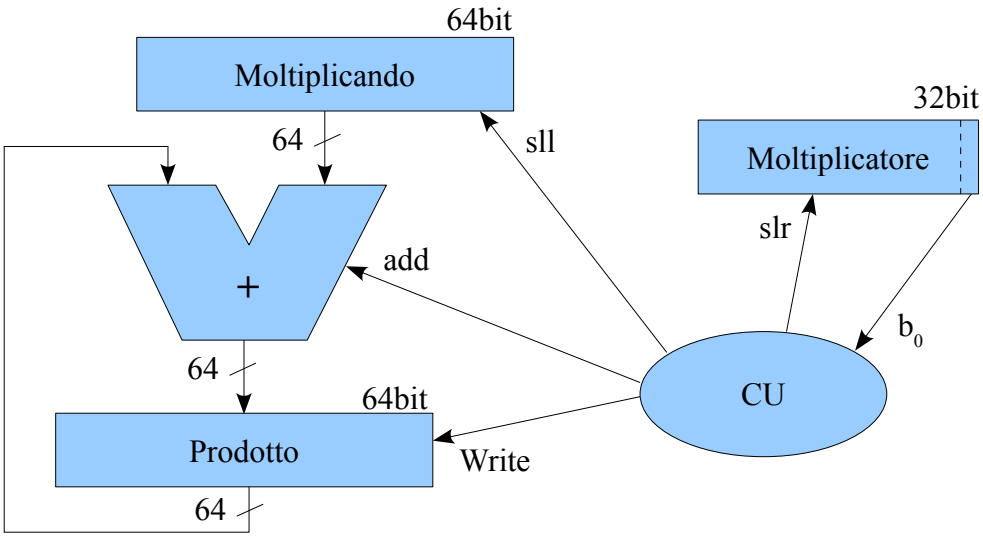
Moltiplicatore firmware – Ottimizzazione I

Di seguito è riportato lo schema di un moltiplicatore firmware ottimizzato nella gestione del moltiplicatore (ottimizzato I).



Il registro moltiplicando (MD) viene inizializzato ponendo il moltiplicando nella parte bassa e ponendo la parte alta, i bit da 32 a 63, a **0**. Il registro moltiplicatore (MT) viene inizializzato con moltiplicatore. Il registro prodotto (P) viene inizializzato interamente a **0**. Formalmente:

$$MD = \mathbf{0}^{32BIT}|\langle moltiplicando \rangle$$

$$MT = \langle moltiplicatore \rangle$$

$$P = \mathbf{0}^{64BIT}$$

Poiché i calcoli rimarranno sempre su 64 bit non occorre gestire il riporto in uscita dalla ALU.

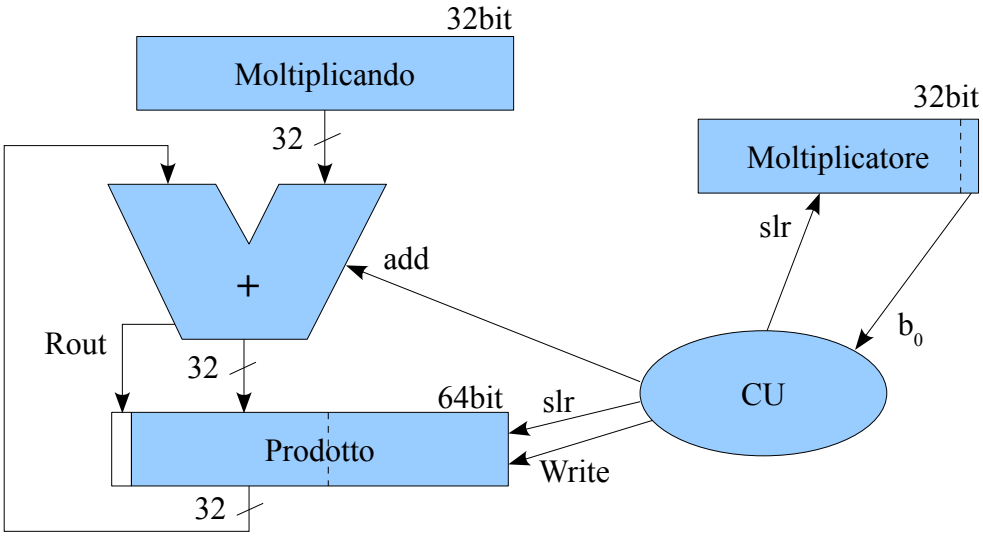
*Es: 6*2 (considerare parole di 4bit)*

Sol: Siccome si usano parole di 4 bit, occorre una ALU e registri MT e P da 4x2=8 bit. Occorrono 4 iterazioni, come il numero di bit.

Iterazione	Nota	MT	MD	P
0	Init	**001<u>0</u>**	0000.**0110**	0000.0000
1	MT.b0=0 → NOP	0010	0000.0110	0000.0000
	sll MD	0010	←**0000.110**0	0000.0000
	slr MT	→0**00<u>1</u>**	0000.1100	0000.0000
2	MT.b0=1 → P+=MD	0001	0000.1100	**0000.1100**
	sll MD	0001	←**0001.100**0	0000.1100
	slr MT	→0**00<u>0</u>**	0001.1000	0000.1100
3	MT.b0=0 → NOP	0000	0001.1000	0000.1100
	sll MD	0000	←**0011.0000**	0000.1100
	slr MT	→0**00<u>0</u>**	0011.0000	0000.1100
4	MT.b0=0 → NOP	0000	0011.0000	0000.1100
	sll MD	0000	←**0110.0000**	0000.1100
	slr MT	→**0000**	0110.0000	<u>0000.1100=12_{10}</u>

Moltiplicatore firmware – Ottimizzazione II

Di seguito è riportato lo schema di un moltiplicatore firmware ottimizzato anche nella dimensione del moltiplicando (ottimizzato II).



Il registro moltiplicando (MD) viene inizializzato con il moltiplicando e non viene shiftato durante l'esecuzione dell'algoritmo. Il registro moltiplicatore (MT) viene inizializzato con moltiplicatore. Il registro prodotto (P) viene inizializzato interamente a **0** e ad ogni iterazione viene shiftato a destra di una posizione. Formalmente:

MD = <moltiplicando>

MT = <moltiplicatore>

Rout|P = $\mathbf{0}|\mathbf{0}^{64BIT}$

Poiché i calcoli possono eccedere i 32 bit occorre gestire il riporto in uscita dalla ALU.

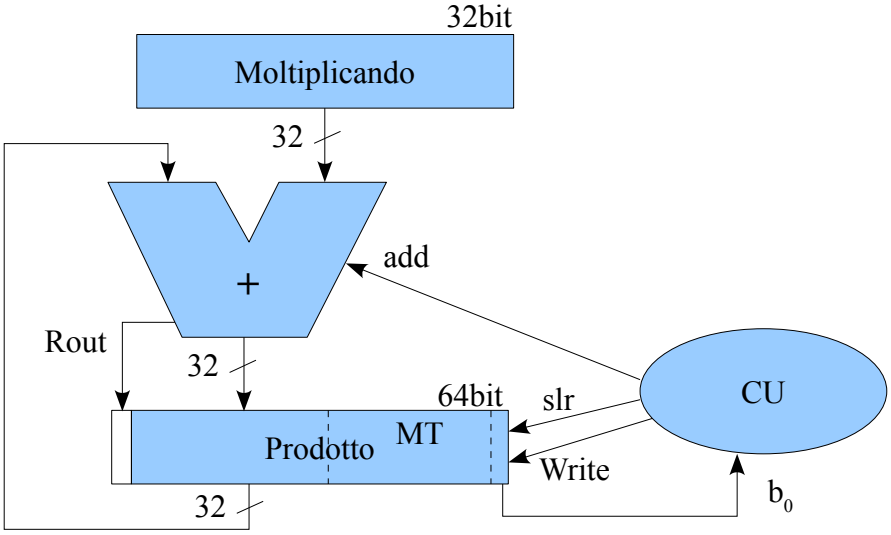
*Es: 6*2 (considerare parole di 4bit)*

Sol: Occorrono 4 iterazioni, come il numero di bit.

Iterazione	Nota	MT	MD	Rout.P
0	Init	**0010**	**0110**	0.0000.0000
1	MT.b0=0 → NOP	0010	0110	0.0000.0000
	slr P	0010	0110	0.**0000.0000**
	slr MT	0**001**	0110	0.0000.0000
2	MT.b0=1 → +MD	0001	0110	**0.0110**.0000
	slr P	0001	0110	0.**0011.0000**
	slr MT	0**000**	0110	0.0011.0000
3	MT.b0=0 → NOP	0000	0110	0.0011.0000
	slr P	0000	0110	0.**0001.1000**
	slr MT	0**000**	0110	0.0001.1000
4	MT.b0=0 → NOP	0000	0110	0.0001.1000
	slr P	0000	0110	0.**0000.1100**
	slr MT	0**000**	0110	0.0000.1100=12_{10}

Moltiplicatore firmware – Ottimizzazione III

Di seguito è riportato lo schema di un moltiplicatore firmware ottimizzato usando la parte bassa del prodotto per memorizzare il moltiplicatore (ottimizzato III).



Il registro moltiplicando (MD) viene inizializzato con il moltiplicando e non viene shiftato durante l'esecuzione dell'algoritmo. Il registro prodotto (P) viene inizializzato nella parte alta a 0 mentre la parte bassa viene inizializzata con il moltiplicatore (MT). Ad ogni iterazione viene shiftato a destra di una posizione. Formalmente:

$$MD = \langle moltiplicando \rangle$$

$$Rout|P = \mathbf{0}|\mathbf{0}^{32BIT}|\langle moltiplicatore \rangle$$

Poiché i calcoli possono eccedere i 32 bit occorre gestire il riporto in uscita dalla ALU.

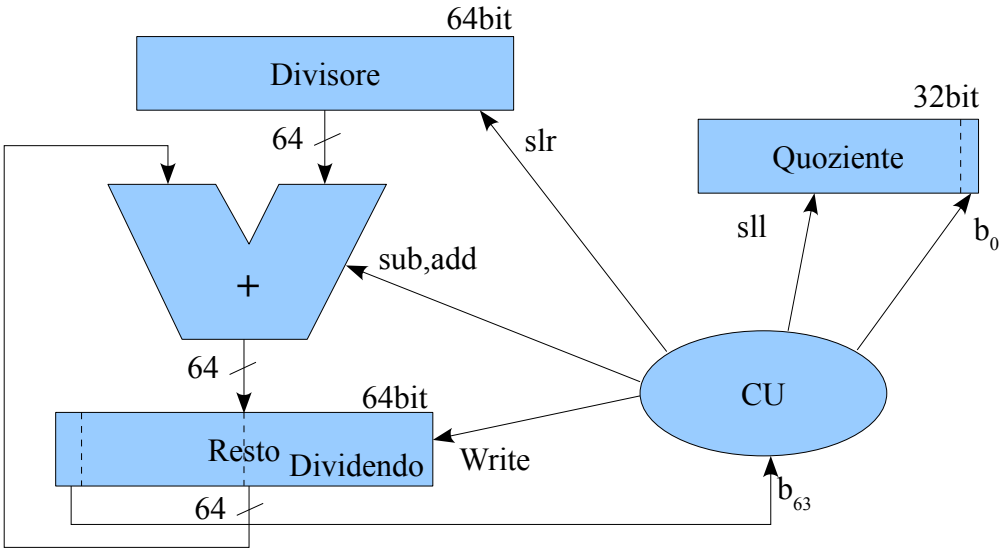
*Es: 6*2 (considerare parole di 4bit)*

Sol: Occorrono 4 iterazioni, come il numero di bit.

Iterazione	Nota	MD	Rout.P
0	Init	**0110**	0.0000.**0010**
1	P.b0=0 → NOP	0110	0.0000.0010
	slr P	0110	0.**0000.0001**
2	P.b0=1 → +MD	0110	**0.0110**.0001
	slr P	0110	0.**0011.0000**
3	P.b0=0 → NOP	0110	0.0011.0000
	slr P	0110	0.**0001.1000**
4	P.b0=0 → NOP	0110	0.0001.1000
	slr P	0110	0.**0000.1100**=12_{10}

Divisore firmware

Di seguito è riportato lo schema di un divisore firmware



Il registro divisore (DT) viene inizializzato ponendo il divisore nella parte alta e ponendo la parte bassa, i bit da 0 a 31, a **0**. Il registro Quoziente (Q) viene inizializzato a **0**. Il registro Resto (Rem) viene inizializzato nella parte bassa con il dividendo (DD) e nella parte alta a **0**. Formalmente:

$$DT = <divisore>|\ \mathbf{0}^{32BIT}$$

$$Q = \mathbf{0}^{32BIT}$$

$$Rem = \mathbf{0}^{32BIT}|<dividendo>$$

Poiché i calcoli rimarranno sempre su 64 bit non occorre gestire il riporto in uscita dalla ALU.

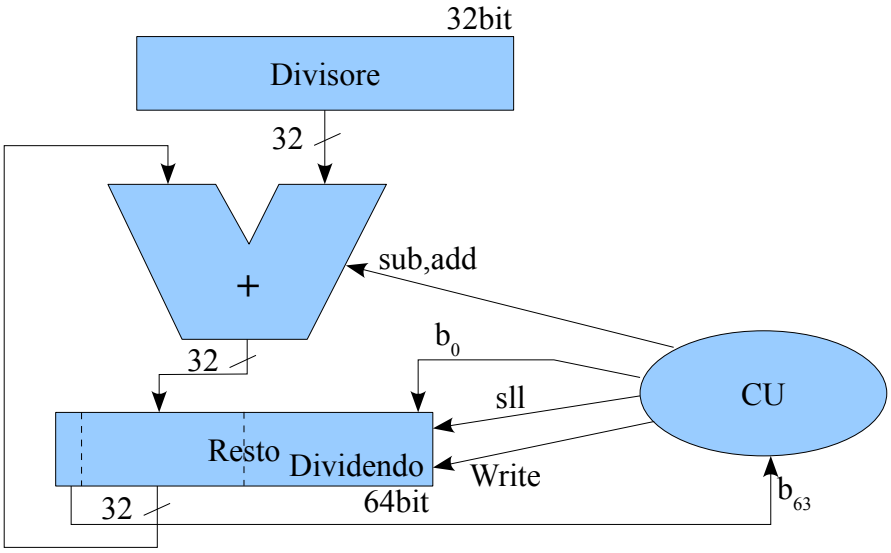
Es: 9/2 (considerare parole di 4bit)

Sol: Siccome si usano parole di 4 bit, occorre una ALU e il registro P da 4x2=8 bit. Occorrono 5 iterazioni, il numero di bit +1.

Iterazione	Nota	Q	DT	Rem
0	Init	0000	**0010**.0000	0000.**1001**
1	Rem=Rem-DT	0000	0010.0000	**1110.1001**
	$Rem.b_{63}=1 \rightarrow$ +DT,sll Q, $Q.b_0=0$	**0000**	0010.0000	0000.1001
	slr DT	0000	**0001.0000**	0000.1001
2	Rem=Rem-DT	0000	0001.0000	**1111.1001**
	$Rem.b_{63}=1 \rightarrow$ +DT,sll Q, $Q.b_0=0$	**0000**	0001.0000	0000.1001
	slr DT	0000	**0000.1000**	0000.1001
3	Rem=Rem-DT	0000	0000.1000	**0000.0001**
	$Rem.b_{63}=0 \rightarrow$ sll Q, $Q.b_0=1$	**0001**	0000.1000	0000.0001
	slr DT	0001	**0000.0100**	0000.0001
4	Rem=Rem-DT	0001	0000.0100	**1111.1101**
	$Rem.b_{63}=1 \rightarrow$ +DT,sll Q, $Q.b_0=0$	**0010**	0000.0100	0000.0001
	slr DT	0010	**0000.0010**	0000.0001
5	Rem=Rem-DT	0010	0000.0010	**1111.1111**
	$Rem.b_{63}=1 \rightarrow$ +DT,sll Q, $Q.b_0=0$	**0100**	0011.0000	0000.0001
	slr DT	$0100=4_{10}$	**0000.0001**	$0000.0001=1_{10}$

Divisore firmware – Ottimizzazione

Di seguito è riportato lo schema di un divisore firmware ottimizzato.



Il registro divisore (DT) viene inizializzato con il divisore e non viene shiftato durante l'esecuzione dell'algoritmo.I l registro Resto (Rem) viene inizializzato nella parte bassa con il dividendo (DD) e nella parte alta a **0**. il quoziente viene via via accumulato nella parte bassa. Formalmente:

$$DT = \langle divisore \rangle$$

$$Rem = \mathbf{0}^{32BIT} | \langle dividendo \rangle$$

Poiché i calcoli rimarranno sempre su 32 bit non occorre gestire il riporto in uscita dalla ALU.

Es: 9/2 (considerare parole di 4bit)

Sol: Siccome si usano parole di 4 bit, occorre una ALU e il registro P da 4x2=8 bit. Occorrono 5 iterazioni, il numero di bit +1.

Iterazione	Nota	DT	Rem
0	Init	0010	0000.**1001**
1	Rem.Hi=Rem.Hi-DT	0010	**1**110.1001
	$Rem.b_{63}=1 \rightarrow$ +DT,sll Rem, $Rem.b_0=0$	0010	**0001.0010**
2	Rem=Rem-DT	0010	**1**111.0010
	$Rem.b_{63}=1 \rightarrow$ +DT,sll Rem, $Rem.b_0=0$	0010	**0010.0100**
3	Rem=Rem-DT	0010	**0**000.0100
	$Rem.b_{63}=0 \rightarrow$ sll Rem, $Rem.b_0=1$	0010	**0000.1001**
4	Rem=Rem-DT	0010	**1110.1001**
	$Rem.b_{63}=1 \rightarrow$ +DT,sll Rem, $Rem.b_0=0$	0010	**0001.0010**
5	Rem=Rem-DT	0010	**1**111.0010
	$Rem.b_{63}=1 \rightarrow$ +DT,sll Rem, $Rem.b_0=0$	0010	**0010.0100**=4_{10}