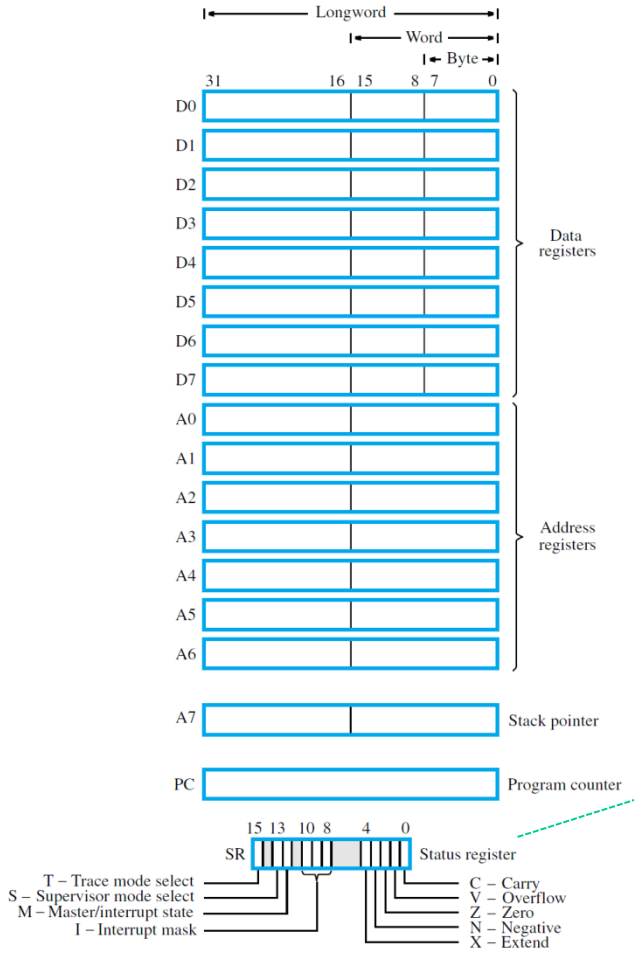
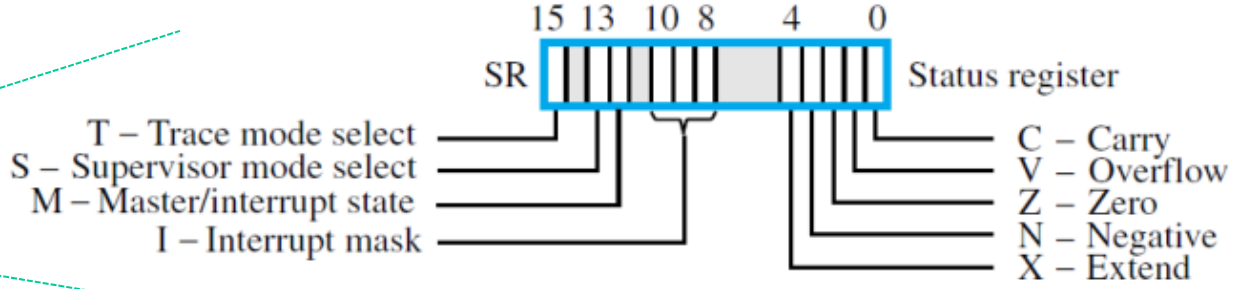
ColdFire ISA

Modello di Programmazione



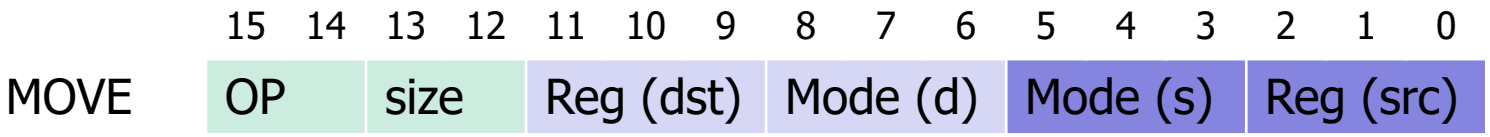
- Byte-addressable, 32-bit address space
- Big-endian addressing scheme
- Longword (32-bit), word (16-bit), and byte (8-bit) sizes for integer data (.L, .W, .B)
- Eight data registers, D0 to D7
- Eight address registers, A0 to A7, and register A7 is the stack pointer (SP)
- Status register (SR) with condition codes
- Program Counter (PC)



Istruzioni

- Codifica a lunghezza variabile:
  - Da una a tre parole consecutive
  - *OP-code* nella prima parola insieme a informazioni di indirizzamento
- La maggior parte delle istruzioni aritmetiche e di trasferimento hanno formato:

  OP src, dst
  - .L, .W, o .B come suffisso di OPcode per specificare dimensione del dato

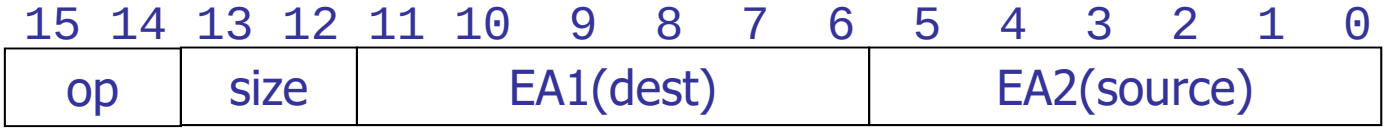


OP/Size = 0000 -> opcode expansion

Modi di Indirizzamento

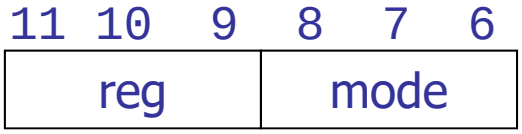
- Diretto a Registro
  - Data-register Direct
  - Address-register Direct
- Immediato (o Literal)
- Assoluto
  - Short
  - Long
- Indiretto a registro (indirizzo)
- Auto-Incremento (post)
- Auto-Decremento (pre)
- Indicizzato short
  - Con Base
  - Con Base e Spiazzamento
- Relativo
- Relativo con Indice

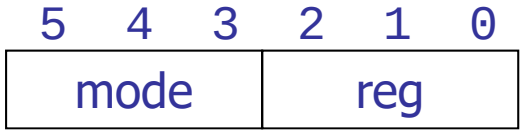
Codifica dell'istruzione MOVE



MOVE

- Il Campo EA, di 6 bit, è organizzato in due sottocampi da 3 bit ciascuno





alcuni modi possibili:

mode	reg	syntax	EA	name	#e.w.
0	0-7	Dn	Dn	Data-register direct	0
1	0-7	An	An	Address-register direct	0
2	0-7	(An)	MEM[An]	Address-register indirect	0
7	0	addr	MEM[addr]	Absolute short	1
7	4	#data	data	Immediate	1o 2

- Tutti i modi di indirizzamento utilizzabili per entrambi gli operandi (ad eccezione dell'immediato per la destinazione)
- Istruzione MOVE ortogonale (ISA ortogonale => tutte le istruzioni sono ortogonali)

Modi di Indirizzamento

- Diretto a Registro
  - Data-register Direct
  - Address-register Direct

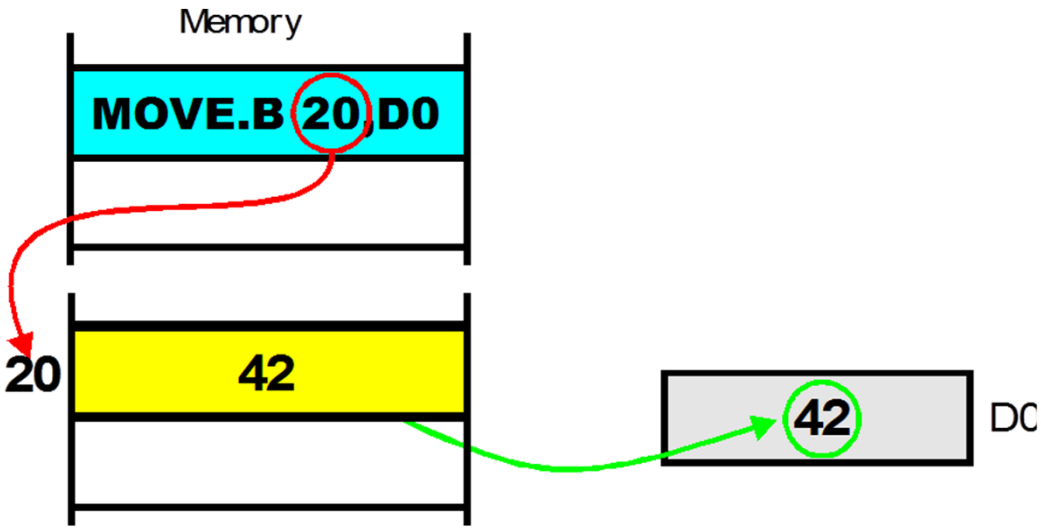
```
MOVE.B   D0,D1     ;[D0]->D1
```

- Immediato (o Literal)

```
MOVE.B   #4,D0     ;4->D0
```

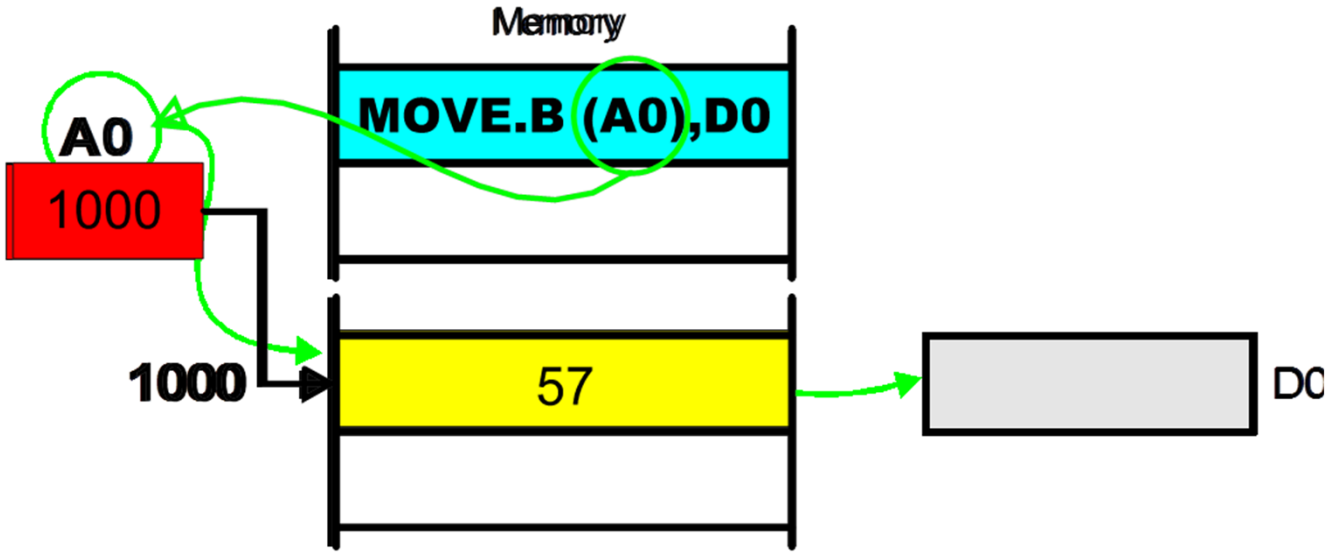
Modi di Indirizzamento: Assoluto

- Assoluto: **MOVE ADDR, D0**
  - Short
  - Long



Indiretto a registro indirizzo

- **MOVE (Ax), D0**



Post-Incremento e Pre-Decremento

- Il registro viene prima decrementato di un valore pare alla dimensione dell'operando successivamente il valore in D0 viene trasferito
  - **`MOVE.W D0,-(A7)`** Push D0 sullo stack (SP = A7)

- Prima il valore viene prelevato dall'indirizzo in A7 per essere trasferito in D0 dopodiché il valore in A7 è incrementato di un valore che dipende dalla dimensione del dato
  - **`MOVE.W (A7)+, D0`** Pop D0 dallo stack (SP = A7)

Indicizati

- Con Base e Spiazzamento **`MOVE X(An),D0`**
  - L'EA sorgente è calcolato sommando il contenuto del registro alla costante X (a 16 bit)

- Con Base, Indice e Spiazzamento **`MOVE X(An,In),D0`**
  - L'EA sorgente è calcolato sommando il contenuto del registro Base (A*n*) a quello del registro Indice (A*n* oppure D*n*) a quello della costante X (a 8 bit)

- Sostituendo il registro base a PC si hanno
  - **Relativo a PC** e
  - **Relativo a PC con indice**

Tabella A2.2 Modi di indirizzamento e relativa notazione simbolica nelle ISA NIOS II, ColdFire, ARM e IA-32

Modo di indirizzamento	NIOS II	ColdFire	ARM	IA-32
Immediato	Valore	#Valore	#Val	Valore
Assoluto o diretto	LOC(r0)	Valore	Val	LOC
Di registro	r*i*	R*i*	R*i*	R
Indiretto di registro	(r*i*)	(A*i*)	[R*i*]	[R]
Con base e spiazzamento	X(r*i*)	W(A*i*)	[R*i*,#Val]	[R+X]
Con indice e spiazz.				[R_x*S+X]
Con base e indice			[R*i*,±R*j*,s]	[R+R_x*S]
Con base, indice e spiazz.		B(A*i*,R*j*)		[R+R_x*S+X]
Con autoincremento		(A*i*)+		
Con autodecremento		–(A*i*)		
Relativo a PC		W(PC)	L	L
Relativo a PC con indice		B(PC,R*i*)		
Indiretto da memoria				*LOC oppure [R_x*S+X]
Con pre-base e spiazz.			[R*i*,#Val]!	
Con post-base e spiazz.			[R*i*],#Val	
Con pre-base e indice			[R*i*,±R*j*,s]!	
Con post-base e indice			[R*i*],±R*j*,s	

Legenda:

Valore numero con segno (a 16 bit in NIOS II, a 8 o 32 bit in IA-32) rappresentato esplicitamente o da etichetta;
Val numero rappresentato in valore assoluto e segno a 9 bit nel modo immediato, a 13 bit nei modi assoluto e con spiazzamento;
LOC indirizzo assoluto (a 16 bit in NIOS II, a 32 bit in IA-32);
R, R_x uno degli otto registri generali IA-32, ma non si può usare il registro ESP (puntatore alla pila) come registro indice R_x; R*i*, R*j*, ISA ColdFire: registro A*i* o D*i* (rispettivamente A*j* o D*j*);
X spiazzamento: numero con segno (a 16 bit in NIOS II, a 8 o 32 bit in IA-32, ma solo a 32 bit nel modo con indice e spiazzamento);
S fattore di scala (IA-32): 1, 2, 4 o 8;
s scorrimento logico (ARM): ds #vs dove ds ∈ {LSL, LSR}: direzione dello scorrimento e vs: valore dello scorrimento (numero a 5 bit);
W Valore a 16 bit;
B Valore a 8 bit;
L Etichetta.

COLDFIRE INSTRUCTIONS

https://www.nxp.com/docs/en/reference-manual/MCF5485RM.pdf

Pseudo Operatori e Direttive

- **ORG**: inizializza il PLC
- **END**: fine del programma riporta l'indirizzo di inizio
- **DC.W**: riserva e inizializza un'area di una o più word
- **DC.B**: riserva un'area di uno o più byte
- **DS.B**: riserva un'area di memoria di n byte senza inizializzarla
- **EQU** definisce una sostituzione (simile #define in C)

```
    ORG   $1000

    START
          …
          END START

    V DC.W 3,4,5

    S DC.B "Ciao",0
    V DC.B 5

    DEST DS.B 3

    EOL EQU 0
```

Istruzioni Aritmetiche

op SRC, DST → DST = DST op SRC

- Somma, sottrazione, confront e negazione:
  ADD.L, SUB.L, CMP.L, NEG.L
- **ADDI.L, ADDQ.L** per operandi immediate piccoli
- **ADDA.L, SUBA.L, CMPA.L** per registri indirizzo
  - Tutte le operazioni aritmetiche modificano i codici di condizione
- **ADDX.L, SUBX.L, NEGX.L** per numeri > 32 bits
- **MULS/MULU DIVS/DIVU** (Dst/Src->Dst)
  - Signed/unsigned
  - N e Z flags settati opportunamente; V e C resettati

Consultare il manuale, es. MOVE

MOVE Copy data from source to destination

Operation: [destination] ← [source]

Syntax: MOVE <ea>,<e>

Sample syntax:
```
MOVE (A5),-(A2)
MOVE -(A5),(A2)+
MOVE #$123,(A6)+
MOVE Temp1,Temp2
```

Attributes: Size = byte, word, longword

Description: Move the contents of the source to the destination location. The data is examined as it is moved and the condition codes set accordingly. Note that this is actually a *copy* command because the source is not affected by the move. The move instruction has the widest range of addressing modes of all the 68000's instructions.

Condition codes:

X	N	Z	V	C
-	*	*	0	0

Condition Code Register (CCR) values:

U The state of the bit is undefined (i.e., its value cannot be predicted)
- The bit remains unchanged by the execution of the instruction
* The bit is set or cleared according to the outcome of the instruction.

Source operand addressing modes

Dn	An	(An)	(An)+	-(An)	(d,An)	(d,An,Xi)	ABS.W	ABS.L	(d,PC)	(d,PC,Xn)	imm
✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔

Destination operand addressing modes

Dn	An	(An)	(An)+	-(An)	(d,An)	(d,An,Xi)	ABS.W	ABS.L	(d,PC)	(d,PC,Xn)	imm
✔		✔	✔	✔	✔	✔	✔	✔			

Istruzione Logiche e Shift

- **AND.L, OR.L, EOR.L, NOT.L**
  - Almeno un operando di tipo Data Register
- **ANDI.L, ORI.L, EORI.L**
  - Operando Sorgente di tipo immediate

- N e Z modificai opportunamente; V e C resettati

- **LSL.L, LSR.L, ASL.L, ASR.L**
  - Lo shift è specificato con un immediate o un registro indirizzo
  - L'operando per lo shift in un data register

Modi Elementari

Consideriamo l'operazione

Z = Y + 24

Essa corrisponde al seguente codice

```
        ORG       $400    code section
        MOVE.B    Y,D0
        ADD       #24,D0
        MOVE.B    D0,Z

        ORG       $600    data section
Y       DC.B      27      store a constant
Z       DS.B      1       reserve a byte for Z
```

Assembled code

```
1     00000400                          ORG  $400
2     00000400 103900000600             MOVE.B   Y,D0
3     00000406 06000018                 ADD.B  #24,D0
4     0000040A 13C000000601             MOVE.B   D0,Z
5     00000410 4E722700                 STOP     #$2700
6              *
7     00000600                          ORG      $600
8     00000600 1B               Y:      DC.B     27
9     00000601                  Z:      DS.B     1
10    00000400                          END      $400
```

Example

```
* File: autoinc.a68 – Somma elementi di un vettore

            ORG     $8000
            MOVE.B  #5,D0
            LEA     Table,A0        A0 points the list
            CLR.B   D1              clear the accumulator
Loop        ADD.B   (A0)+,D1        add up next element
            SUB.B   #1,D0
            BNE     Loop

            ORG     $8100
Table       DC.B    1,2,3,4,5       Sample vector
```

Somma a 64 bit

```
MOVE.L    #$A72C10F8, D2    D2 contains A72C10F8.
MOVE.L    #$10, D3          D3 contains 10.
MOVE.L    #$5C00FE04, D4    D4 contains 5C00FE04.
MOVE.L    #$4A, D5          D5 contains 4A.
ADD.L     D2, D4            Add low-order 32 bits; carry-out sets X and C flags.
ADDX.L    D3, D5            Add high-order bits with X flag as carry-in bit.
```

Program to add numbers larger than 32 bits using the ADDX instruction.

Moltiplicazione

```
MOVE.W    #$FFFF, D2    The low-order word of D2 is treated as –1.
MOVE.W    #$0001, D3    The low-order word of D3 contains 1.
MULS.W    D2, D3        The signed longword result in D3 is –1
                          or $FFFFFFFF, hence the N flag is set.
```

(a) Signed computation of $-1 \times 1 = -1$

```
MOVE.W    #$FFFF, D2    The low-order word of D2 is treated as 65535.
MOVE.W    #$0001, D3    The low-order word of D3 contains 1.
MULU.W    D2, D3        The unsigned longword result in D3 is 65535
                          or $0000FFFF, hence the N flag is cleared.
```

(b) Unsigned computation of $65535 \times 1 = 65535$

Istruzioni Branch e Jump

- **JMP** è un salto assoluto non condizionato
  - Destinazione assoluta o registro indirizzo
- **B*cc***: Branch definisce un salto relative condizionato dai codici di condizione e.g., BEQ controlla se Z=1
  - **BRA** è un salto relative non condizionato

Condition suffix *cc*	Name	Test condition
HI	High	$C \vee Z = 0$
LS	Low or same	$C \vee Z = 1$
CC	Carry clear	$C = 0$
CS	Carry set	$C = 1$
NE	Not equal	$Z = 0$
EQ	Equal	$Z = 1$
VC	Overflow clear	$V = 0$
VS	Overflow set	$V = 1$
PL	Plus	$N = 0$
MI	Minus	$N = 1$
GE	Greater or equal	$N \oplus V = 0$
LT	Less than	$N \oplus V = 1$
GT	Greater than	$Z \vee (N \oplus V) = 0$
LE	Less or equal	$Z \vee (N \oplus V) = 1$

Somma gli elementi di un vettore

```
        MOVEA.L   #NUM1, A2    Put the address NUM1 in A2.
        MOVE.L    N, D1        Put the number of entries n in D1.
        CLR.L     D0
LOOP:   ADD.L     (A2)+, D0    Accumulate sum in D0.
        SUBQ.L    #1, D1
        BGT       LOOP
        MOVE.L    D0, SUM      Store the result when finished.
```

Ricerca carattere in una stringa D1=posizione

```
 ORG     $1000
START:
     MOVE.B   C,D0
     MOVE.B   #0,D1
     MOVE.B   #0,D3
     LEA      S,A0

LOOP
     CMP.B    #1,D3
     BEQ      FINELOOP

     MOVE.B (A0)+,D2
     BEQ      FINELOOP
```

```
     CMP.B    D0,D2
     BNE      NONTROV
     MOVE     #1,D3

NONTROV ADDQ.B #1,D1
     BRA      LOOP

FINELOOP
     CMP      #0,D3
     BNE      fine
     MOVE     #0,D1

fine
```

Tabella A2.7a Istruzioni NIOS II, ColdFire, ARM e IA-32

Tipo di istruzioni	NIOS II	ColdFire	ARM	IA-32
(a) Istruzioni di trasferimento, di controllo				
Trasferimento				
Load	ld*b* r*i*, X(r*j*)		LDR*b*	
Store	st*b* r*i*, X(r*j*)		STR*b*	
Move	mov*a*	MOVE*a.b*	MOV, MVN	MOV, LEA, PUSH, POP
Multiplo		MOVEM.*b*	LDM*w*, STM*w*	POPAD, PUSHAD
Controllo				
Salto incondiz.	br *l*, jmp r*i*	JMP	B *l*	JMP
Salto condiz.	bc r*i*, r*j*, *l*	Bc *l*	Bc *l*	Jc *l*
				LOOP *l*
Chiamata e rientro: si veda Tabella A2.5				

Legenda:

b suffisso del codice operativo, dimensione del dato in memoria, estensione a 32 bit: NIOS II: $b \in$ {w,b,h,bu,hu}, ColdFire: $b \in$ {B,W,L}, ARM: $b \in$ {B,H,SB,SH} opzionale;

a suffisso opzionale del codice operativo, modo di indirizzamento: NIOS II: $a \in$ {i,ui,ia}, indirizzamento immediato, ColdFire: $a \in$ {A,Q}, se la dest. è un registro indirizzo o dati, rispettivamente;

w suffisso del codice operativo, progressione del registro di base, $w \in$ {IA,DA,IB,DB}; *l*, etichetta; *c*, suffisso del codice operativo, condizione aritmetico-logica di salto: NIOS II: $c \in$ {eq,ne,ge,geu,gt,gtu,le,leu,lt,ltu}, ColdFire, ARM, IA-32: si veda Tabella A2.8

Tabella A2.7b Istruzioni NIOS II, ColdFire, ARM e IA-32

Tipo di istruzioni	NIOS II	ColdFire	ARM	IA-32
(b) Istruzioni aritmetiche, di confronto, logiche, di scorrimento				
Aritmetiche				
Addizione	add*m*	ADD*m*.L	ADD*f*, ADC	ADD, ADC
Sottrazione	sub*m*	SUB*m*.L	SUB*f*, SBC	SUB, SBB
		NEG*m*.L		NEG
Moltiplicazione	mul*m*	MULs.*b*	MUL, MLA	IMUL
Divisione	div, divu	DIVs.*b*		IDIV
Resto		REMs.L		
Altre		EXT.*b*		INC, DEC
		CLR.*b*		
Confronto	cmp*cm*	CMP*m*.L	CMP, CMN	CMP
Logiche				
Congiunzione	and*m*, andhi	AND*m*.L	AND, TST	AND
Disgiunzione	or*m*, orhi	OR*m*.L	ORR	OR
Disg. esclusiva	xor*m*, xorhi	EOR*m*.L	EOR, TEQ	XOR
Negazione		NOT*m*.L		NOT
Altre	nor		BIC	
Scorrimento				
Logico	srl*m*	LSR.L	*t*, LSR	SHR
	sll*m*	LSL.L	*t*, LSL	SHL
Aritmetico	sra*m*	ASR.L	*t*, ASR	SAR
		ASL.L		SAL
Rotazione	ror		*t*, ROR	ROR, RCR
	rol*m*			ROL, RCL

Legenda:
m suffisso opzionale del codice operativo, NIOS II: $m \in$ {i}, secondo operando sorgente immediato, ColdFire: $m \in$ {I,A,X}, sorgente immediato (sola opzione nelle istruzioni logiche), o dest. registro indirizzo, o riporto in ingresso da bit di esito X (sola opzione in NEG, esclusa in CMP);
f suffisso opzionale del codice operativo, imposta i bit di esito C, V se *f* = S; *s*, suffisso del codice operativo, operandi con o senza segno, $s \in$ {S,U};
b suffisso del codice operativo, dimensione dell'operando sorgente o di destinazione; ColdFire: $b \in$ {W,L} in MUL, DIV; $b \in$ {B,W,L} in EXT, CLR;
c suffisso del codice operativo, condizione di confronto (come nelle istruzioni di salto); *t*, istruzione MOV o ADD, lo scorrimento si applica al secondo operando sorgente.

Gestione delle Subroutine