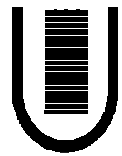


Architettura dei Calcolatori

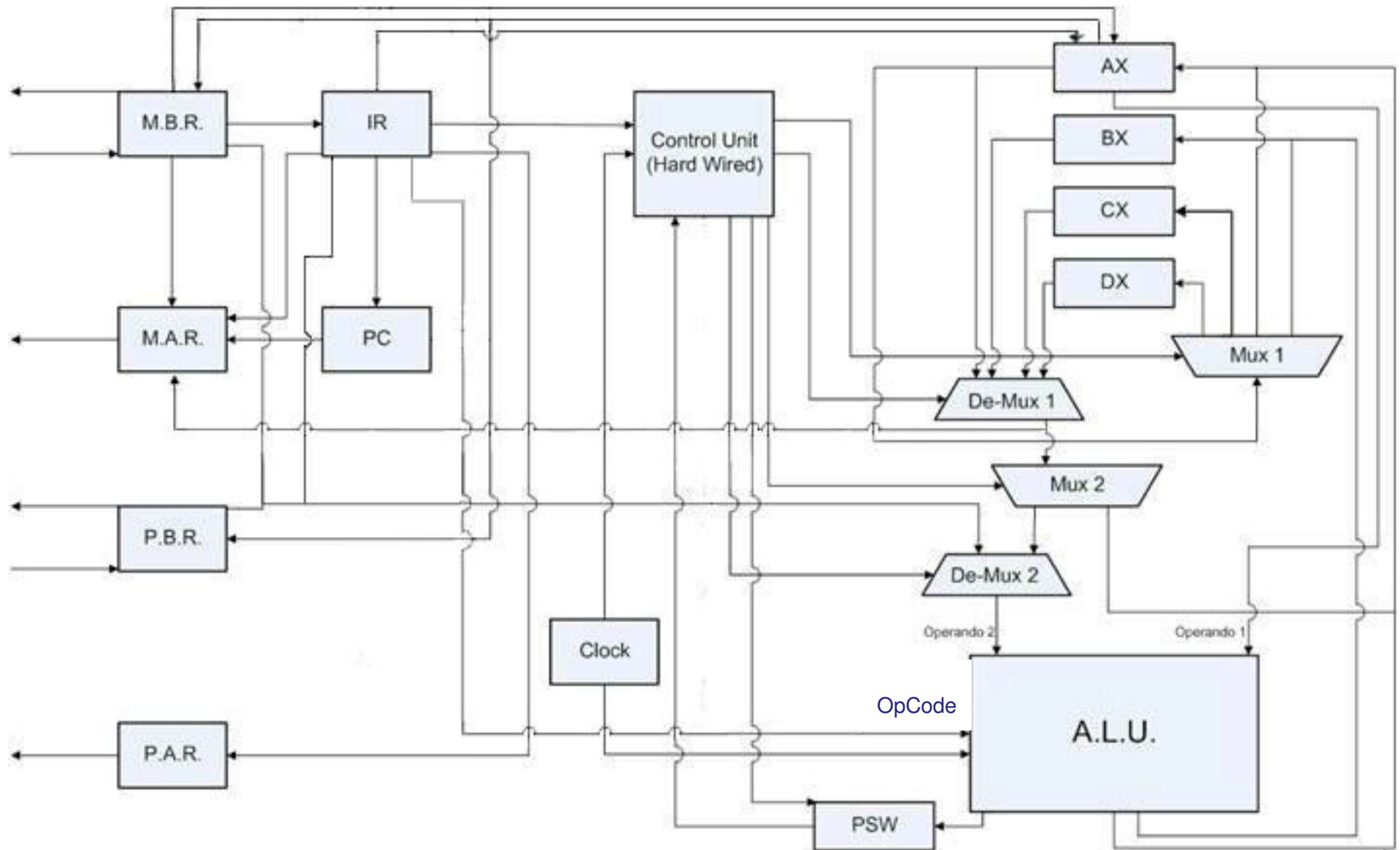
Prof. Enrico Nardelli



Università degli Studi di Roma “Tor Vergata”

Virtual CPU (Eniac): parte 2

Dove eravamo rimasti



I Flag

- ❑ Bit del registro PSW.
- ❑ Utilizzati per prendere decisioni in base all'esito di istruzioni precedenti, così da poter cambiare il flusso del programma
- ❑ Sono la rappresentazione di una parte dello stato finale a cui è arrivata l'ALU nell'effettuazione dell'ultima operazione

I Flag

- Semplici - il meccanismo che ne determina il valore dipende **solo** dallo stato finale dell'ALU, **non** da quale operazione è stata effettuata
- Complessi - il loro valore viene determinato anche in base all'operazione effettuata
 - Definizioni diverse per le diverse operazioni
 - Differenti interpretazioni a livello aritmetico

Flag semplici

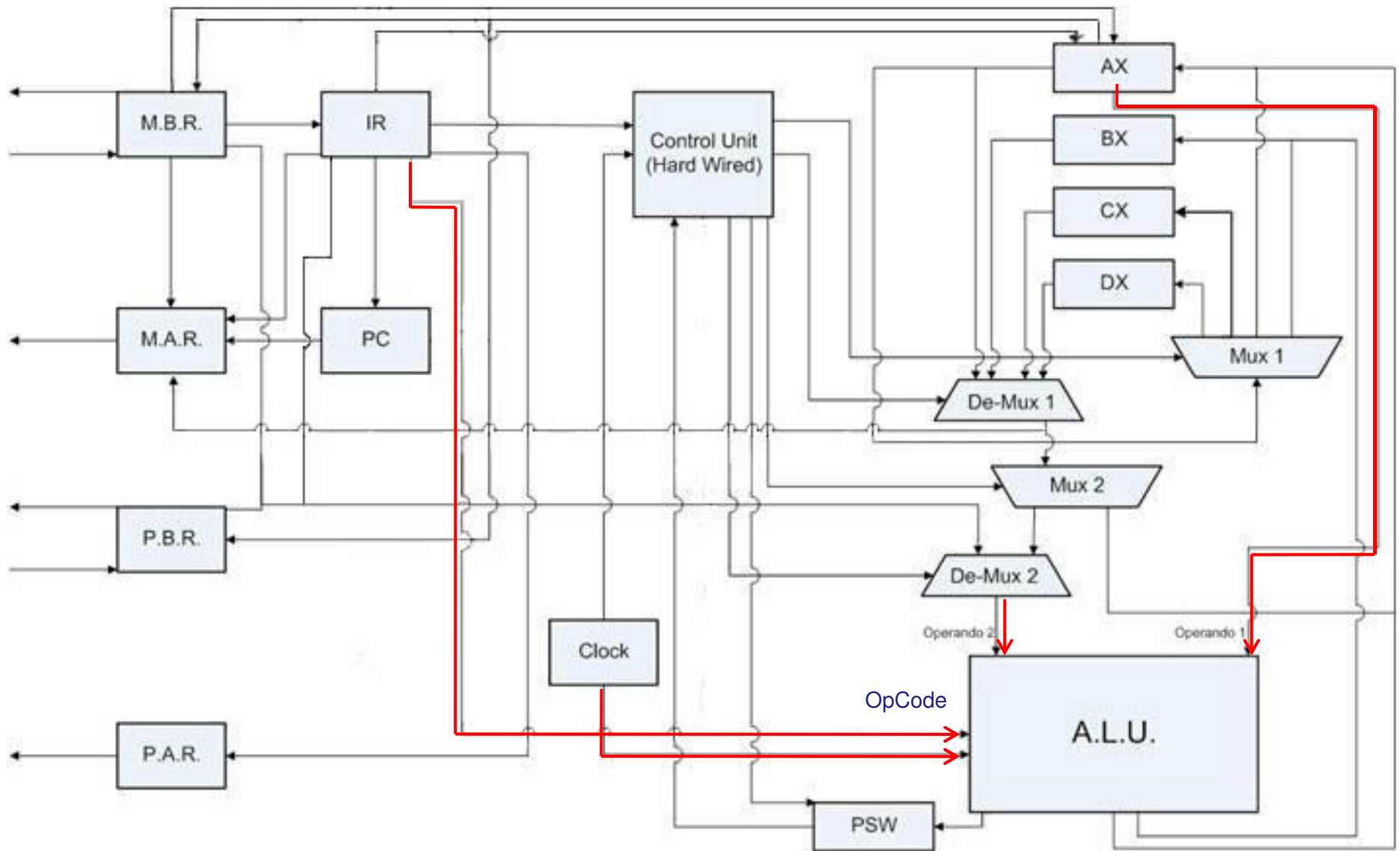
- Semantica indipendente dall'operazione effettuata (per i flag complessi non è così)
- $SI = 1$ se l'ultima operazione ha generato un risultato in cui il bit più significativo = 1,
= 0 altrimenti;
- $ZE = 1$ se il risultato dell'ultima operazione è zero,
= 0 altrimenti;
- $EV = 1$ se la somma dei bit a 1 del risultato dell'ultima operazione è pari (*even*) – Attenzione! non implica che il risultato sia pari!,
= 0 altrimenti.

La ALU e le sue funzionalità

- Operazioni possibili:
somma, sottrazione, divisione,
moltiplicazione, negazione, modulo,
and, or, or esclusivo (xor) e not.

- Ha quattro ingressi
 - Due di dati
 - Uno di controllo (direttamente da IR!)
 - Uno per il clock

Ingressi



La ALU e le sue funzionalità

- Scelte progettuali
 - L'ingresso di controllo è direttamente il codice operativo dell'istruzione
 - Il primo operando è sempre AX.
 - Il secondo operando è selezionato dalla CU sulla base dell'opcode in IR
 - L'uscita va sempre in AX ma, nel caso specifico della moltiplicazione, viene usato anche BX
 - Insieme al risultato escono anche i valori che configurano i flag nel registro PSW
 - Registri buffer interni sia in ingresso che in uscita
- I due operandi in ingresso hanno tanti bit quanti una parola di memoria

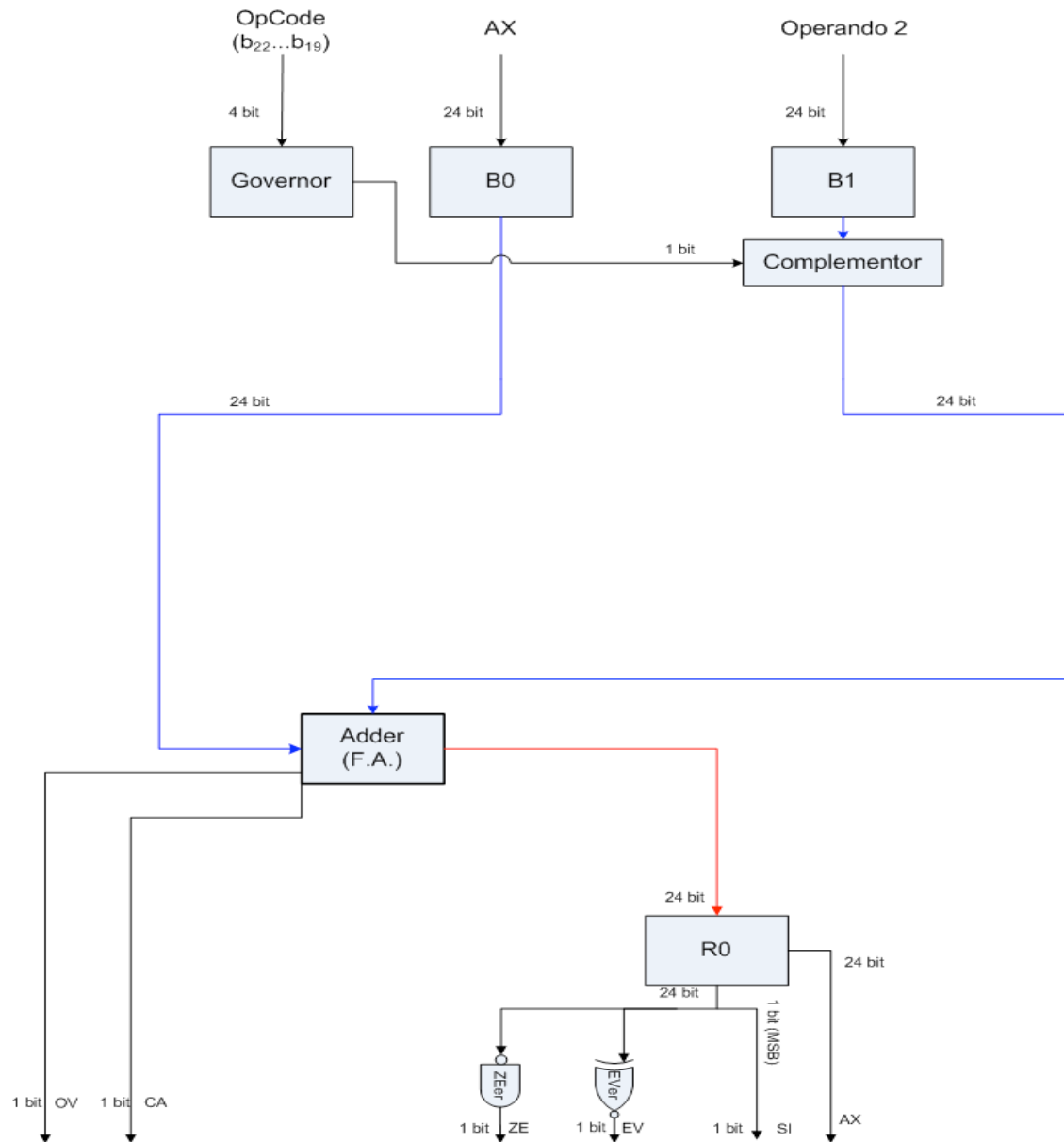
Comprendere la struttura della ALU di vCPU

- ▣ Descrizione incrementale (comprensione graduale)

Gli elementi :

- ▣ **Adder,**
 - implementato per mezzo di un Full Adder, in grado di effettuare delle addizioni
 - Gli operandi dell'operazione vengono inizialmente memorizzati nei registri interni alla ALU, B0 e B1 (memorizzano rispettivamente il valore di AX e l'operando specificato nell'istruzione)
 - La presenza di un circuito di complementazione a 2, indicato nella figura come **Complementor**, permette inoltre di eseguire le sottrazioni.

Una semplice ALU in grado di effettuare somme e sottrazioni



Rappresentazione degli interi

- ❑ vCPU è dotata di istruzioni in grado di elaborare solo interi relativi.
- ❑ Codifica adottata è quella del complemento a due.

Vantaggi del complemento a due rispetto ad altre notazioni

- Ha una sola rappresentazione dello zero
- Permette di effettuare le operazioni aritmetiche utilizzando la stessa circuiteria in grado di compiere operazioni aritmetiche su interi senza segno

Rappresentazione degli interi in complemento a 2

- E' il metodo più diffuso per la rappresentazione dei numeri relativi in informatica
- E' inoltre una operazione di negazione
- Ragione della diffusione: i circuiti di addizione e sottrazione non devono esaminare il segno di un numero rappresentato con questo sistema per determinare quale operazione sia necessaria.
- Il bit iniziale rappresenta il segno
(se 1 il numero e' negativo: per ottenere il modulo complemento e sommo 1)
- Il complemento a due di un numero negativo restituisce il numero positivo pari al valore assoluto

Rappresentazione degli interi in complemento a 2

La rappresentazione in complemento a 2 richiede di definire a priori la grandezza dell' intervallo di rappresentazione, cioè il numero di bit dedicati alla rappresentazione del numero stesso.

Un numero rappresentato in complemento a due usando N bit si può rappresentare anche utilizzando $N+K$ bit aggiungendo prima del MSB per K volte il valore del MSB stesso.

$$(-3)_{10} = (101)_2 \quad \text{con 3 bit}$$

$$(-3)_{10} = (11101)_2 \quad \text{con 5 bit}$$

Rappresentazione degli interi in complemento a 2

- Usando $4(n)$ bit si rappresentano i numeri tra $-8(-2^{n-1})$ e $+7(+2^{n-1}-1)$

-1 = 1111	+0 = 0000
-2 = 1110	+1 = 0001
-3 = 1101	+2 = 0010
-4 = 1100	+3 = 0011
-5 = 1011	+4 = 0100
-6 = 1010	+5 = 0101
-7 = 1001	+6 = 0110
-8 = 1000	+7 = 0111

Rappresentazione degli interi in complemento a 2

Esempio: Rappresentiamo il numero -5 con 8 bit in complemento a due.

- Parto dalla rappresentazione in binario del numero 5.

0000 0101 (5)

La prima cifra è 0, il numero è positivo

- Inverto i bit (ottengo il complemento a uno)

1111 1010

- Aggiungendo uno ottengo il complemento a due

1111 1011 (-5)

Il primo bit (è 1) evidenzia che il numero è negativo

- Il risultato è la rappresentazione in complemento a due di un numero binario.

Rappresentazione degli interi in complemento a 2

- Un numero intero con segno è rappresentato in complemento a 2 dalla stringa

$$b_n \ b_{n-1} \ \dots \ b_1 \ b_0$$

dove $b_i \in \{0, 1\}$ tali che

$$-b_n 2^n + b_{n-1} 2^{n-1} + \dots + b_1 2^1 + b_0 2^0 = X$$

Si rappresentano quindi tutti i numeri relativi che vanno da -2^n a 2^n-1 inclusi.

Notare che il bit più significativo ha un peso negativo.

Comprendere la struttura della ALU di vCPU

□ Il modulo **Governor** :

- decodifica l'opcode in input
- attiva la circuiteria per effettuare l'operazione richiesta.

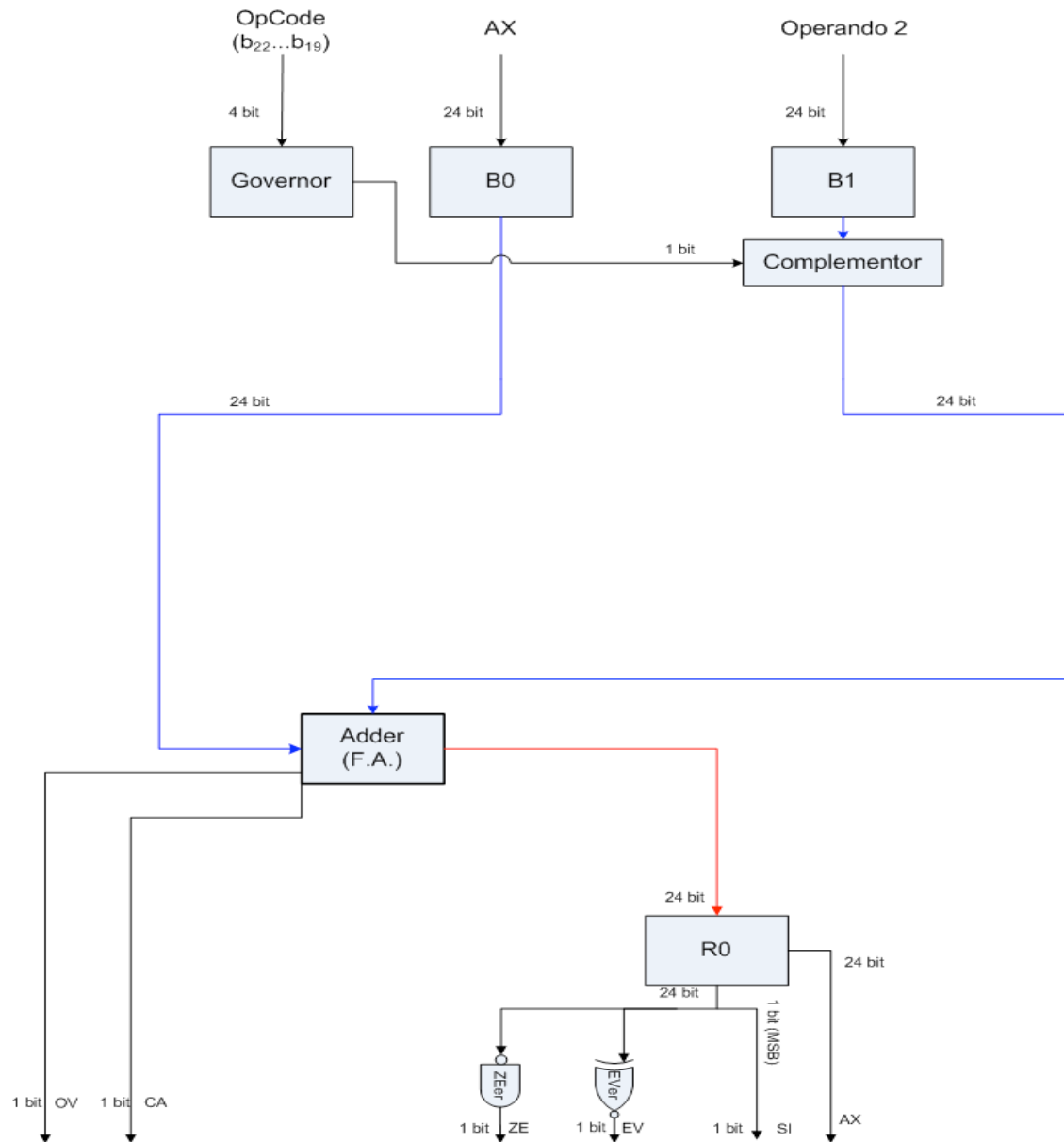
Per esempio, quando l'opcode identifica una sottrazione, il Governor attiva il Complementor che trasforma il secondo operando nel suo complemento a 2 e rende così possibile effettuare la sottrazione dal primo operando mediante l'Adder.

$$(3) - (5)$$

$$(3) + (\text{compl. a 2 di } 5)$$

$$\text{ovvero } (0011)_2 + (1011)_2$$

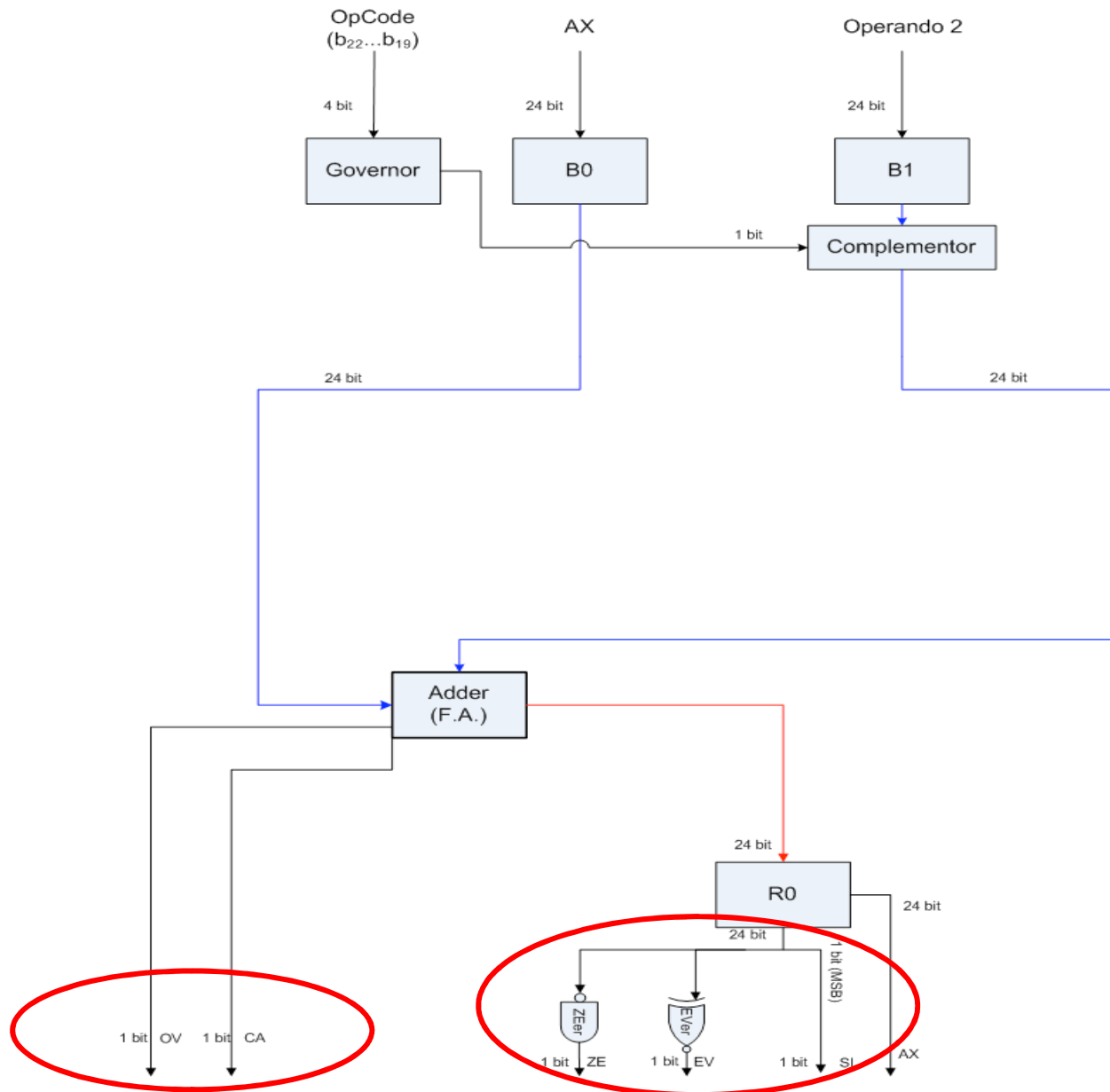
Una semplice ALU in grado di effettuare somme e sottrazioni



Comprendere la struttura della ALU di vCPU

- Risultato viene memorizzato nel registro R0 da dove vengono prelevati i valori necessari per il calcolo dei flag semplici
 - SI coincide con il bit più significativo in R0 (segno)
 - ZE con l'AND di tutti i bit in R0 preventivamente negati
 - EV con lo XOR negato di tutti i bit in R0.
- Flag complessi forniti direttamente dall'Adder (semantica e costruzione)
 - CA = 1 se il bit di riporto dell'Adder vale 1 nell'ultima operazione, CA=0 altrimenti.
 - OV = 1 se l'ultima operazione ha generato un risultato il cui bit più significativo differisce da entrambi i bit più significativi degli operandi (tra loro uguali, cioè P+P, N+N, P-N, N-P), OV=0 altrimenti;

Una semplice ALU in grado di effettuare somme e sottrazioni



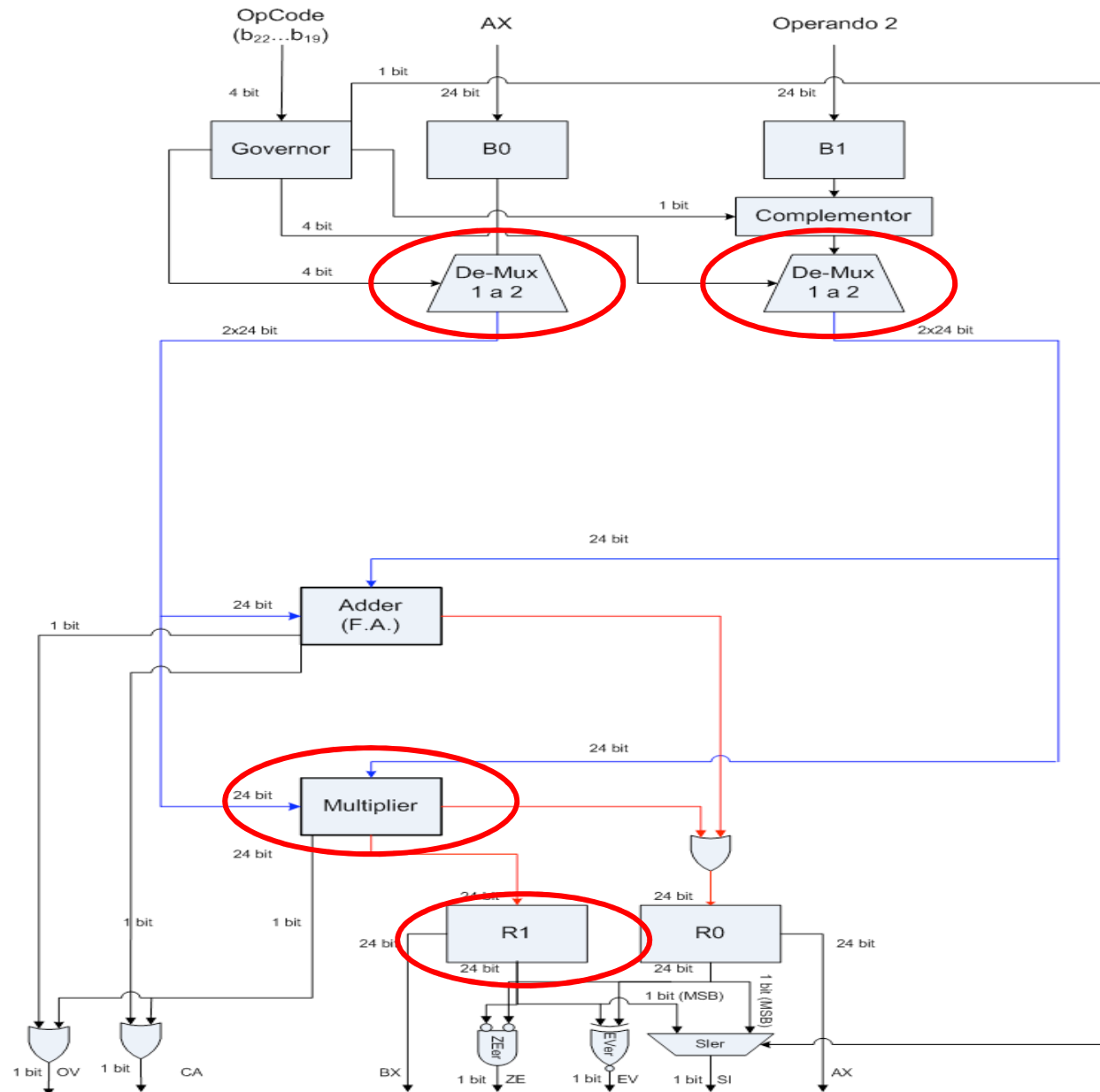
ALU somme sottrazioni e moltiplicazioni

Successivo livello di complessità: aggiunta di un modulo in grado di effettuare le operazioni di moltiplicazione: il **Multiplier**

- introduzione di due de-multiplexer, guidati dal Governor, in grado di indirizzare gli operandi verso il modulo d'interesse (Adder o Multiplier).
- La natura dell'operazione di moltiplicazione comporta l'introduzione di un secondo registro per la memorizzazione del risultato: il registro R1.

il Multiplier memorizza la parte meno significativa del risultato in R0 e la più significativa in R1

Una semplice ALU in grado di effettuare somme e sottrazioni e moltiplicazioni

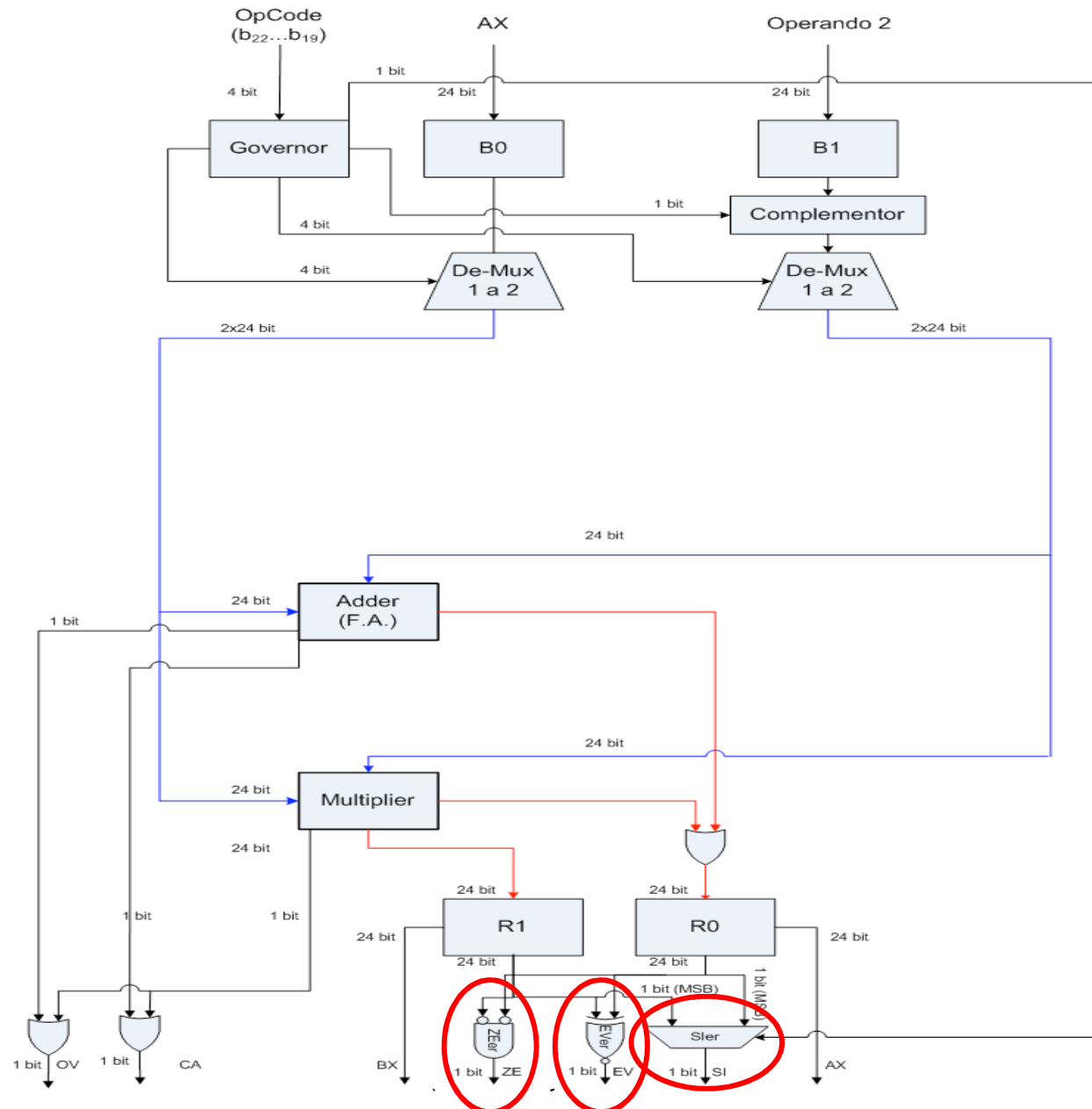


ALU somme sottrazioni e moltiplicazioni

- Conseguentemente all'introduzione del nuovo registro, la logica per il calcolo dei flag semplici cambia.
 - Per ZE ed EV vengono considerati anche i bit in R1
 - Per SI viene utilizzato un de-multiplexer, pilotato dal Governor, in grado di selezionare il bit più significativo che definirà valore del flag.

In particolare se l'operazione da effettuare è una moltiplicazione, verrà utilizzato il bit più significativo in R1, altrimenti quello in R0.

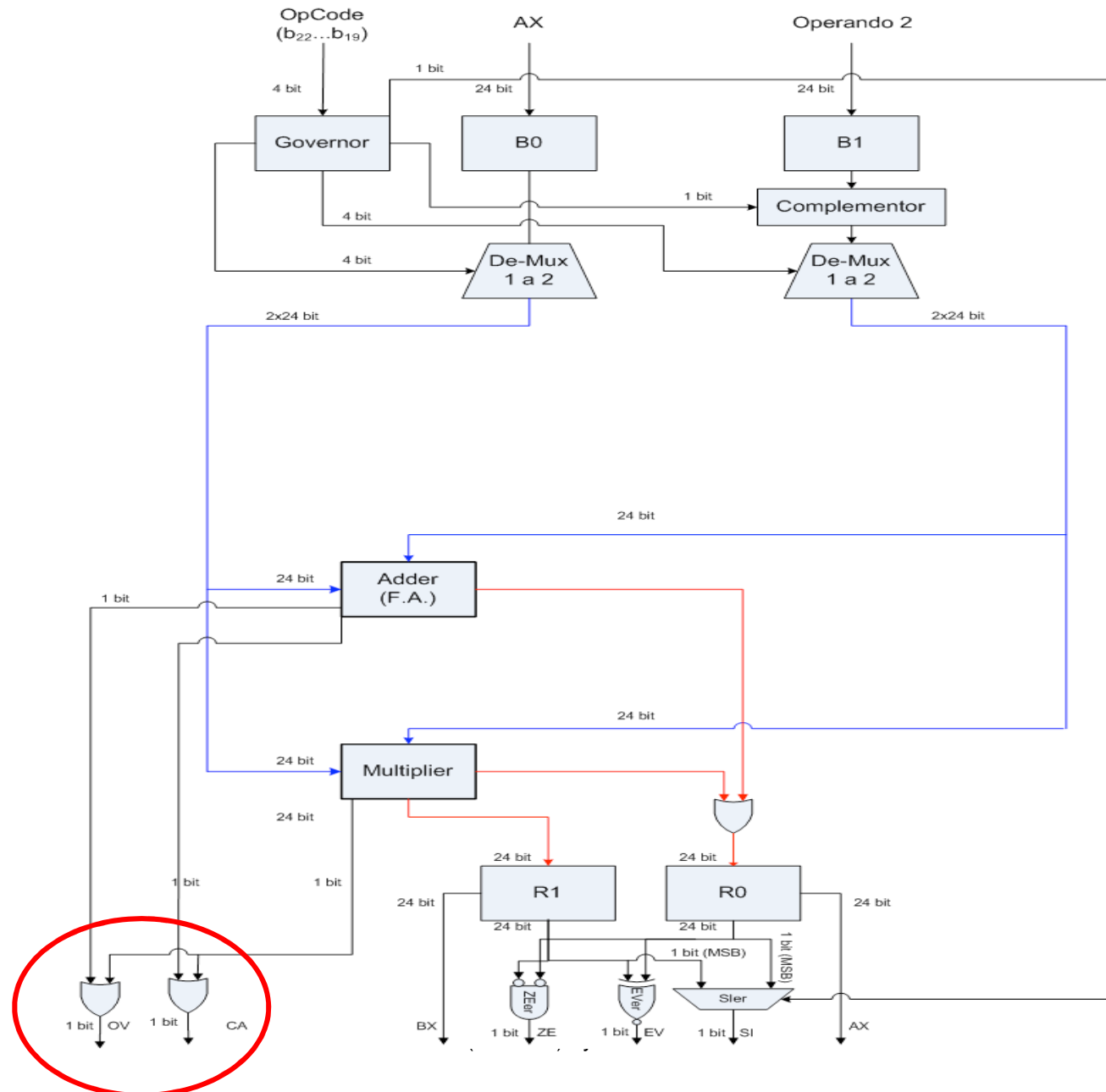
Una semplice ALU in grado di effettuare somme e sottrazioni e moltiplicazioni



ALU somme sottrazioni e moltiplicazioni

- Flag complessi forniti direttamente dal Multiplier (semantica e costruzione)
 - $OV=0$ quando i bit in R1 e il bit più significativo in R0 sono tutti uguali tra di loro,
 $OV=1$ altrimenti.
[indica se l'estensione a R1 della rappresentazione contenuta in R0 è aritmeticamente corretta, cioè se $OV=0$ il risultato sta tutto dentro R0]
 - CA: viene costruito con lo stesso criterio usato per OV.
[stessa semantica]
- Per poter avere solo un bit in uscita sia per il flag OV che per CA, sono state introdotte due porte OR in grado di riunire i segnali uscenti dai moduli Adder e Multiplier in uno solo.

Una semplice ALU in grado di effettuare somme e sottrazioni e moltiplicazioni



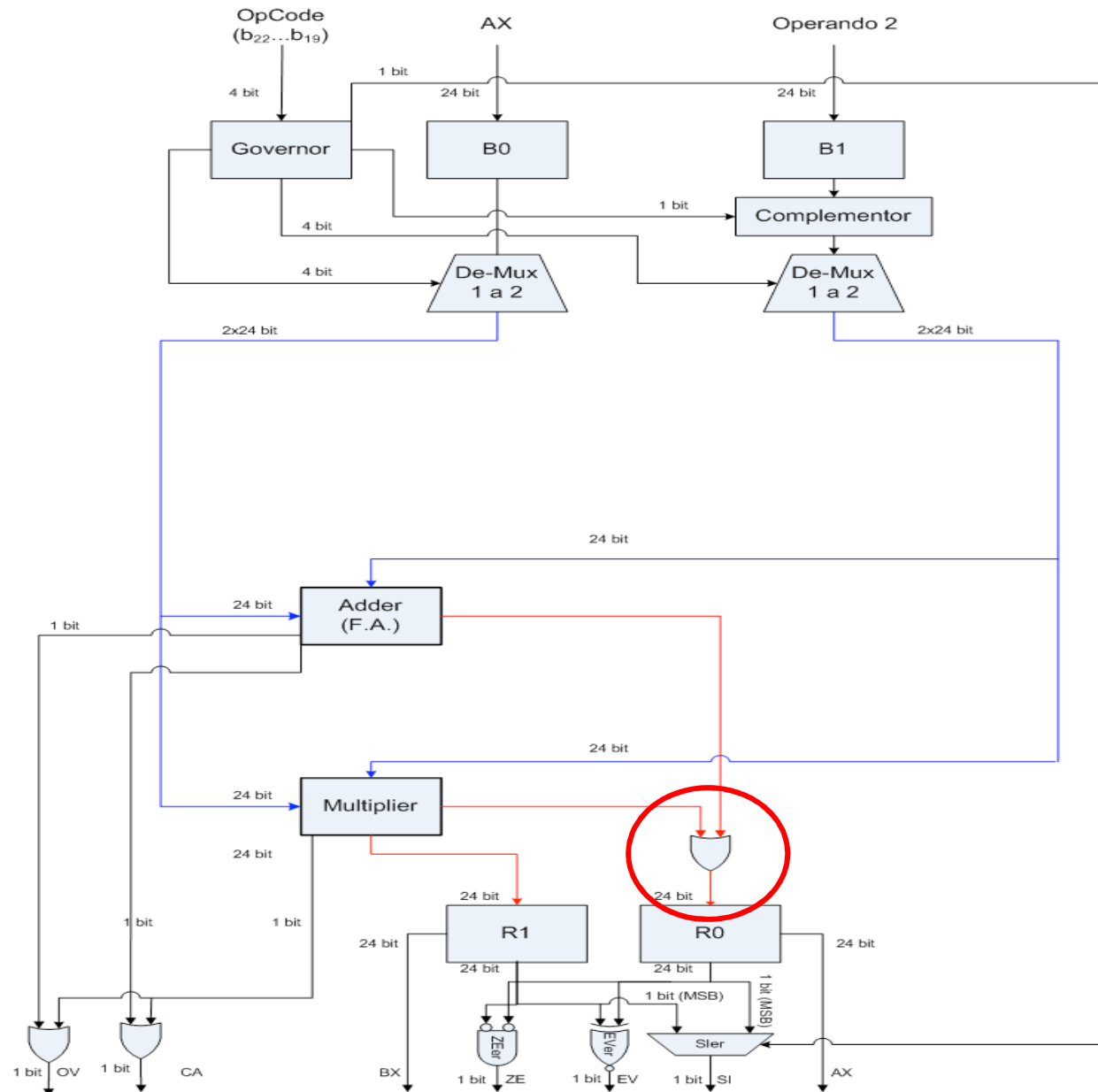
ALU somme sottrazioni e moltiplicazioni

- Tenendo presente che :
 - le porte OR hanno output 1 nel caso in cui almeno uno degli ingressi valga 1
 - grazie ai de-multiplexer collegati con gli operandi i moduli Adder e Multiplier non possono essere attivi contemporaneamente
- i flag OV e CA in uscita dalla ALU hanno sempre il valore generato dal modulo attivato.

ALU somme sottrazioni e moltiplicazioni

- ❑ Analogamente ai flag complessi, anche il registro R0 riceve in ingresso l'OR di tutte le uscite dei moduli.
- ❑ Come visto prima, grazie ai de-multiplexer, può essere attivo solo un modulo alla volta e il segnale in uscita dal modulo attivato, in OR con tutte le uscite a 0 degli altri moduli, coincide proprio con il risultato dell'operazione richiesta.
- ❑ Conseguenza di ciò è che non si ha la necessità di avere dei segnali di abilitazione per ogni modulo.

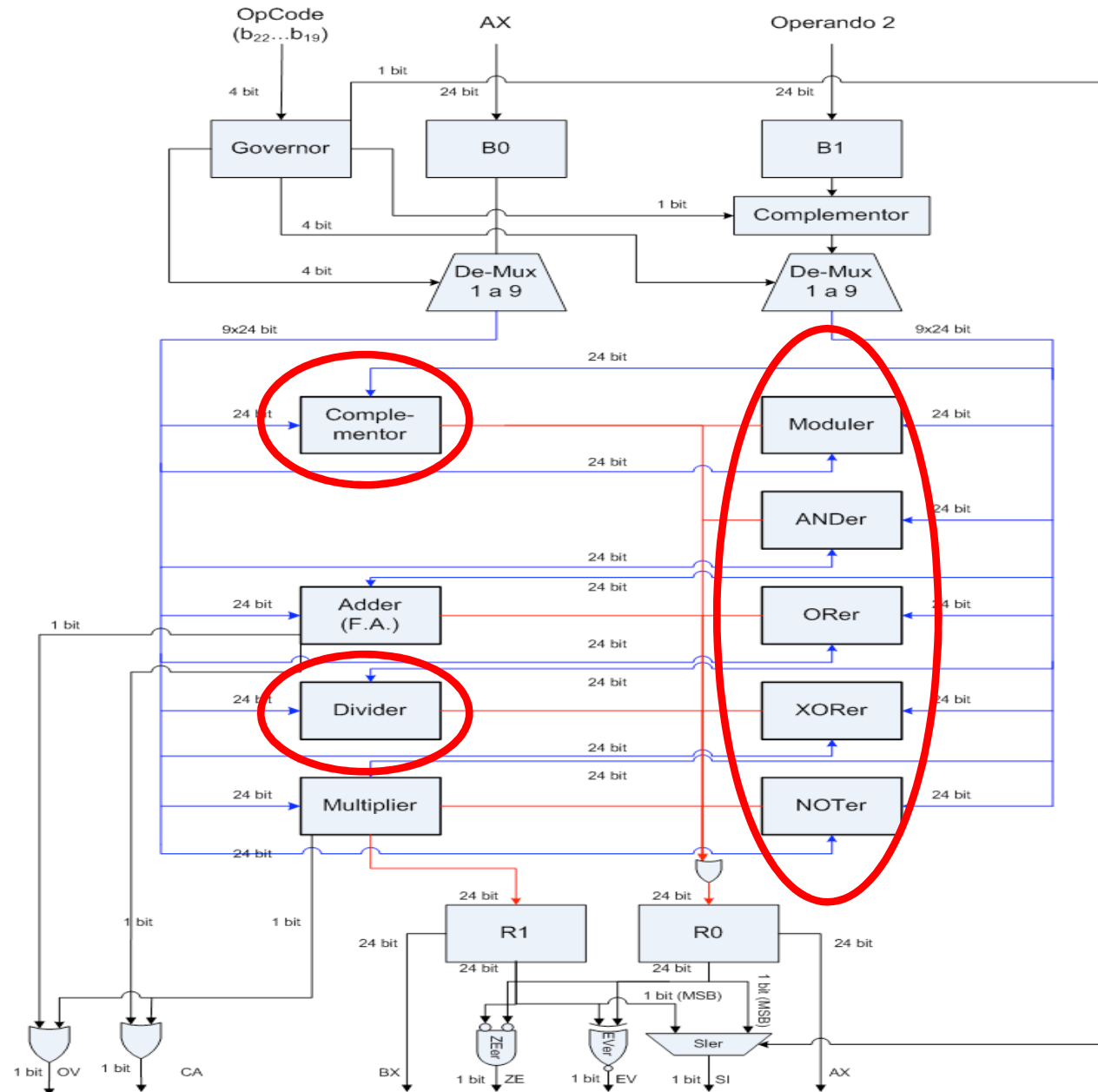
Una semplice ALU in grado di effettuare somme e sottrazioni e moltiplicazioni



La ALU di vCPU

- Completiamo ora la descrizione della struttura della ALU di vCPU, con la presentazione degli altri moduli necessari per implementare il resto delle operazioni
 - Divisione
 - Operazioni booleane

La ALU di vCPU



La ALU di vCPU

Elenco completo dei moduli presenti :

- **Adder** effettua la somma di due interi relativi.

$$R0 = \text{Operando1} + \text{Operando2}$$

- **Complementor** restituisce l'opposto di un intero relativo.

$$R0 = -\text{Operando}$$

N.B. L'operando può essere uno qualunque dei due ingressi

La ALU di vCPU

- **Multiplier** effettua il prodotto di due interi relativi.

$$R1, R0 = \text{Operando1} * \text{Operando2}$$

- **Divider** effettua la divisione tra due interi relativi.

$$R0 = \text{Operando1} / \text{Operando2}$$

- **Moduler** effettua il modulo della divisione tra due interi relativi.

$$R0 = \text{Operando1} \text{ MOD } \text{Operando2}$$

- **ANDer** effettua l'AND logico bit a bit tra due interi relativi.

$$R0 = \text{Operando1} \text{ AND } \text{Operando2}$$

La ALU di vCPU

- **ORer** effettua l'OR logico bit a bit tra due interi relativi.

$$R0 = \text{Operando1} \text{ OR } \text{Operando2}$$

- **XORer** effettua l'OR esclusivo bit a bit tra due interi relativi.

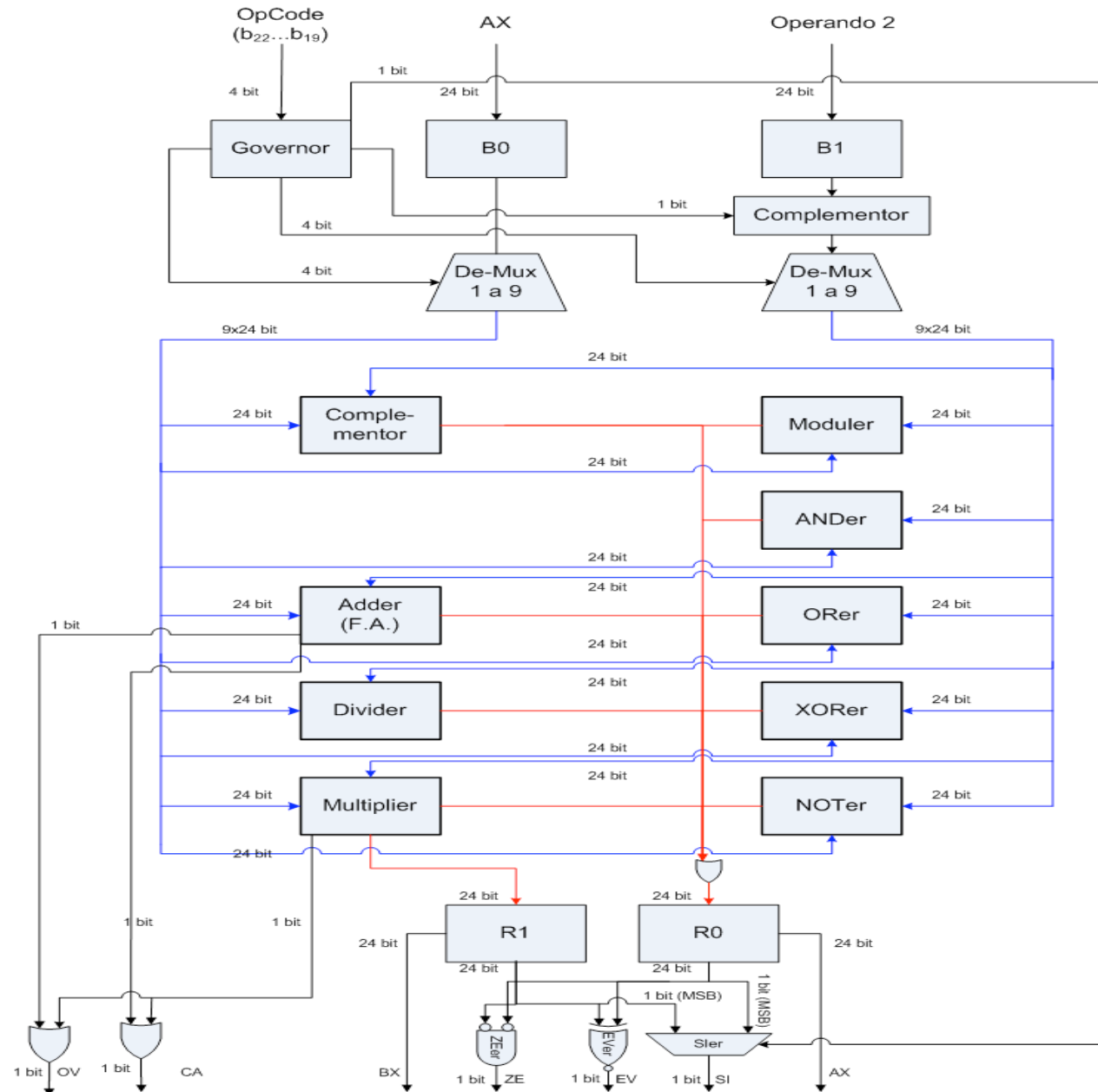
$$R0 = \text{Operando1} \text{ XOR } \text{Operando2}$$

- **NOTer** effettua il NOT logico bit a bit di un intero relativo.

$$R0 = \text{NOT}(\text{Operando})$$

N.B. L'operando può essere uno qualunque dei due ingressi

La ALU di vCPU



La ALU di vCPU

- La struttura del Multiplier e del Divider, e dei relativi componenti interni, non verrà esaminata in maggior dettaglio.
- Per ulteriori informazioni relative a tale struttura si rimanda alla tesi del Dott. Mauro Codella.

Nella prossima lezione...

- ▣ Vedremo in dettaglio le istruzioni e il loro formato
- ▣ Faremo esercizi utilizzando l'emulatore Eniac.

Eniac

- ❑ Emulatore di una CPU virtuale progettata e realizzata dal Dott. Mauro Codella e Dott. Dario Dussoni nell'ambito delle loro Tesi di Laurea con il Prof. Enrico Nardelli.
- ❑ Attualmente il software è reperibile alla seguente URL :

<http://sourceforge.net/projects/eniac/>