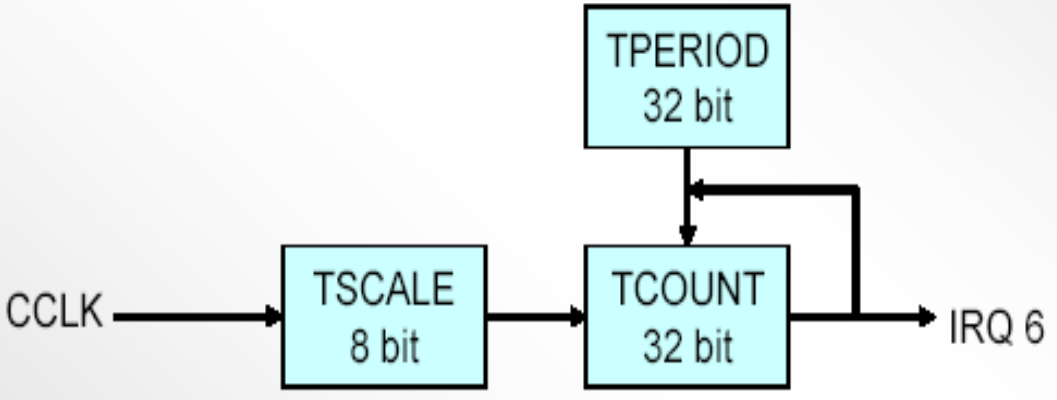
Timers

6 Timers (BF533)

- Un Core Timer
- Un Watchdog Timer
- Un Real-Time Clock (RTC)
- Trois 32-bit General Purpose Timers:
  - Pulse and Capture Mode
  - PWM Mode
  - External Clock Mode

Core Timer

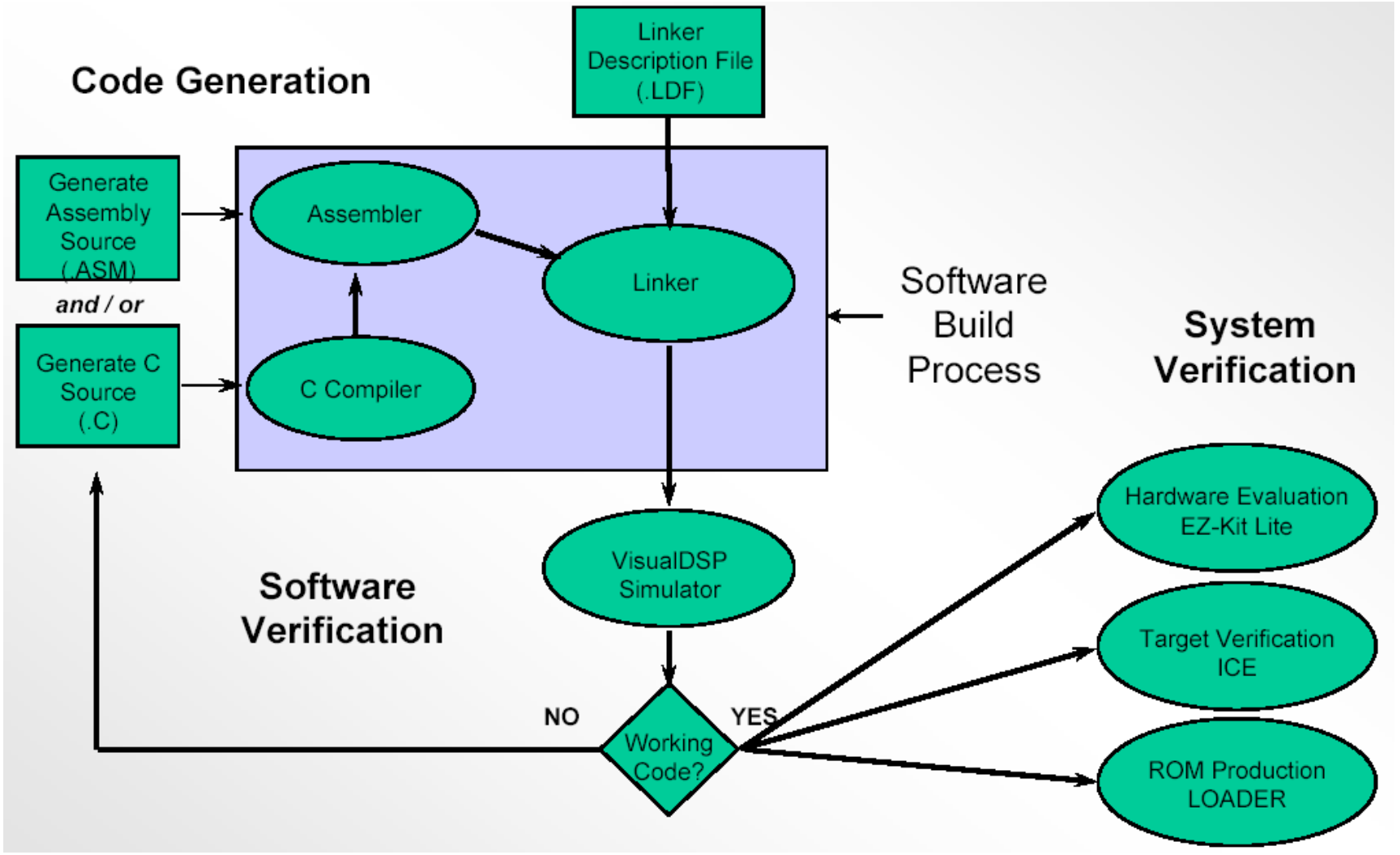
- Generates interrupts at multiples of CCLK rate
  - 32-bit tick timer
- Dedicated Interrupt Priority 6 (IRQ6 = IVTMR)
- Optional Auto-Reload Feature



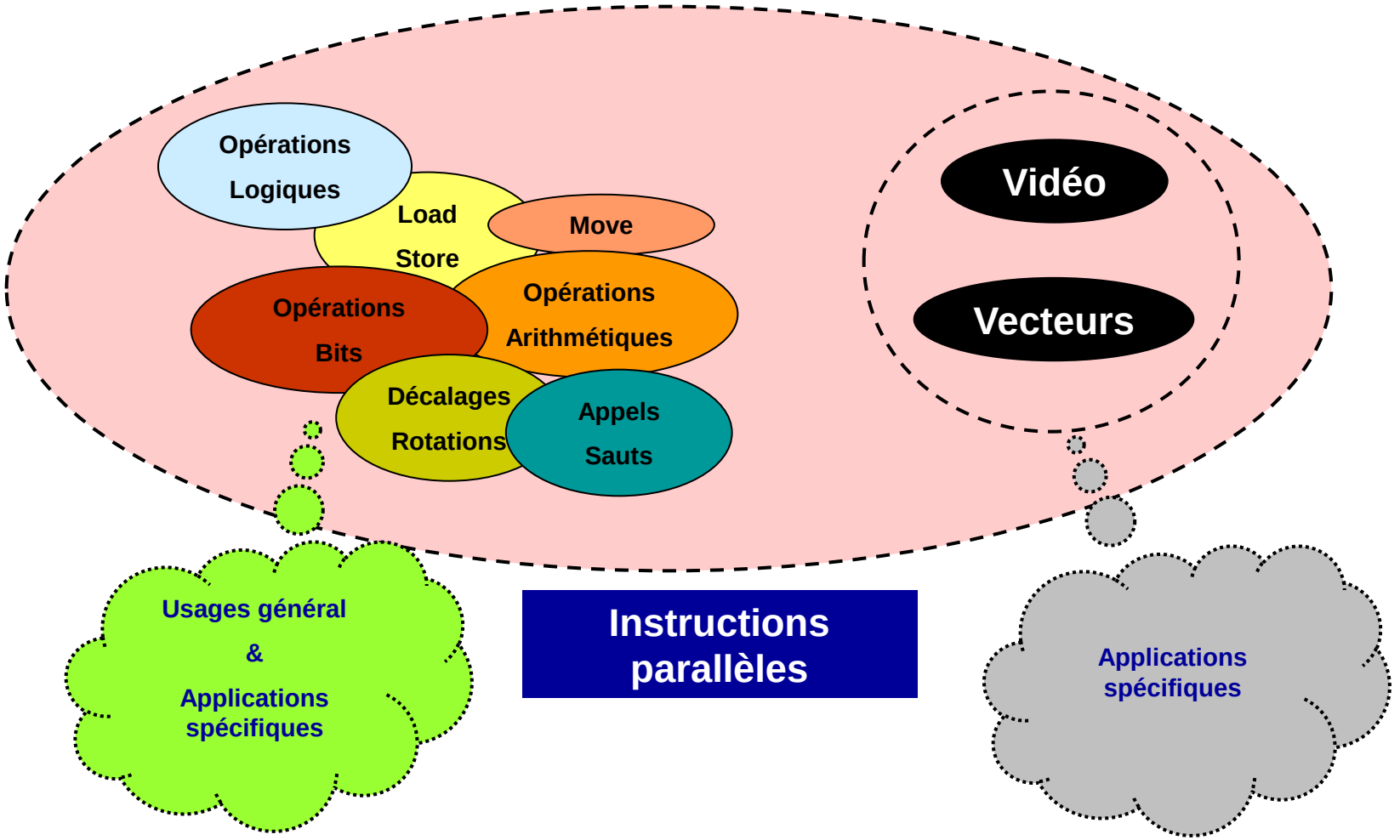
Interrupt Rate = CCLK / [(TSCALE + 1) x TPERIOD]

Jeu d'instructions

Étapes de développement logiciel



Classes d'instructions



Exemple 1: *génération d'un flux série*

BITMUX (source_1, source_0, A0) (ASR); (Right Shift)

BITMUX (source_1, source_0, A0) (ASL); (Left Shift)

In the Shift Right version, the processor performs the following sequence:

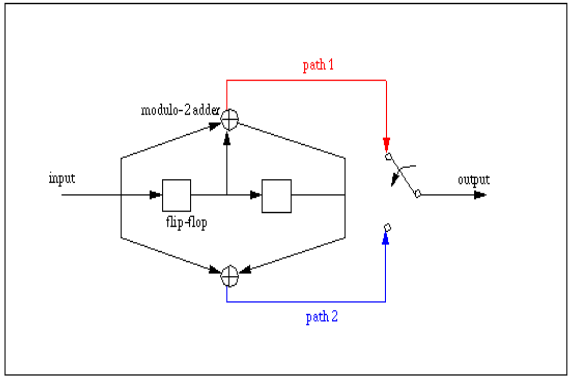
1. Right shift Accumulator A0 by one bit. Right shift the LSB of source_1 into the MSB of the Accumulator.

2. Right shift Accumulator A0 by one bit. Right shift the LSB of source_0 into the MSB of the Accumulator.

IF	39.....32	31.....24	23.....16	15.....8	7.....0
source_1:		xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx
source_0:		yyyy yyyy	yyyy yyyy	yyyy yyyy	yyyy yyyy
Accumulator A0:	zzzz zzzz	zzzz zzzz	zzzz zzzz	zzzz zzzz	zzzz zzzz

After « Right Shift »

IF	39.....32	31.....24	23.....16	15.....8	7.....0
source_1:[1]		0xxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx
source_0:[2]		0yyy yyyy	yyyy yyyy	yyyy yyyy	yyyy yyyy
Accumulator A0:[3]	yxzz zzzz	zzzz zzzz	zzzz zzzz	zzzz zzzz	zzzz zzzz

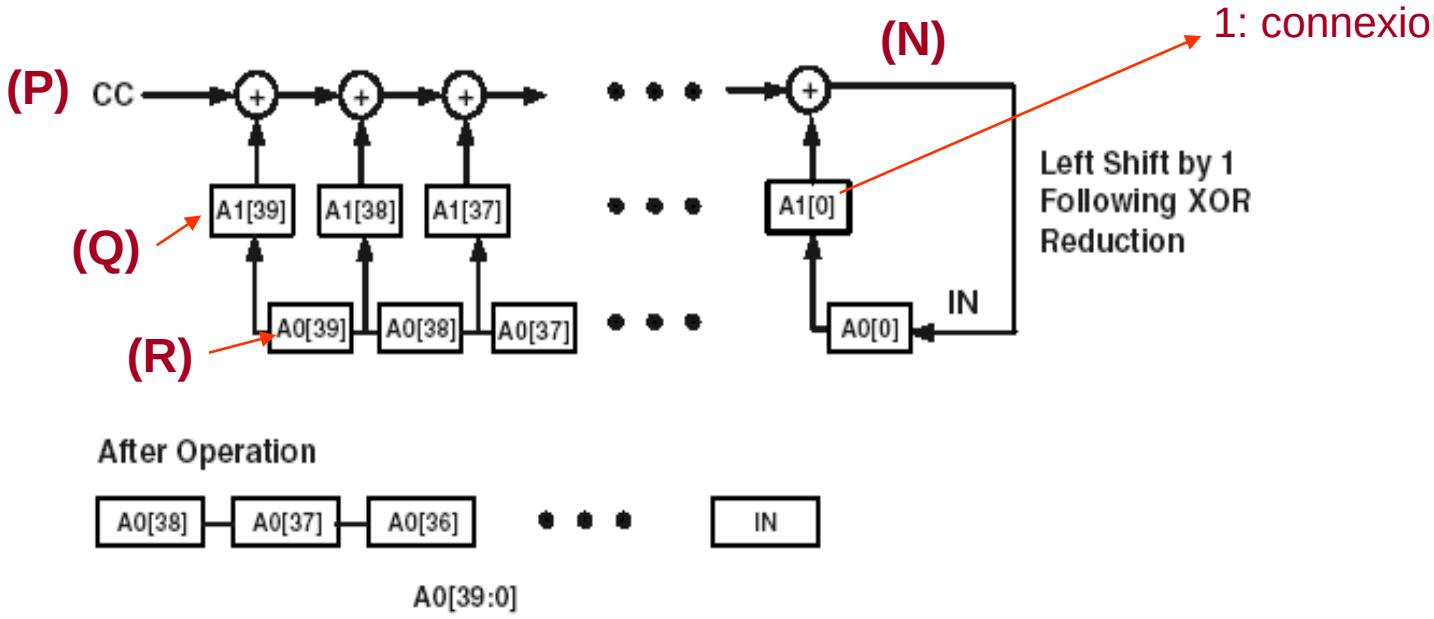


Codeur convolutif (5,7)

Exemple 2: *LFSR*

A0 = BXORSHIFT (A0, A1, CC) ;

Implémentation des Linear Feedback Shift Registers



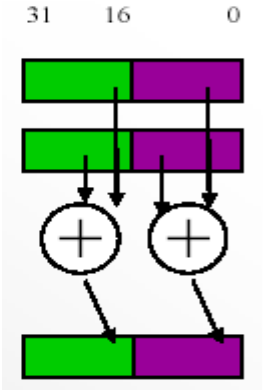
Applications

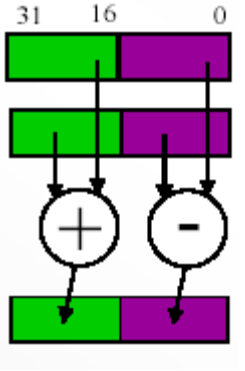
- **Division plynômiale: P/Q = N+R**
- **Codeurs et décodeurs cycliques**

Instructions sur les vecteurs

- Instructions qui manipulent plusieurs données 16 bits en même temps
- Opérations supportées: Addition, soustraction, multiplications, décalages,…

```
r5=r3 +|+ r4 ; /* dual 16-bit operations, add|add */
r6=r0 -|+ r1(s) ; /* same as above, subtract|add with saturation */
r5=r3 +|- r4, r7=r3 -|+ r4 ; /* 4 additions in the same time */
r7 = max (r1, r0) (v) ;
```





Instructions parallèles

- ***32-bit ALU/MAC instruction* || *16-bit instruction* || *16-bit instruction* ;**
- ***32-bit ALU/MAC instruction* || *16-bit instruction* ;**

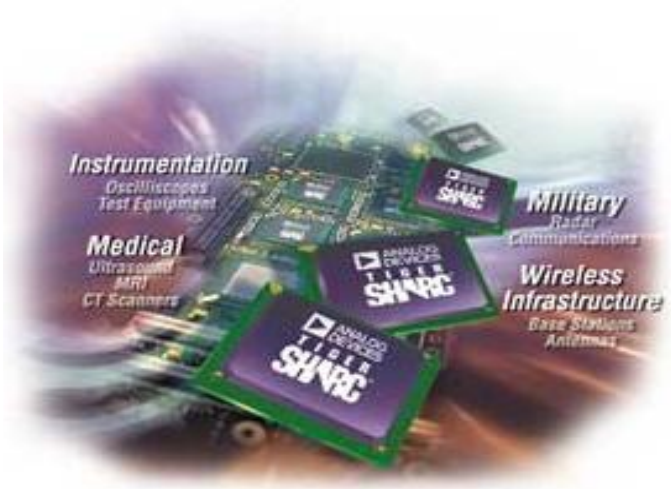
/* Multiply and accumulate while loading two data registers. One load uses an Ireg pointer. */

A1+=R0.L*R2.H,A0+=R0.L*R2.L || R2.L=W[I2++] || R0=[I1--] ;

R3.H=(A1+=R0.L*R1.H), R3.L=(A0+=R0.L*R1.L) || R0=[P0++] || R1=[I0] ;

/* Multiply-Accumulate to a Data register while incrementing an Ireg. */

r6=(a0+=r3.h*r2.h)(fu) || i2-=m0 ;



Fin
de la 2ème partie…