ARCHITETTURA DEGLI ELABORATORI

A.A. 2020-2021

Università di Napoli Federico II
Corso di Laurea in Informatica

Docenti
Proff. Luigi Sauro gruppo 1 (A-G)
Silvia Rossi gruppo 2 (H-Z)

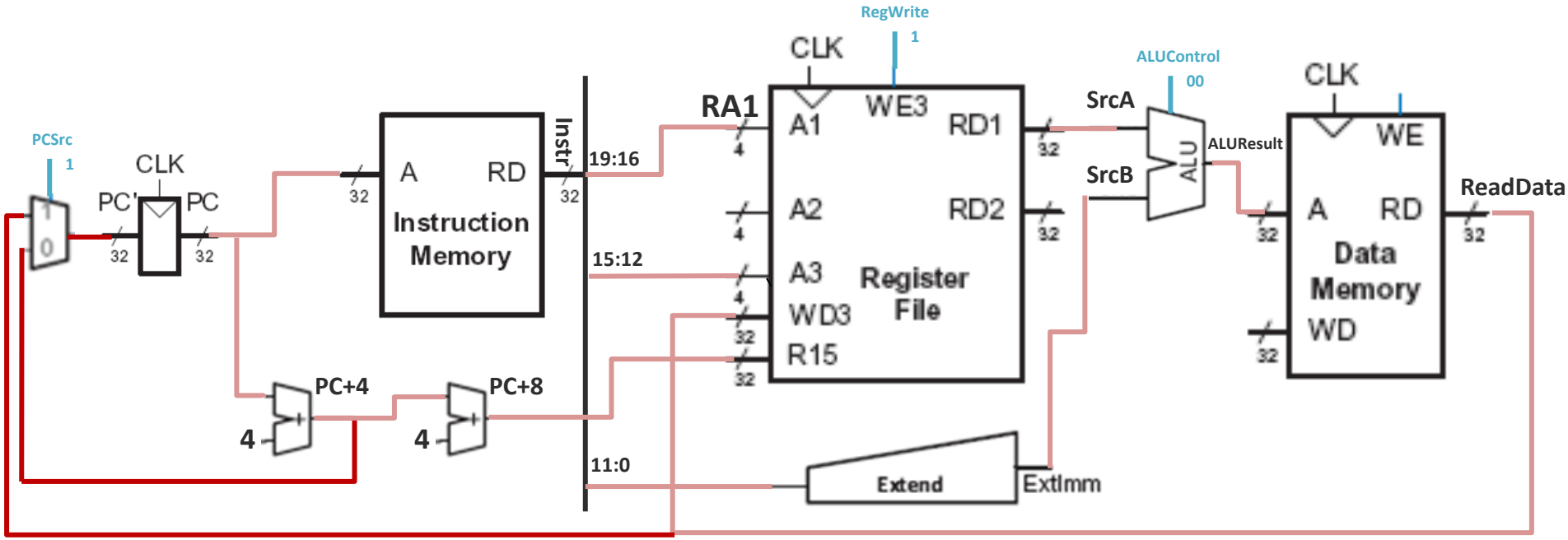
MICROARCHITETTURA ARM

Datapath LDR

Infine, trattandosi di uno stored program paradigm, l'istruzione successiva potrebbe essere letta anche dalla memoria e quindi corrispondere al contenuto di **ReadData**. Un **multiplexer**, permette di selezionare fra:

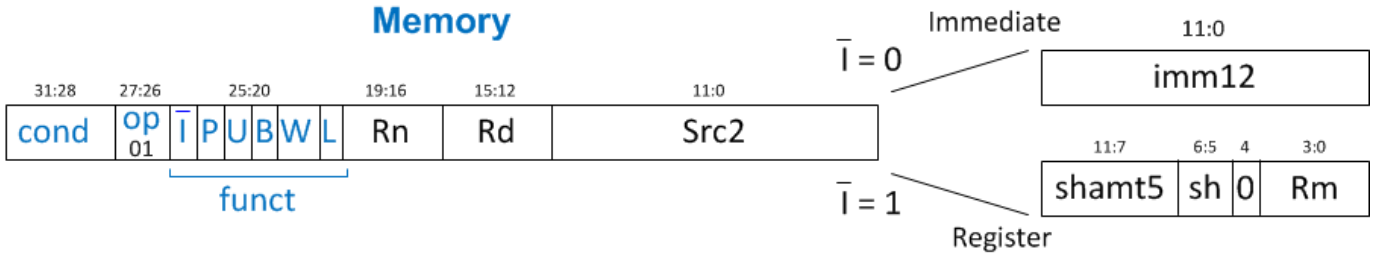
- 0 – **PCPlus4**
- 1 – **ReadData**.

Il segnale di controllo associato al multiplexer è **PCSrc**.



Datapath STR

L'istruzione **STR** scrive una parola di 32 bit contenuta in un registro nella memoria centrale. Il modo in cui questa operazione viene effettuata dipende dalla politica di indirizzamento specificata.

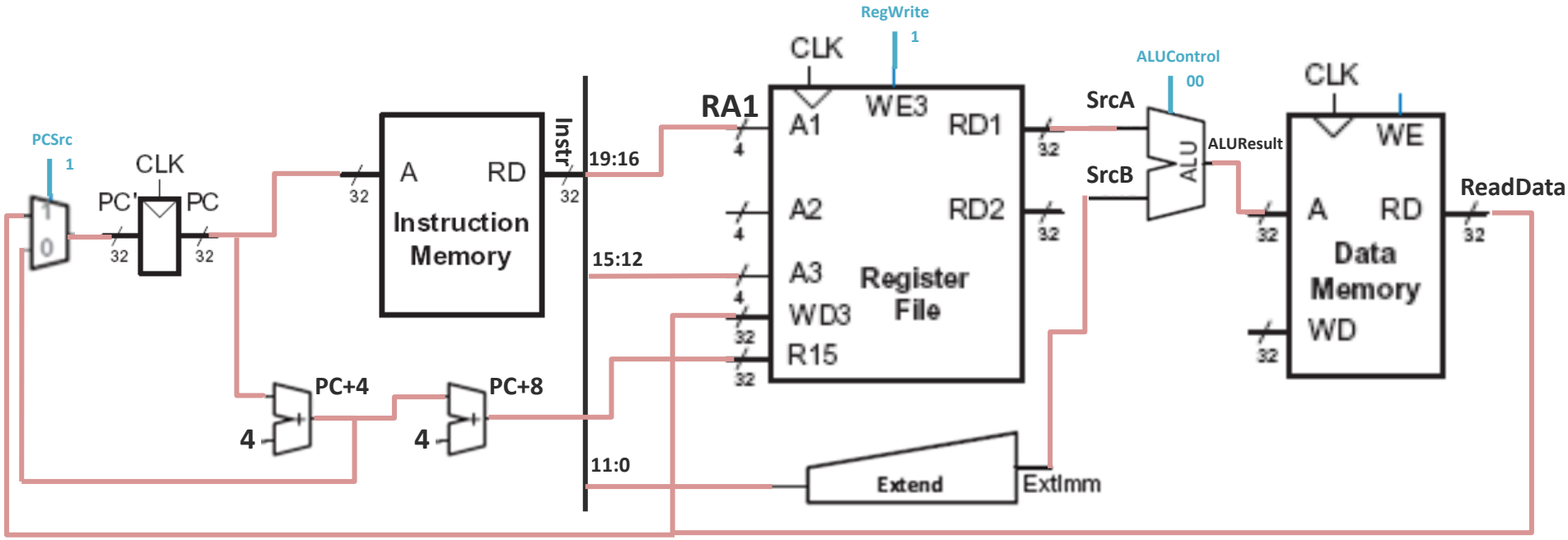


`STR Rd, [Rn, imm12]`

Datapath STR

Estendiamo il **datapath** in modo da poter gestire anche l'istruzioni **STR**.

Come LDR, **STR** legge un indirizzo di base dalla porta **A1** del register file e completa l'immediate. L'**ALU** aggiunge l'indirizzo di base alla costante per trovare l'indirizzo di memoria. Tutte queste funzioni sono già supportate nel datapath.

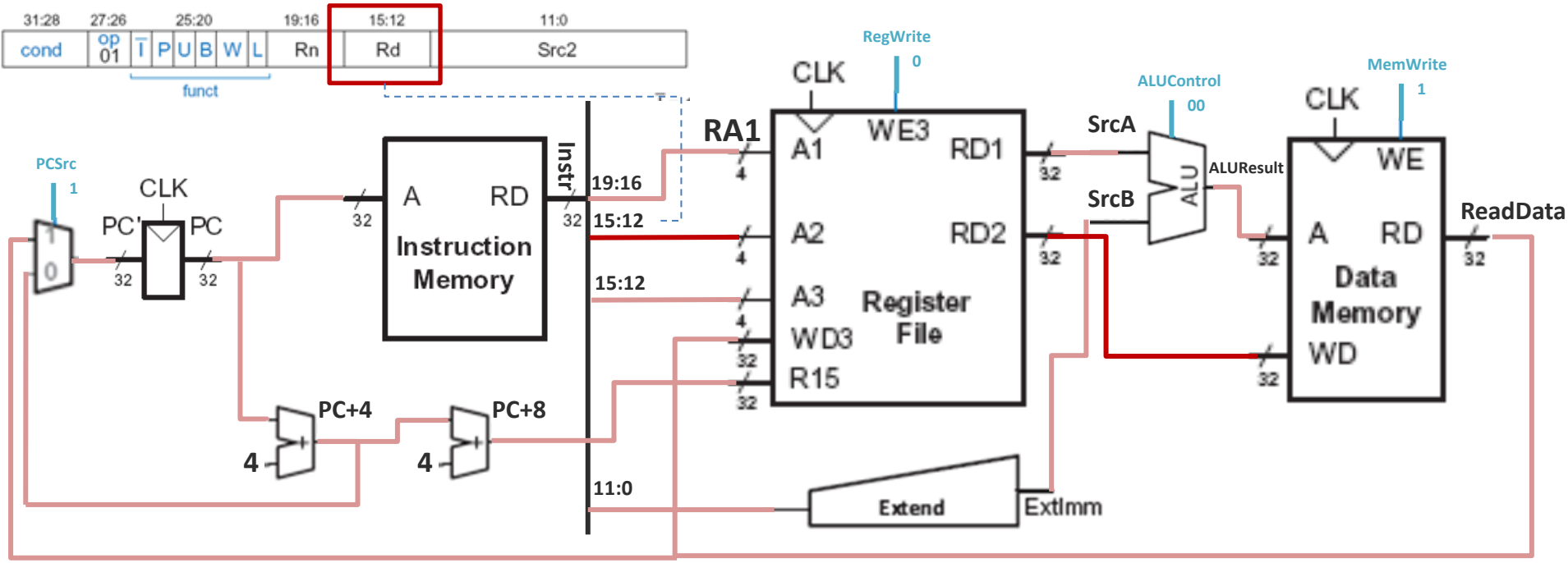


Datapath STR

Il registro è specificato nel campo **Rd**, **Instr$_{15:12}$**, che è collegato alla porta **A2** del register file.

Il valore del registro viene letto sulla porta **RD2**, che è collegata alla porta dati di scrittura (**WD**) della memoria dati.

L'abilitazione del segnale di scrittura **WE** è controllato da **MemWrite**, il quale è 1 se i dati devono essere scritti in memoria.

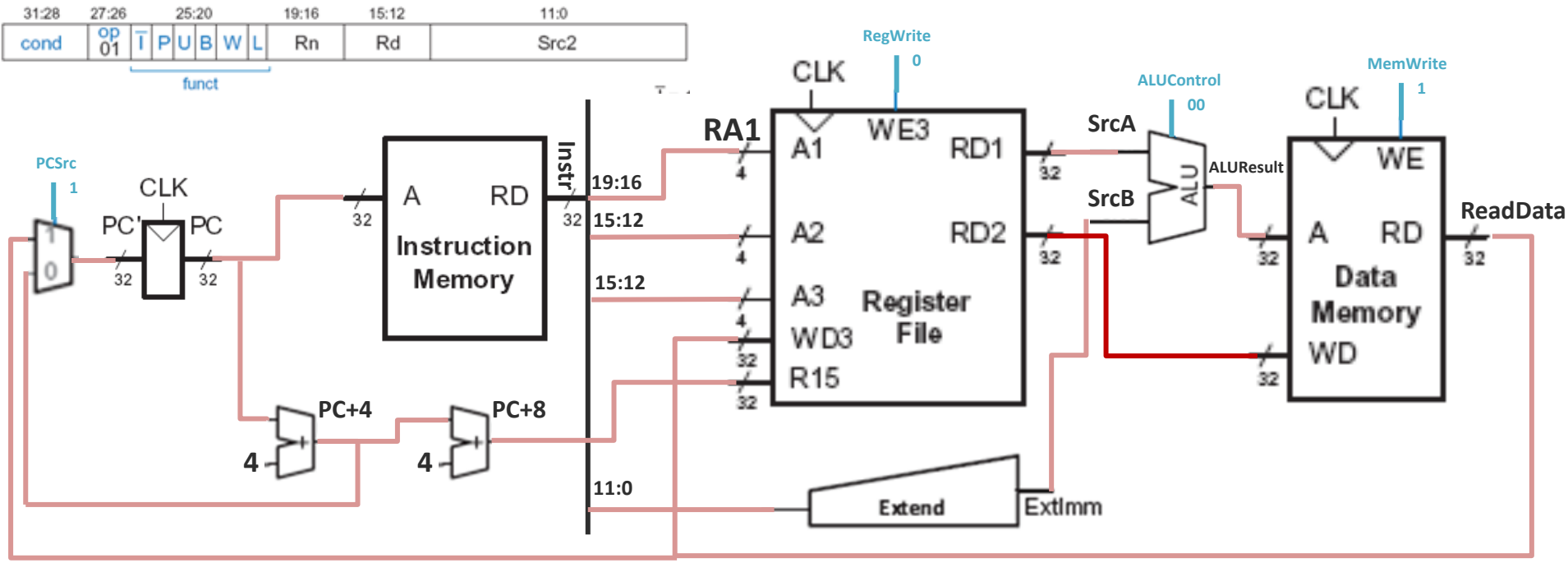


Datapath STR

Il segnale **ALUControl** deve essere impostato a **00** per sommare l'indirizzo di base e l'offset.

Il segnale **RegWrite** è impostato a **0**, perché nulla deve essere scritto nel register file.

Si noti che, pur essendo **Rd** associato anche a A3 e **ReadData** a WD3, questo non produce effetti sul File register, essendo **RegWrite** impostato a **0**.



Istruzioni di data processing

MOV	Move a 32-bit value	MOV Rd,n	Rd = n
MVN	Move negated (logical NOT) 32-bit value	MVN Rd,n	Rd = ~n
ADD	Add two 32-bit values	ADD Rd,Rn,n	Rd = Rn+n
ADC	Add two 32-bit values and carry	ADC Rd,Rn,n	Rd = Rn+n+C
SUB	Subtract two 32-bit values	SUB Rd,Rn,n	Rd = Rn-n
SBC	Subtract with carry of two 32-bit values	SBC Rd,Rn,n	Rd = Rn-n+C-1
RSB	Reverse subtract of two 32-bit values	RSB Rd,Rn,n	Rd = n-Rn
RSC	Reverse subtract with carry of two 32-bit values	RSC Rd,Rn,n	Rd = n-Rn+C-1
AND	Bitwise AND of two 32-bit values	AND Rd,Rn,n	Rd = Rn AND n
ORR	Bitwise OR of two 32-bit values	ORR Rd,Rn,n	Rd = Rn OR n
EOR	Exclusive OR of two 32-bit values	EOR Rd,Rn,n	Rd = Rn XOR n
BIC	Bit clear. Every '1' in second operand clears corresponding bit of first operand	BIC Rd,Rn,n	Rd = Rn AND (NOT n)
CMP	Compare	CMP Rd,n	Rd-n & change flags only
CMN	Compare Negative	CMN Rd,n	Rd+n & change flags only
TST	Test for a bit in a 32-bit value	TST Rd,n	Rd AND n, change flags
TEQ	Test for equality	TEQ Rd,n	Rd XOR n, change flags

MUL	Multiply two 32-bit values	MUL Rd,Rm,Rs	Rd = Rm*Rs
MLA	Multiple and accumulate	MLA Rd,Rm,Rs,Rn	Rd = (Rm*Rs)+Rn

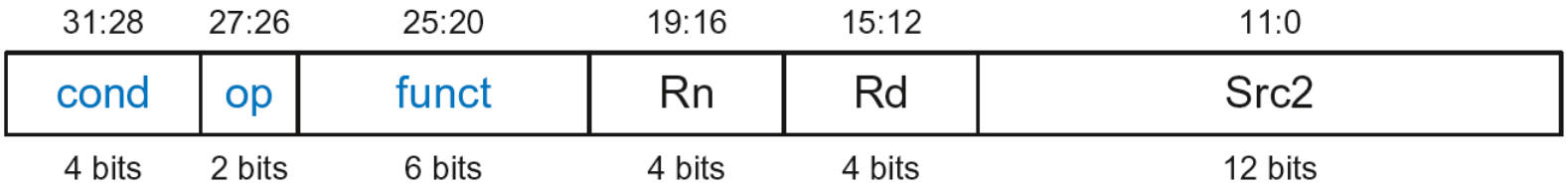
N. Mathivanan

Data-processing Instruction Format

Il formato delle istruzioni di **data-processing** è il più comune.
Il primo operando sorgente è un registro. Il secondo operando sorgente può essere una costante o un registro. La destinazione è un registro.

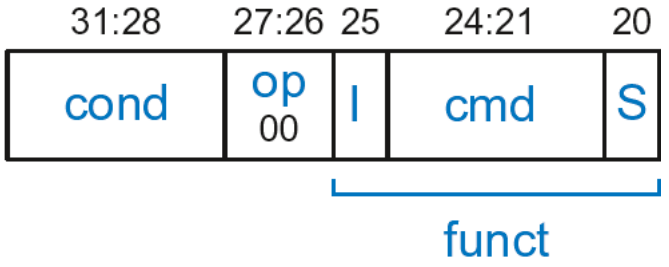
- **Operandi:**
  - ***Rn*:** primo registro sorgente
  - ***Src2*:** secondo registro sorgente o immediate
  - ***Rd*:** registro destinazione
- **Campi di controllo:**
  - ***cond*:** specifica l'esecuzione condizionale in base ai flag
  - ***op*:** 00 è l'opcode per istruzioni di data processing
  - ***funct*:** specifica il tipo di funzione da eseguire

Data-processing



Data-processing Control Fields

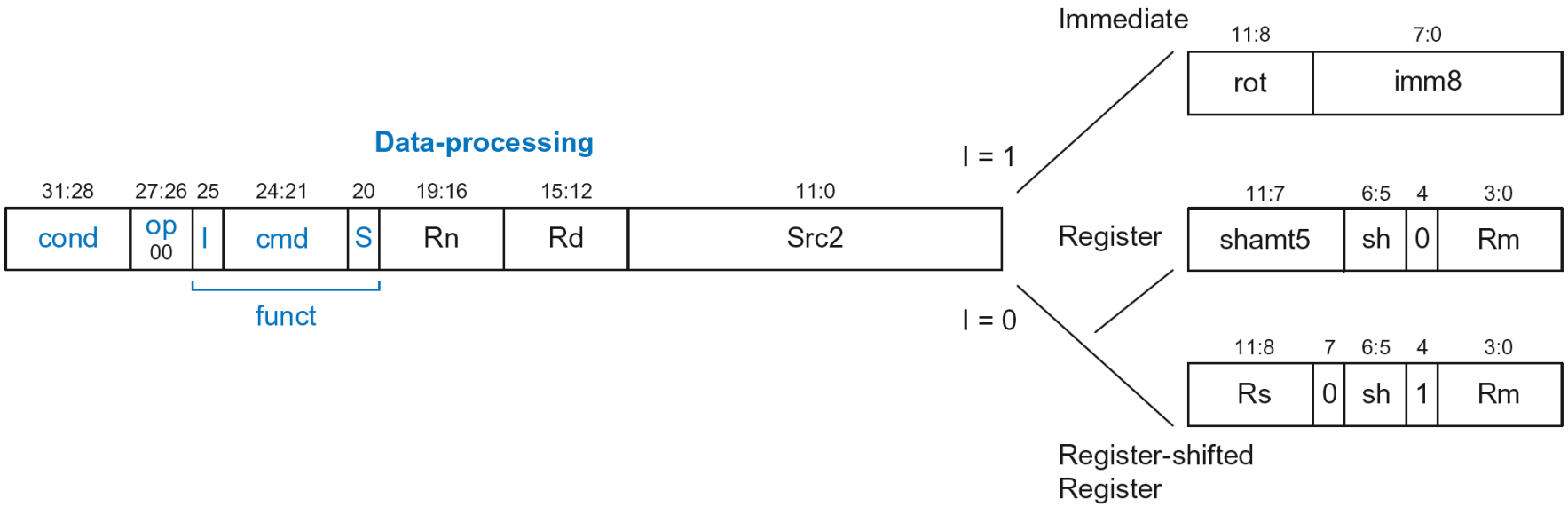
- ***cmd:*** indica l'istruzione specifica da eseguire
  - ***cmd*** = 0100_2 sta per `ADD`
  - ***cmd*** = 0010_2 sta per `SUB`
- ***I***-bit
  - ***I*** = 0: *Src2* è un registro
  - ***I*** = 1: *Src2* è un immediate



- ***S***-bit: quando è 1 allora vengono aggiornate le condition flags
  - ***S*** = 0: `SUB  R0, R5, R7`
  - ***S*** = 1: `ADDS R8, R2, R4` or `CMPS R3, #10`

Data-processing *Src2* Variations

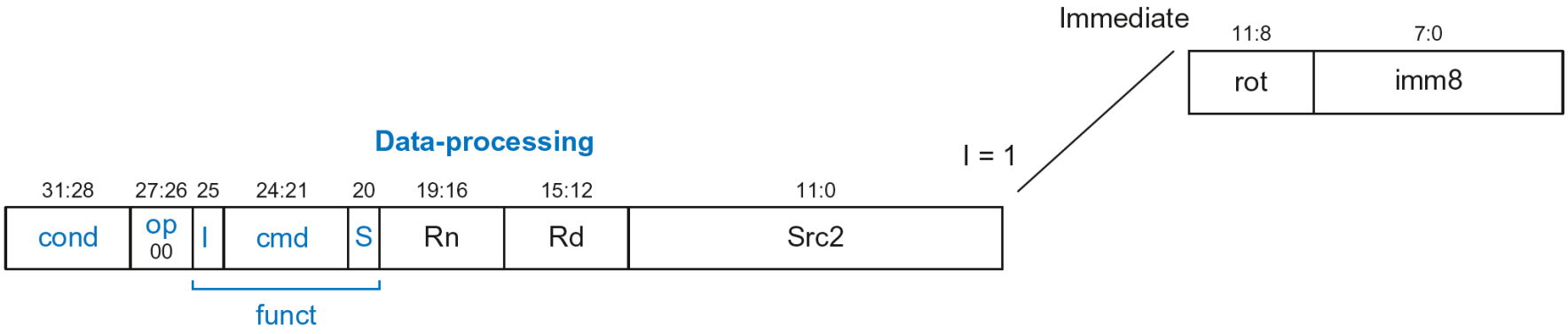
- *Src2* può essere:
  - Immediate
  - Registro
  - Registro “shiftato” da un altro registro



Immediate *Src2*

- **Un immediate è codificato come segue:**
  - *imm8*: 8-bit unsigned immediate
  - *rot*: 4-bit rotation value
- **Da cui si ricava una constante a 32-bit :**

  imm8 **ROR** (*rot* × 2)



Immediate *Src2*

- **Un immediate è codificato come segue:**
  - *imm8*: 8-bit unsigned immediate
  - *rot*: 4-bit rotation value
- **Da cui si ricava una constant a 32-bit :**

  imm8 **ROR** (*rot* × 2)

Esempio: *imm8* = 10001111

rot	32-bit constant
0000	0000 0000 0000 0000 0000 0000 1000 1111
0001	1100 0000 0000 0000 0000 0000 0010 0011
0010	1111 0000 0000 0000 0000 0000 0000 1000
…	…
1111	0000 0000 0000 0000 0000 0010 0011 1100

DP Instruction with Immediate *Src2*

```
ADD R0, R1, #42
```

cond = 1110_2 (14) indica un'esecuzione non condizionata
op = 00_2 (0) indica un'istruzione di data-processing
cmd = 0100_2 (4) è il codice di ADD
I = 1 indica che ***Src2*** è un immediate, ***S***= 0,
Rd = 0, ***Rn*** = 1
imm8 = 42, ***rot*** = 0

Field Values

31:28	27:26	25	24:21	20	19:16	15:12	11:8	7:0
1110_2	00_2	1	0100_2	0	1	0	0	42
cond	op	I	cmd	S	Rn	Rd	shamt5	sh Rm
1110	00	1	0100	0	0001	0000	0000	00101010

0xE281002A

DP Instruction with Immediate *Src2*

```
SUB R2, R3, #0xFF0
```

cond = 1110_2 (14)
op = 00_2 (0)
cmd = 0010_2 (2) è il codice di SUB
I=1, ***S***=0
Rd = 2, ***Rn*** = 3
imm8 = 0xFF
per produrre 0xFF0 ***imm8*** deve essere ruotato di 4 bit a sinistra, ovvero 28 bit a destra. Quindi, ***rot*** = 14

Field Values

31:28	27:26	25	24:21	20	19:16	15:12	11:8	7:0
1110_2	00_2	1	0010_2	0	3	2	14	255
cond	op	I	cmd	S	Rn	Rd	rot	imm8
1110	00	1	0010	0	0011	0010	1110	11111111

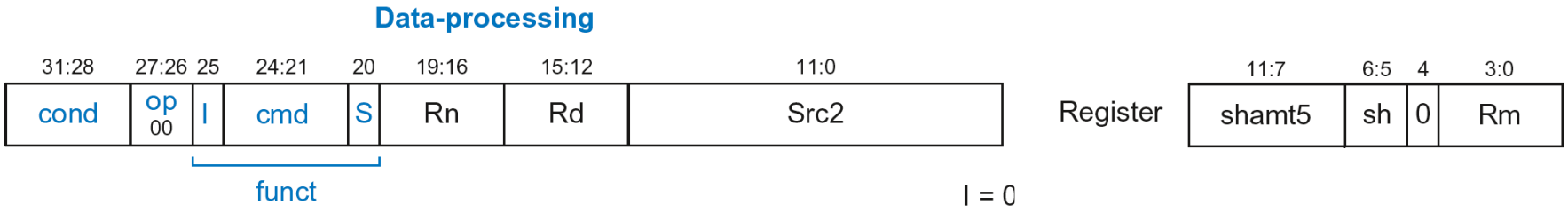
0xE2432EFF

DP Instruction with Register *Src2*

Rm: indica il registro che fa da secondo operado

shamt5: indica di quanto il valore in Rm è shiftato

sh: indica il tipo di shift (i.e., >>, <<, >>>, ROR)



Shift Type	*sh*
LSL	00_2
LSR	01_2
ASR	10_2
ROR	11_2

DP Instruction with Register *Src2*

```
ADD R5, R6, R7
```

Operation: R5 = R6 + R7
cond = 1110_2 (14)
op = 00_2 (0)
cmd = 0100_2 (4)
I=0 indica che ***Src2*** è un registro
Rd = 5, ***Rn*** = 6, ***Rm*** = 7
shamt = 0, ***sh*** = 0

Field Values

31:28	27:26	25	24:21	20	19:16	15:12	11:7	6:5	4	3:0
1110_2	00_2	0	0100_2	0	6	5	0	0	0	7
cond	op	I	cmd	S	Rn	Rd	shamt5	sh		Rm
1110	00	0	0100	0	0110	0101	00000	00	0	0111

0xE0865007

DP Instruction with Register *Src2*

`ORR R9, R5, R3,` **`LSR #2`**

Operation: R9 = R5 OR (R3 >> 2)

cond = 1110_2 (14)

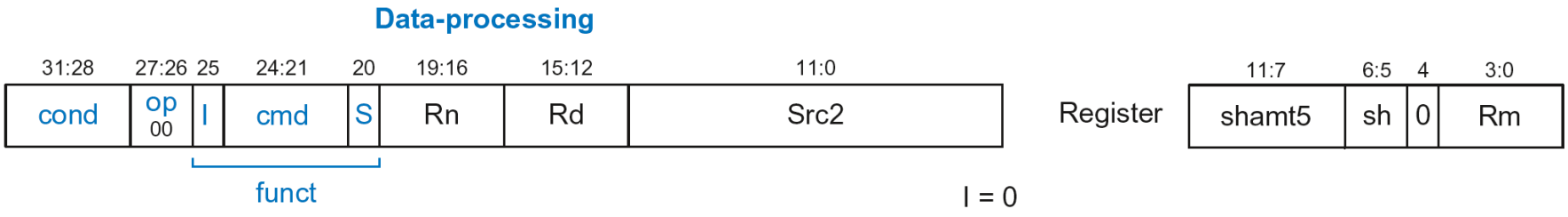
op = 00_2 (0)

cmd = 1100_2 (12) indica l'istruzione `ORR`

I=0

Rd = 9, ***Rn*** = 5, ***Rm*** = 3

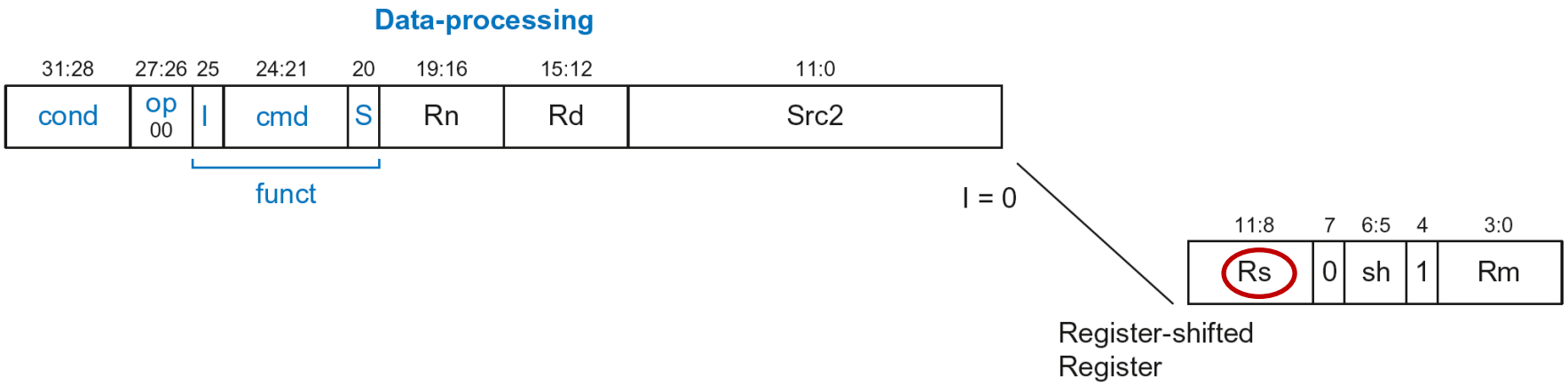
shamt5 = 2, ***sh*** = 01_2 (LSR)



1110 00 0 1100 0 0101 1001 00010 01 0 0011

0xE1859123

DP with Register-shifted Reg. *Src2*



Simile al formato register, con la differenza che il numero di posizioni di cui Rm deve shiftare non è una costante ma è indicato dal registro Rs.

DP with Register-shifted Reg. *Src2*

```
EOR R8, R9, R10, ROR R12
```

Operation: R8 = R9 XOR (R10 ROR R12)

cond = 1110_2 (14)

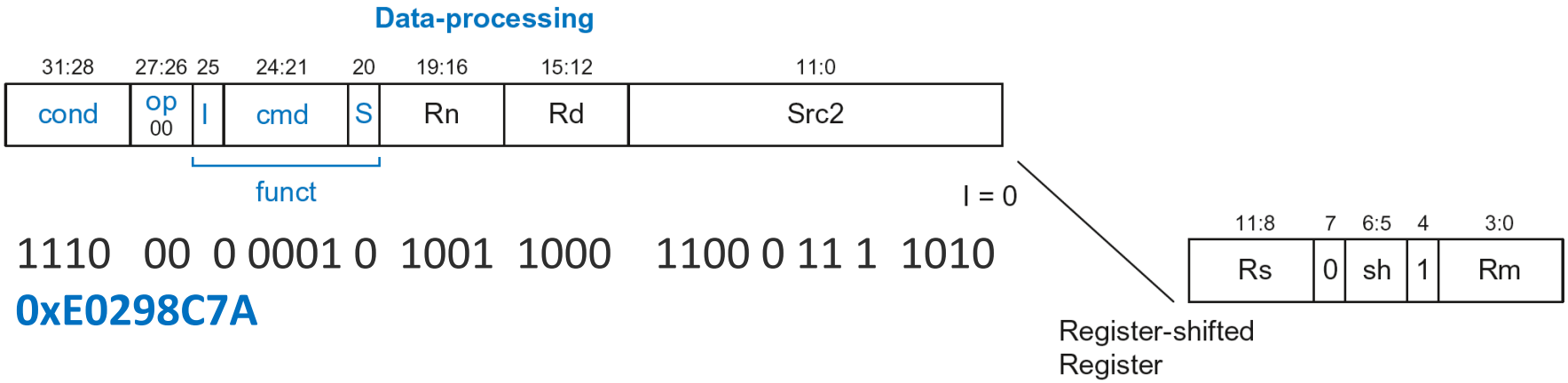
op = 00_2 (0)

cmd = 0001_2 (1) indica l'instruzione `EOR`

I=0

Rd = 8, ***Rn*** = 9, ***Rm*** = 10, ***Rs*** = 12

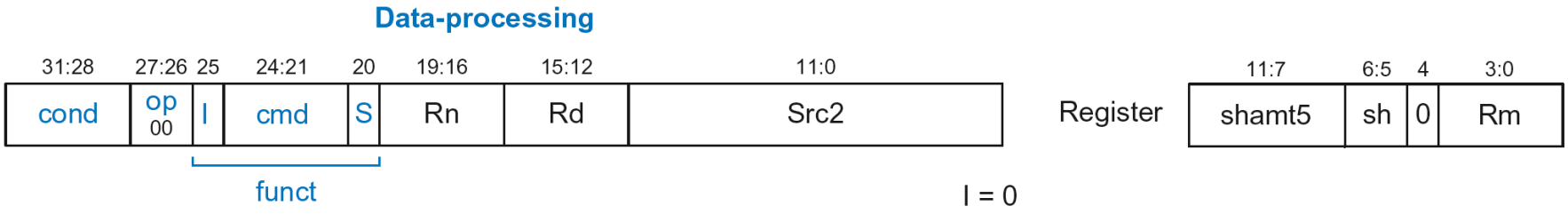
sh = 11_2 (ROR)



Shift Instructions: Immediate shamt

```
ROR R1, R2, #23
```

Operation: R1 = R2 ROR 23
cond = 1110_2 (14)
op = 00_2 (0)
cmd = 1101_2 (13) codice usato per tutti I tipi di shift (LSL, LSR, ASR, ROR)
Src2 è un immediate-shifted register quindi ***I***=0
Rd = 1, ***Rn*** = 0, ***Rm*** = 2
shamt5 = 23, ***sh*** = 11_2 (ROR)



1110 00 0 1101 0 0000 0001 10111 11 0 0010

0xE1A01BE2

Datapath ADD, SUB, AND, ORR

Estendiamo il **datapath** per gestire le istruzioni di data processing **ADD**, **SUB**, **AND** e **ORR**, utilizzando la modalità di indirizzamento immediato.

In tal caso, le istruzioni hanno come operandi un registro ed una costante contenuta nei bit dell'istruzione stessa. L'**ALU** esegue l'operazione e il risultato viene scritto in un terzo registro.

Esse differiscono solo nella specifica operazione eseguita dall'**ALU**. Quindi, possono essere implementate tutte con lo stesso hardware utilizzando diversi segnali **ALUControl**.

I valori per **ALUControl** sono:

- **ADD – 00**;
- **SUB – 01**;
- **AND – 10**;
- **ORR – 11**.

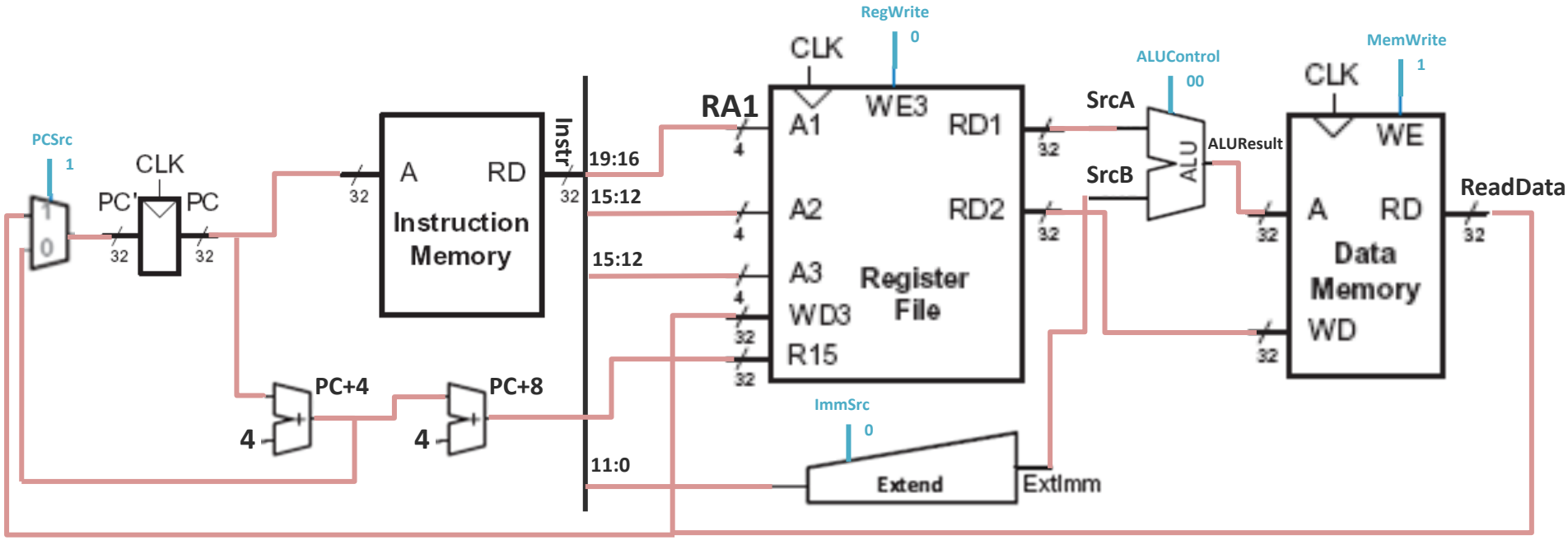
L'ALU imposta anche dei bit in **$ALUFlags_{3:0}$**:

- **Zero**;
- **Negativo**;
- **Carry**;
- **oVerflow**.

Datapath ADD, SUB, AND, ORR

Le istruzioni di data processing utilizzano costanti di **8** bit (non **12** bit), per cui il blocco **Extend** riceve in input un segnale di controllo **ImmSrc**:

- **ImmSrc** = **0** → **ExtImm** è esteso da $\mathbf{Instr}_{7:0}$;
- **ImmSrc** = **1** → **ExtImm** è esteso da $\mathbf{Instr}_{11:0}$ (per **LDR** o **STR**);



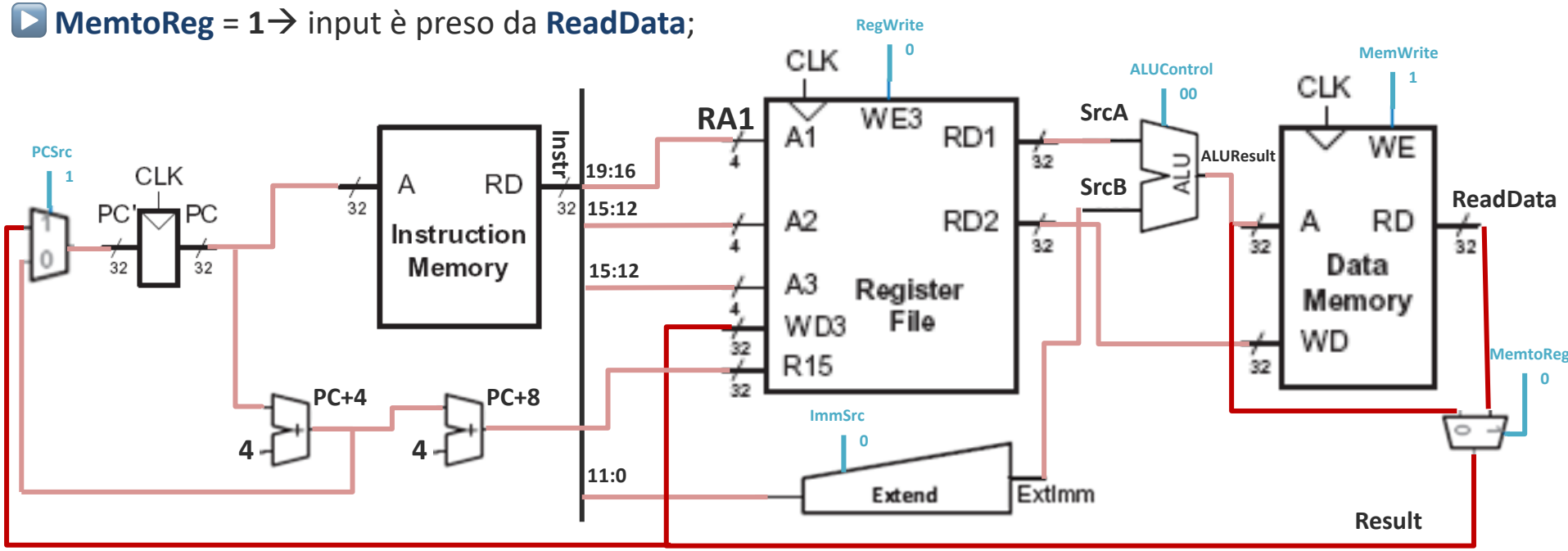
Datapath ADD, SUB, AND, ORR

Un altro aspetto da disambiguare riguarda la scrittura nel register file.
Esso può ricevere l'input sia dalla **memoria dati** (**LDR**), che dall'**ALU** (operazioni aritmetiche).

Aggiungiamo un altro **multiplexer** che permette di selezionare la sorgente di input tra **ReadData** e **ALUResult**. L'uscita del multiplexer è indicata con **Result**.

Il multiplexer richiede un segnale di controllo, ovvero **MemtoReg**.

- **MemtoReg** = **0** → input è preso da **ALUResult**;
- **MemtoReg** = **1** → input è preso da **ReadData**;

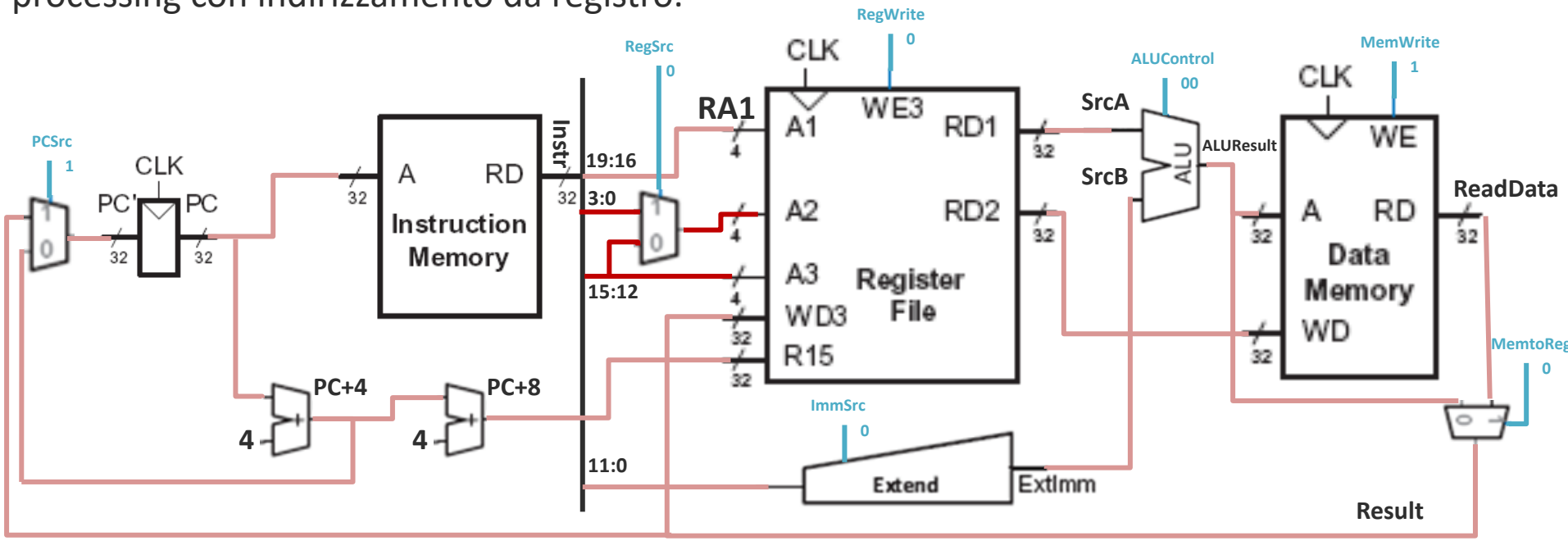


Datapath ADD, SUB, AND, ORR

Le istruzioni di data processing con indirizzamento da registro ricevono la loro seconda fonte da **Rm**, specificato da **$Instr_{3:0}$**, piuttosto che da una costante.

Aggiungiamo un ulteriore **multiplexer** sugli ingressi del file registro. In base al valore del segnale di controllo **RegSrc** , **RA2** può essere selezionato fra:

- **Rd** (**$Instr_{15:12}$**) per **STR**;
- **Rm** (**$Instr_{3:0}$**) per istruzioni di data processing con indirizzamento da registro.



Datapath ADD, SUB, AND, ORR

Aggiungiamo un ulteriore **multiplexer** sugli ingressi dell'**ALU** per selezionare questo secondo registro sorgente. In base al valore del segnale di controllo **ALUSrc**, la seconda sorgente della ALU viene selezionata tra:

- **ExtImm** per istruzioni, che utilizzano costanti;
- dal **register file** per istruzioni di data processing con indirizzamento da registro.