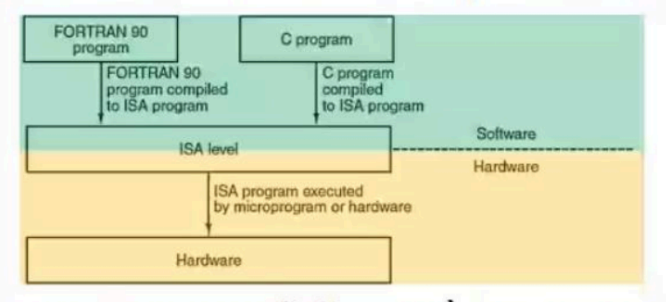
Quinto capitolo

Livello di Macroarchitettura

> Il **livello ISA** rappresenta **l'interfaccia tra il SW e HW,** quindi è il linguaggio che entrambi possono comprendere.

Anche se sarebbe possibile disporre di un hardware in grado di eseguire direttamente programmi scritti in C, Java o altri linguaggi di alto livello, non è una delle migliori scelte, in quanto andremmo a perderci di prestazioni. Oltre a ciò, per ragioni di praticità, è auspicabile che i computer siano capaci di eseguire codice scritto in più linguaggi, invece che in uno solo.



L'approccio ideale consiste nel partire da vari linguaggi d'alto livello, per poi tradurli in una forma intermedia comune, ovvero il livello ISA, e quindi **costruire l'HW in grado di eseguire direttamente i programmi di livello ISA.**

Inoltre, il livello ISA deve essere **retrocompatibile,** quindi deve essere in grado di eseguire vecchi programmi senza modifiche.

Un'altra proprietà del livello ISA è che la maggior parte dei processori è dotata di almeno **due modalità d'esecuzione:**

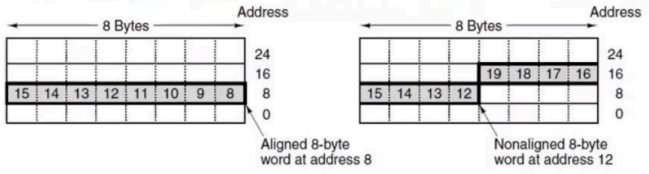
- **modalità kernel →** serve a eseguire il sistema operativo e permette l'esecuzione di tutte le istruzioni;
- **modalità utente →** ha lo scopo di eseguire i programmi applicativi e non consente l'esecuzione di certe istruzioni (come la manipolazione della cache).

L'ISA si compone:

- Di un modello di memoria;
- Dell'insieme dei registri;
- Dei tipi di dati possibili;
- Dell'insieme delle istruzioni.

Modelli di memoria

In genere, tutti i computer suddividono la memoria in celle adiacenti di un byte che sono a loro volta raggruppati in gruppi di 4 o 8 byte.



Molte architetture esigono che le parole siano allineate lungo le loro estremità: per esempio, una parola di 4 byte può cominciare agli indirizzi 0, 4, 8 e così via.

Nonostante l'allineamento ha vantaggi in termini di efficienza, ha vantaggi in termini di costi (richiede nel chip funzionalità logiche supplementari, che lo rende più grande e costoso).

I processori a livello ISA dispongono di uno spazio lineare degli indirizzi, che si estende dall'indirizzo 0 fino a 2^32 o 2^64. Tuttavia esistono macchine che dispongono di spazi degli indirizzi separati per le istruzioni e per i dati. Nonostante lo schema sia più complesso, presenta due vantaggi:

- È possibile referenziare 2^32 byte di programma e 2^32 byte di dati usando indirizzi di soli 32 bit.
- Poiché le scritture avvengono sempre nello spazio dei dati, diviene impossibile sovrascrivere il programma accidentalmente; inoltre, sono più difficili gli attacchi malware, in quanto il software maligno non può accedere al programma.

I registri

Ogni computer dispone di qualche registro visibile a livello ISA. Il loro compito è **il controllo dell'esecuzione del programma, il contenimento dei risultati temporanei o altro.**

Alcuni registri, come MAR, non sono visibili a livello ISA; alcuni di loro, però, come il PC o SP, sono visibili a entrambi i livelli. D'altro canto i registri visibili a livello ISA sono sempre visibili a livello della microarchitettura, perché è lì che sono implementati.

I registri ISA possono essere suddivise in due categorie:

- **registri specializzati →** PC, SP e altri registri dedicati a funzioni specifiche;
- **registri d'uso generale →**contengono le variabili locali più importanti e i risultati parziali del calcolo. La loro funzione principale è di consentire un accesso rapido a dati usati ricorrentemente (per evitare accessi in memoria).

I **registri di flag (o PSW)** è una specie di ibrido tra la modalità kernel e quella utente. Questo registro di controllo contiene vari bit necessari alla CPU, tra cui i codici di condizione, che riflettono lo stato del risultato dell'operazione più recente.

Tra i più importanti abbiamo:

N - 1 se il risultato è negativo

Z - 1 se il risultato è 0

V - 1 se il risultato ha causato un overfloat

C - 1 se il risultato ha causato un riporto oltre l'ultimo bit più significativo

A - 1 se si è verificato un riporto oltre il terzo bit

P - 1 se il risultato è pari

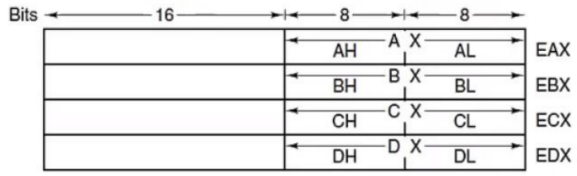
Panorama del livello ISA del Core I7

Il Core I7 è dotato di tre diverse modalità operative:

- **Modalità reale →** si comporta esattamente come un 8088, ma nel caso vengano eseguite istruzioni errate la macchina va in blocco;

- **Modalità virtuale →** permette di eseguire programmi 8088 in modo protetto e controllato da un vero sistema operativo;
- **Modalità protetta →** si comporta come un Core i7 con 4 privilegi controllati dai bit del PSW:
  - Livello 0 corrisponde alla modalità kernel e ha accesso completo alla macchina;
  - Livello 1 e 2 sono usati raramente;
  - Livello 3, usato per i programmi utente, blocca l'accesso a certe istruzioni critiche e controlla i registri per impedire a un programma utente malintenzionato di mandare in avaria l'intera macchina.

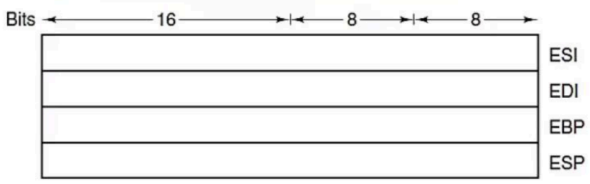
Il Core i7 dispone di 4 registri di uso generale:



- EAX → per le operazioni aritmetiche;
- EBX → puntatore a indirizzi di memoria;
- ECX → usato come contatore nei cicli;
- EDX → necessario per le moltiplicazioni/divisioni durante le quali, insieme a EAX, contiene prodotti e dividendi di 64 bit.

Tutti questi registri sono utilizzabili come registri di 16 bit o di 8 bit.

Ulteriori 4 registri di uso generale sono:



- ESI → puntatore in memoria alla stringa sorgente;
- EDI → puntatore in memoria alla stringa destinazione;
- EBP → puntatore usato per referenziare l'indirizzo base del record d'attivazione corrente;
- ESP → puntatore allo stack (come SP)

Tipi di dato

I tipi di dato sono raggruppabili in:

- **numerici → Interi (col segno o senza), Reali e Booleani** (si usa la rappresentazione numerica per indicare True/False);
- **non numerica → Caratteri, BitMap, Puntatore.**

Rappresentazione del tipo intero con segno

1) con bit di segno:

b_2	b_1	b_0	
0	0	0	4
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	0
1	0	1	-1
1	1	0	-2
1	1	1	-3

2) con complemento alla base: si inverte il numero (complemento a uno) e si somma uno:

Es/ $-15_{10} = ?_2$

$15_{10} = 00001111_2$ ⇒ $11110000 + 1 = 11110001$

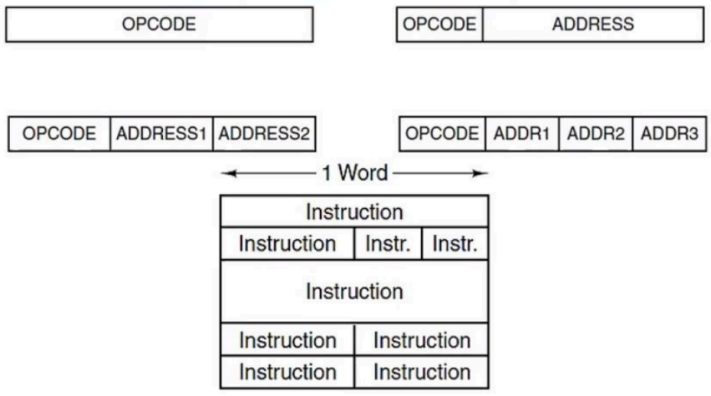
$-15_{10} = 11110001_2$

b_2	b_1	b_0	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	-4
1	0	1	-3
1	1	0	-2
1	1	1	-1

Formati di istruzione

Un'istruzione si compone di un **codice operativo (opcode)** che ne specifica il comportamento, e da nessuno o massimo tre indirizzi di riferimento degli operandi.

Alcune macchine hanno tutte le istruzioni della stessa lunghezza, alcune d'istruzioni di lunghezza diversa. La scelta di avere tutte le istruzioni della stessa lunghezza semplifica la loro decodifica, ma spesso implica uno spreco di spazio, visto che le istruzioni devono essere lunghe quanto la più lunga.



Criteri progettuali per i formati d'istruzioni

- **Le istruzioni corte preferibili a quelle lunghe:**
  - Meno utilizzo di memoria (programmi con istruzioni corte occupano meno memoria);
  - Permette di memorizzare maggiori quantità di istruzioni nelle cache dei processori che hanno una larghezza di banda limitata rispetto alla capacità di calcolo dei processori.

  Minimizzare la dimensione delle istruzioni potrebbe però rendere difficoltosa la decodifica.
- **Prevedere nel formato delle istruzioni spazio sufficiente a esprimere tutte le operazioni volute;**

- **Il numero di bit da utilizzare nell'indirizzamento della memoria, uno spazio di indirizzamento ampio conduce ad istruzioni lunghe.**

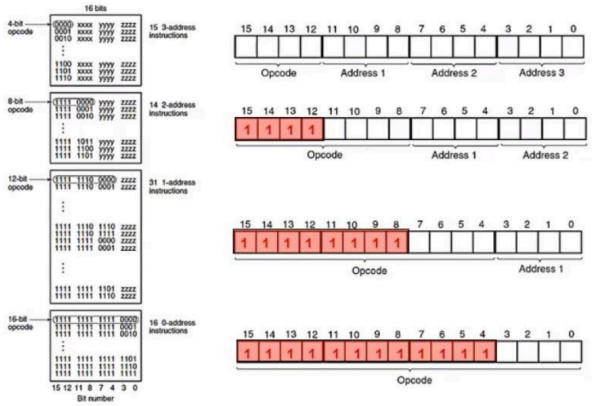
Codice operativo espandibile

Si consideri una macchina in cui le istruzioni sono lunghe 16 bit e gli indirizzi 4 bit. Un'architettura possibile sarebbe quella di comporre ogni istruzione con 4 bit di opcode e tre indirizzi, per un totale di 16 bit.

Nel caso in cui i progettisti necessitano di istruzioni con due indirizzi, potrebbero usare gli opcode da 8 a 14, interpretando il bit 15 diversamente. L'opcode 15 indica che il codice operativo è contenuto nei bit da 8 a 15 invece che nei bit da 12 a 15.

Possibili formati di una istruzione sono le istruzioni:

- senza indirizzi (formata dal solo campo OPCODE)
- a un indirizzo (formata dal campo OPCODE e l'indirizzo di 1 operando)
- a due indirizzi (formata dal campo OPCODE e gli indirizzi di 2 operandi)
- a tre indirizzi (formata dal campo OPCODE e gli indirizzi di 3 operandi)



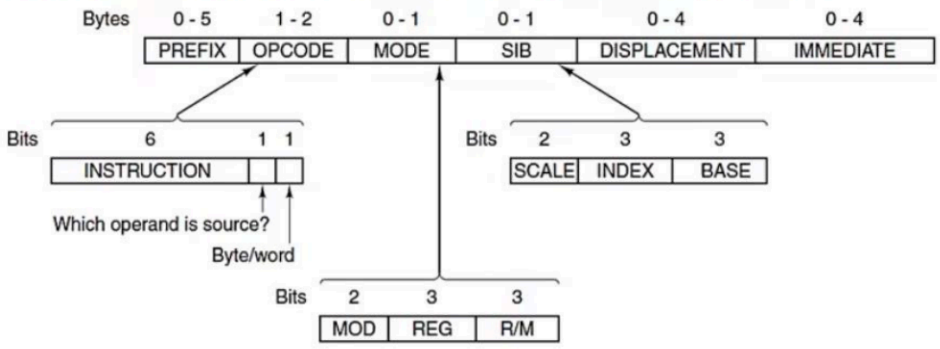
41

Ovviamente più indirizzi si inseriscono più è piccolo il campo OPCODE. Si assume inoltre che gli indirizzi abbiano stesse dimensioni.

Formati delle istruzioni del Core i7

Ci accorgiamo che l'istruzione non è fissa e certi codici operativi generano ulteriori informazioni → dinamicità.

Questo ci porta al concetto di non ortogalità → non abbiamo la stessa modalità di indirizzamento. Non avere questo, dal punto di vista della logica, porta complicazioni



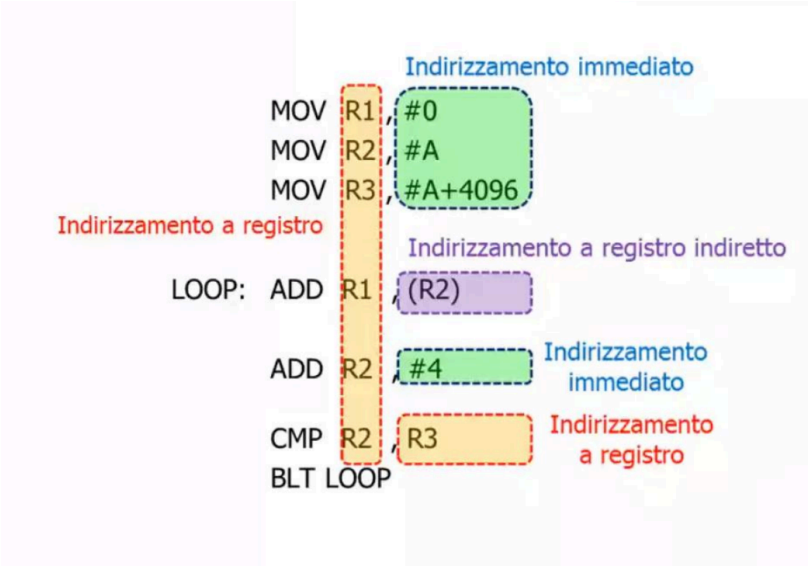
Modalità di indirizzamento

- **Immediato →** L'istruzione contiene il valore dell'operando

Esempio: Caricamento in R1 del valore 4

MOV	R1	4

- **Diretto →** L'istruzione contiene l'indirizzo di memoria completo dell'operando
- **A registro →** L'istruzione specifica un registro che contiene l'operando
- **A registro indiretto →** L'istruzione specifica un registro che contiene l'indirizzo in cui è presente l'operando in memoria. Quindi, specifica il puntatore all'indirizzo in memoria.
- **Indicizzato →** L'indirizzamento alla memoria si ottiene specificando un registro.
- **Indicizzato esteso →** L'indirizzo di memoria è calcolato sommando tra loro il contenuto di due registri più un offset (opzionale).
- **A stack →** L'operando è sulla cima dello stack.



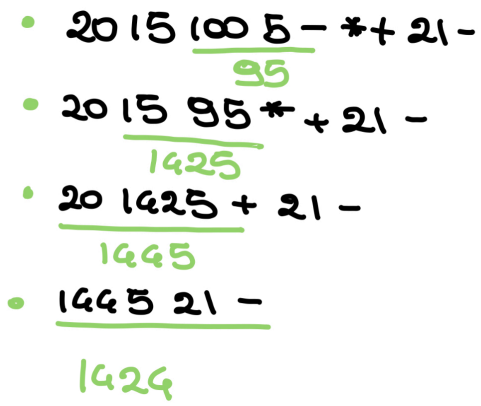
Notazione polacca inversa

> La **notazione polacca** è un modo per esprimere le espressioni matematiche in modo implicito senza utilizzo delle parentesi.

Regola di calcolo:

1. scandisci l'espressione da sinistra a destra fino a che raggiungi il primo operatore
2. applica l'operatore ai due operandi alla sua sinistra, sostituisci il risultato nell'espressione al posto di operandi e operatore

Esempio:

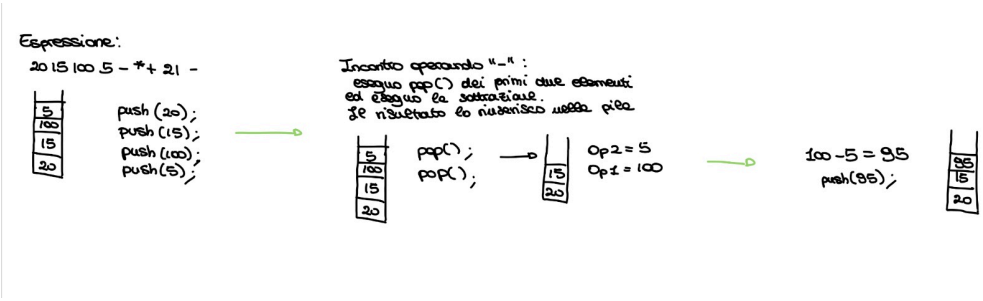


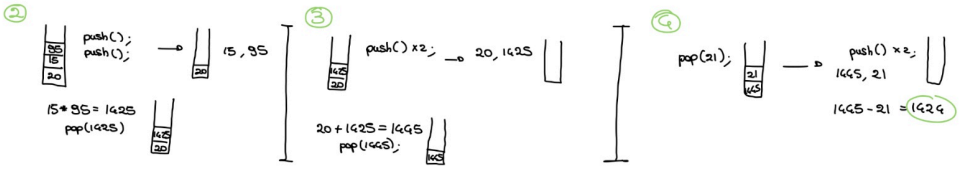
Algoritmo di calcolo di espressioni in notazione polacca inversa

- Scandisci l'espressione da sinistra a destra; per ogni operando, esegui *pop()* per inserirlo nella pila. Quando incontri un operatore, sospendi la scansione.
- Incontrato un operatore, esegui *push()* due volte, ricavando i primi due elementi che si trovano al di sopra della pila. Il primo elemento ricavato lo chiameremo Op2, il secondo Op1;
- Eseguiamo il calcolo con Op1, Op2 e l'operatore incontrato nella scansione:

  Op1 *operatore* Op2 = Op3
- Inseriamo Op3 nella pila con push();
- Riprendiamo dal punto 1) finché non abbiamo scansionato tutta l'espressione (oppure finché la pila non è vuota.

Esempio:





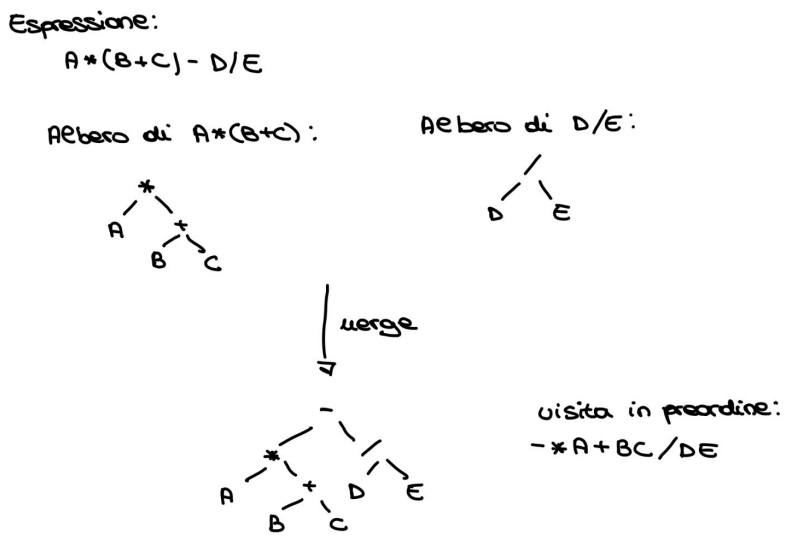
Da espressione "normale" a espressione in notazione polacca inversa

- Data l'espressione, costruiamo un albero, in cui

- ogni operatore è un nodo interno
- ogni operando è una foglia

- Per ricavare l'espressione, eseguiamo una visita dell'albero in preordine (rad-sx-dx)

Esempio:

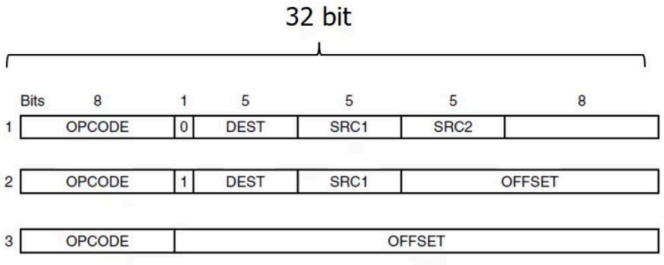


Ortogonalità

Dal punto di vista SW, le istruzioni e l'indirizzamento dovrebbero manifestare una struttura regolare, con un numero minimo di formati d'istruzioni. Tale struttura agevolerebbe molto il compilatore a produrre codice di qualità. Gli opcode dovrebbero consentire tutte le modalità d'indirizzamento e ogni registro dovrebbe essere utilizzabile da tutte le modalità a registro.

Quando tutte le modalità di indirizzamento sono utilizzabili con tutti i codici operativi si dice che i codici e le modalità di indirizzamento sono tra loro **ortogonali.**

Esempio di istruzione ortogonale



Tipi d'istruzioni

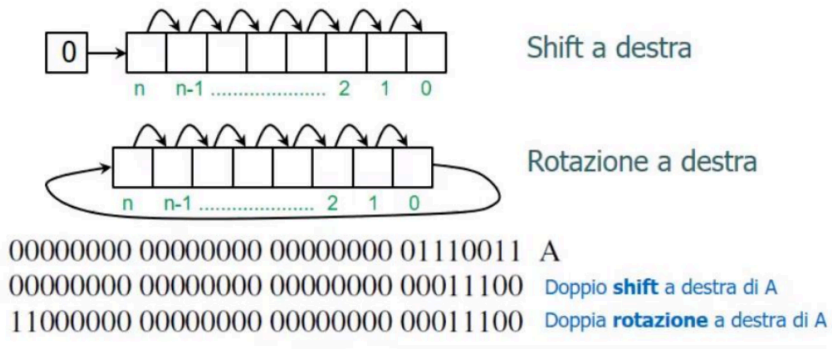
Istruzioni unarie su bit

Le operazioni unarie prendono in ingresso un operando e restituiscono un risultato.

Le **istruzioni per lo scorrimento o la rotazione del contenuto di una parola** si rivelano molto utili:

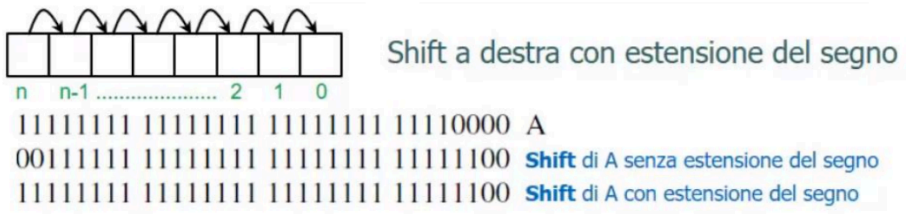
- Le operazioni di scorrimento possono spostare i bit verso destra o sinistra, e i bit che fuoriescono dalla parola vanno perduti;
- Le rotazioni sono scorrimenti in cui i bit che escono da un'estremità della parola riappaiono all'altra estremità.

Esempio:



Gli scorrimenti verso destra sono spesso usati in associazione con l'estensione del segno, ovvero le posizioni all'estremità di sinistra lasciate vacanti dallo scorrimento vengono riempite con il bit di segno originario, 0 o 1.

Esempio:



Lo scorrimento può essere usato per accelerare certe operazioni aritmetiche. Consideriamo il calcolo di $18 * n$, dove n è un numero intero positivo. Poiché

18*n = 16*n + 2*n

è possibile ottenere $16 * n$ facendo scorrere n verso sinistra di 4 bit ($16 = 2^4$) e $2 * n$ facendo scorrere n verso sinistra di 1 bit ($2 = 2^1$). La somma dei due numeri è $18 * n$.

Nel caso dello scorrimento di numeri negativi, anche se fatto con estensione del segno, non produce la moltiplicazione per 2, a differenza dello scorrimento a destra che simula la divisione correttamente.