1 Lezione del 29-10-24

1.1 Riduzione di campo di interi

Vogliamo creare un circuito che passa dalla rappresentazione A su $n+1$ cifre di un numero intero a, ad un A^{RID} su n cifre, che rappresenta sempre a. Chiaramente questo non è sempre possibile, e vale soltanto se:

$$a \in \left[-\frac{\beta^n}{2}, \frac{\beta^n}{2} - 1\right] \subset \left[-\frac{\beta^{n+1}}{2}, \frac{\beta^{n+1}}{2} - 1\right]$$

Avremo quindi bisogno di un flag di overflow `ow`, per indicare la non rappresentabilità. Verifichiamo da quanto visto sulle estensioni di campo, che i numeri che rispettano tale proprietà sono i tali per cui MSD $= 0$ e la cifra successiva $a_{n-1} < \frac{\beta}{2}$, e i tali per cui MSD $= \beta - 1$ e la cifra successiva $a_{n-1} \geq \frac{\beta}{2}$. Quindi:

$$\texttt{ow} = 0 \Leftrightarrow \left(a_n = 0 \wedge a_{n-1} < \frac{\beta}{2}\right) \vee \left(a_n = \beta - 1 \wedge a_{n-1} \geq \frac{\beta}{2}\right)$$

Abbiamo sul grafico a farfalla adattato all'estensione su $n+1$ bit, che queste regole isolano le due sezioni del campo di numeri estesi ($\left[-\frac{\beta^{n+1}}{2}, \frac{\beta^{n+1}}{2} - 1\right]$) che hanno riscontro nel campo ridotto ($\left[-\frac{\beta^n}{2}, \frac{\beta^n}{2} - 1\right]$).

Quindi, in questo caso, il numero a è rappresentabile su n cifre, e si può calcolare il ridotto A^{RID} semplicemente rimuovendo l'ultima cifra, cioè calcolando:

$$A^{RID} = |A|_{\beta^n}$$

Chiamiamo il circuito che riconosce la non riducibilità **circuito di overflow**. In base 2, si ha rispetto alle cifre che $a_{n-1} < \frac{\beta}{2}$ vale se $a_{n-1} = 0$, e viceversa $a_{n-1} \geq \frac{\beta}{2}$ vale se $a_{n-1} = 1$, cioè un numero non è rappresentabile su $n-1$ bit se le sue due cifre più significative sono uguali. In questo modo il circuito si traduce in un confronto fra le due cifre più significative a_n e a_{n-1}, che si fa con uno XOR.

1.1.1 Moltiplicazione di interi per potenza della base

Vediamo come si realizza un moltiplicatore per $b = \beta \cdot a$, dato $A = (an - 1a_{n-2}...a_0)$ rappresentante a su n cifre, B rappresentante b su $n+1$ cifre.

Vogliamo chiederci prima di tutto se b è sempre rapprsentabile da B su $n+1$ cifre. Questo è vero, in quanto si può dimostrare che:

$$B = \beta \cdot A$$

Questo viene da:

$$L: \quad B = \begin{cases} b = \beta \cdot a, & 0 \leq a < \frac{\beta^n}{2} \\ \beta^{n+1} + b = \beta^{n+1} + \beta \cdot a = \beta \cdot (\beta^n + a), & -\frac{\beta^n}{2} \leq a < 0 \end{cases}$$

dove si nota che a e $\beta^n + a$ valgono A nei rispettivi campi di esistenza. Si applica quindi quanto conoscevamo sulle moltiplicazioni per potenze di base su naturali, e il prodotto sta su $n+1$ cifre.

Per prodotti con potenze ulteriori della base, diciamo β^k, si ha che:

$$b = \beta^k \cdot a \equiv B = \beta^k \cdot A$$

e quindi il risultato starà su $n+k$ cifre.

1.1.2 Divisione per potenza della base

Vogliamo fare l'operazione equivalente per le divisioni, cioè dato A rappresentante a su $n+1$ cifre, trovare B rappresentante b su n cifre tale per cui $b = \left\lfloor \frac{a}{\beta} \right\rfloor$.

Possiamo dimostrare, come prima, che:

$$B = \left\lfloor \frac{A}{\beta} \right\rfloor$$

Per fare ciò, approfittiamo della proprietà vista sul complemento a radice che ci permette di rappresentare B come $|b|_{\beta^n}$:

$$B = \left|\left\lfloor \frac{a}{\beta} \right\rfloor\right|_{\beta^n} = \left|\left\lfloor \frac{\lfloor a/\beta^{n+1} \rfloor \cdot \beta^{n+1} + |a|_{\beta^n+1}}{\beta} \right\rfloor\right|_{\beta^n} = \left|\left\lfloor \lfloor a/\beta^{n+1} \rfloor \cdot \beta^n + \frac{|a|_{\beta^{n+1}}}{\beta} \right\rfloor\right|_{\beta^n}$$

$$\left|\left\lfloor \frac{|a|_{\beta^{n+1}}}{\beta} \right\rfloor\right|_{\beta^n} = \left|\left\lfloor \frac{A}{\beta} \right\rfloor\right| = \left\lfloor \frac{A}{\beta} \right\rfloor$$

Abbiamo quindi che possiamo sfruttare quanto avevamo detto sulla divisione per potenze di basi su naturali, e il quoziente sta su n cifre.

Per divisioni con potenze ulteriori della base, diciamo β^k, si ha che:

$$b = \left\lfloor \frac{a}{\beta^k} \right\rfloor \equiv B = \left\lfloor \frac{A}{\beta^k} \right\rfloor$$

e quindi il risultato starà su $n-k$ cifre (o A dovrà stare su $n+k$ cifre rispetto a B, solita cosa).

1.1.3 Note sugli shift logico e aritmetico

Abbiamo visto come sono state definiti operazioni diverse per lo shift logico (SH) e aritmetico (SA) in linguaggio assembly. Abbiamo visto adesso, però, che moltiplicazione e divisione per la base si fanno allo stesso modo sia su interi che su naturali.

Possiamo dire che, nel caso dello shift a sinistra, effettivamente le operazioni eseguite dal calcolatore sono uguali sia nel caso di SHL che SAR. Per quanto riguarda lo shift a destra, invece, dobbiamo renderci conto che la $n-1$-esima cifra (quella che avevamo escluso dicendo che A su $n+1$ cifre va in B su n cifre) resterà comunque nella locazione di memoria, cioè non si possono ridimensionare le locazioni. C'è quindi una differenza sul modo in cui si popola l'$n-1$-esimo bit entrante: lo shift aritmetico SAL ripete il MSD (cioè estende l'intero su n cifre) e lo shift logico SHL introduce sempre zeri (ergo perde i segni nel caso di a ngativi).

1.2 Somma di interi

Dati A e B in base β su n cifre, rappresentanti rispettivamente gli interi a e b, vogliamo calcolare S su n cifre tale S rappresenta s e $s = a + b$. Abbiamo, che la somma potrebbe uscire dall'intervallo di rappresentabilità su n cifre, in quanto sta su:

$$-\beta^n \leq s \leq \beta^n - 2$$

e starebbe al massimo su $n+1$ cifre:

$$[-\beta^n, \beta^n - 2] \subset \left[-\frac{\beta^{n+1}}{2}, \frac{\beta^n + 1}{2} - 1 \right]$$

Abbiamo quindi bisogno di flag di overflow, `ow`.

Quando s è invece rappresentabile su n cifre, si ha che:

$$S = |s|_{\beta^n} = |a+b|_{\beta^n} = ||a|_{\beta^n} + |b|_{\beta^n}|_{\beta^n} = |A+B|_{\beta^n}$$

Questa è la proprietà fondamentale per cui si usa il complemento alla radice, e lo passiamo dimostrare da quanto già dimostrato sulle proprietà dell'operatore modulo.

Si può quindi usare un sommatore (che è quindi indifferente per naturali e interi), e l'unico problema resta determinare il flag `ow`.

L'unica cosa che dovremo aggiungere è un modo per calcolare il flag `ow`. Per adesso abbiamo dal sommatore l'uscita C_{out}, cioè il riporto della somma: questa non basta da sola a verificare la rappresentabilità del risultato. Notiamo che la somma è sempre rappresentabile estendendo gli ingressi a $n+1$ bit e riducendo in uscita. Se la riduzione è possibile, ergo le ultime due cifre più significative sono diverse, allora la somma è rappresentabile.

In una base arbitraria, per fare ciò devo effettivamente fare lo XOR delle due cifre, mentre in binario posso sfruttare le proprietà dello XOR, ricordando che internamente al full adder, la cifra in uscita s non è altro che $a \oplus b \oplus c$. Si ha quindi:

$$\texttt{ow} = s_n \oplus s_{n-1} = (a_n \oplus b_n \oplus c_n) \oplus (a_{n-1} \oplus b_{n-1} \oplus c_{n-1}) = 0 \oplus c_n \oplus c_{n-1} = c_n \oplus c_{n-1}$$

Possiamo quindi ricavare il flag `ow` confrontando il C^n_{out} con l'$n-1$-esimo C^{n-1}_{out}: a bit uguali sia `ow` falso, e viceversa, cioè si usa un singolo XOR

Si ha quindi che la stessa circuiteria esegue somme sia fra interi che fra naturali. In assembly, avevamo visto che la ADD esegue le stesse operazioni, ed è compito del programmatore controllare i flag di carry o di overflow a seconda di ciò che era andato a sommare (interi $\rightarrow$ overflow, naturali $\rightarrow$ carry).

1.3 Sottrazione di interi

La sottrazione fra interi è analoga alla somma: abbiamo sempre due A e B in base β su n cifre, e vogliamo trovare D sempre su n cifre tale per cui fra a, b e d rappresentati vale $a - b = d$. Si ha, prendendo il complemento a radice:

$$D = |d|_{\beta^n} = |a-b|_{\beta^n} = |a-b|_{\beta^n} = ||a|_{\beta^n} - |b|_{\beta^n}| = |A-B|_{\beta^n} = |A + \overline{B} + 1|_{\beta^n}$$

Come prima, abbiamo che il flag `ow` è dato dallo XOR degli ultimi due prestiti (prima erano riporti). Questo si dimostra analogamente a prima, prendendo il bit esteso:

$$D^{EST} = |d|_{\beta^{n+1}} = |a-b|_{\beta^{n+1}} = \left||a|_{\beta^{n+1}} - |b|_{\beta^{n+1}}\right|_{\beta^{n+1}}$$

$$= |A^{EST} - B^{EST}|_{\beta^{n+1}} = |A^{EST} + \overline{B^{EST}} - 1|_{\beta^{n+1}}$$

cioè si ha che sull'$n+1$-esimo bit la differenza è uguale prendendo le estensioni degli ingressi su $n+1$ bit, ergo la rappresentabilità è data dalla riducibilità del risultato su n bit, e quindi come prima dallo XOR sugli ultimi due prestiti.

1.4 Comparazione di numeri interi

Notiamo che c'è una differenza fra la comparazione fra interi e quella fra naturali. Per quanto riguarda l'uguaglianza $a = b$, abbiamo effettivamente la stessa cosa dei naturali.

Invece, per la minoranza $a < b$, non possiamo più controllare i prestiti uscenti. Dobbiamo quindi guardare il segno del risultato della sottrazione, che deve quindi poter essere svolta: si estende su $n+1$ cifre e si controlla la n esima cifra del risultato: questa varrà da $\text{sgn}(a-b)$, e quindi da flag di minoranza per $a < b$. Se non si fosse esteso su $n+1$ cifre, non avremmo potuto essere sicuro di non aver scartato eventuali valori negativi (negli intervalli di non rappresentabilità).

1.5 Moltiplicazione e divisione di interi

Moltiplicazioni e divisioni di interi riescono più facili se prima si converte in rappresentazione modulo e segno: i moduli si moltiplicano o dividono come naturali, e il segno viene determinato dai segni degli operandi attraverso la comune algebra alternante ($+ \cdot + = +$, $+ \cdot - = -$, $- \cdot - = +$).

Ricordiamo di aver già visto un circuito di conversione da CR a MS. Ci manca quindi il circuito di conversione opposto:

1.5.1 Conversone da MS a CR

Vogliamo una rete che prende in ingresso il valore assoluto su n cifre ed il segno della rappresentazione di un numero intero, e produce un uscita la sua rappresentazione in complemento alla radice su n cifre. Quest'operazione non è sempre possibile: abbiamo $-(\beta^n - 1) \leq a \leq \beta^n - 1$ in ingresso e $-\frac{\beta^n}{2} \leq a \leq \frac{\beta^n}{2} - 1$ in uscita.

Se l'operazione è fattibile, avremo che:

$$A = |a|_{\beta^n} = \begin{cases} |ABS_a|_{\beta^n}, & a \geq 0 \\ |-ABS_a|_{\beta^n}, & a < 0 \end{cases} = \begin{cases} |ABS_a|_{\beta^n}, & a \geq 0 \\ |\overline{ABS_a} + 1|_{\beta^n}, & a < 0 \end{cases}$$

Quindi si usa un multiplexer, con il segno della rappresentazione MS a variabile di controllo, che distingue fra la rappresentazione stessa ABS_a e il suo complemento (calcolato con un circuito di inversione e incremento).

Per quanto riguarda l'overflow, abbiamo invece che `ow` è impostato in due casi:

- Siamo fuori dal campo di rappresentabilità: questo si verifica quando $\text{abs}(a) > \frac{\beta^N}{2}$, cioè si è passati oltre agli $n-1$ bit su cui dobbiamo ridurre il modulo;
- Si è sull'unico valore positivo che non ha rappresentazione con $a = \frac{\beta^n}{2}$ e il bit di segno vale 0, cioè a è uguale al massimo rappresentabile $\frac{\beta^n}{2} - 1 + 1$.

Questo si sintetizza nella regola, espressa attraverso l'operatore ternario:

$$\texttt{ow} = \left(\left(\text{abs}(a) > \frac{\beta^n}{2}\right) \vee \left(\left(\text{abs}(a) = \frac{\beta^n}{2}\right) \wedge (\text{sgn}(a) = 1)\right)\right) \; ? \; 1 : 0$$

Occorre stare attenti fra la funzione segno e il valore del bit di segno, in quanto vale, come per la convenzione solita sul bit di segno:

$$\begin{cases} \text{sgn}(a) = 1 \Rightarrow \texttt{ow} = 0 \\ \text{sgn}(a) = -1 \Rightarrow \texttt{ow} = 1 \end{cases}$$

Riassumiamo brevemente come si svolge la conversione fra complemento a radice e rappresentazione modulo e segno. Abbiamo sostanzialmente che, salvo il caso della

conversione da MS a CR dove si può incappare in non rappresentabilità, la conversione da CR a MS e viceversa si fa sempre con un multiplexer che distingue fra la rappresentazione A presa così com'è e il suo complemento calcolato con inversione incremento. In particolare:

- Nel caso **CR a MS**, si prende A se per la cifra più significativa a_{n-1} vale $a_{n-1} \geq \frac{\beta}{2}$, altrimenti il complemento $\overline{A}$: questo significa che il multiplexer è pilotato dalla MSD;
- Nel caso **MS a CR**, si prende A se il bit di segno non è impostato, e il complemento $\overline{A}$ altrimenti: questo significa che il multiplexer è pilotato dal bit di segno.

1.5.2 Moltiplicazione

Per svolgere la moltiplicazione vogliamo quindi trasformare due ingressi A e B, rappresentanti gli interi a e b su n e m bit, nella loro rappresentazione MS come:

$$a \Rightarrow \text{sgn}(a), \text{abs}(a), \quad a \Rightarrow \text{sgn}(a), \text{abs}(a),$$

dove i segni stanno su un bit e i moduli su n e m bit.

Abbiamo che la moltiplicazione dei moduli di A e B è sempre rappresentabile, in quanto:

$$\text{sgn}(A) \cdot \text{sgn}(B) \leq \frac{\beta^n}{2} \cdot \frac{\beta^m}{2} = \frac{\beta^{n+m}}{2}$$

Osserviamo quindi che il prodotto intero p è:

$$p = \begin{cases} \text{abs}(a) \cdot \text{abs}(b), & \text{sgn}(a) = \text{sgn}(b) \\ -\text{abs}(a) \cdot \text{abs}(b), & \text{sgn}(a) \neq \text{sgn}(b) \end{cases}$$

ergo:

$$\begin{cases} \text{abs}(p) = \text{abs}(a) \cdot \text{abs}(b) \\ \text{sgn}(p) = \text{sgn}(a) \cdot \text{sgn}(b) \end{cases}$$

cioè quanto avevamo detto sulla rappresentazione modulo e segno per i prodotti.

Si ha quindi che il moltiplicatore fra interi si realizza convertendo gli ingressi da CR a MS, mandando i valori assoluti ad un moltiplicatore per naturali, e ricavando il segno del successivo convertitore da MS a CR (che ci darà il risultato) da uno XOR fra i segni degli MS in ingresso. L'overflow non è considerato in quanto non potra mai verificarsi (da sopra).

1.5.3 Divisione

Vogliamo calcolare, dati due naturali A e B rappresentanti gli interi a e b su n e m cifre, il quoziente Q e il resto R, rispettivamente su n e m cifre, tali che:

$$a = q \cdot b + r$$

Per svolgere questa divisione abbiamo bisogno di una riformulazione del teorema della divisione con resto che funzioni sull'anello $\mathbb{Z}$, in quanto adesso la semplice $a = q \cdot b + r$ con $r < b$ ammette infiniti valori di r (che può essere negativo). Decidiamo quindi di imporre:

- Il quoziente q è positivo se i segni di a e b sono concordi, e negativo viceversa;

- r e b sono uguali in segno.

Da qui si ha la proprietà più importante, cioè:

$$\begin{cases} \text{abs}(r) < \text{abs}(b) \\ \text{sgn}(r) = \text{sgn}(b) \end{cases}$$

Si verifica che questo significa che vogliamo i risultati che ci aspettiamo dalla comune divisione fra interi.

Abbiamo quindi, pensando in modulo e segno, che:

$$a = q \cdot b + r$$

diventa:

$$\text{sgn}(a) \cdot \text{abs}(a) = q \cdot \text{sgn}(b) \cdot \text{abs}(b) + \text{sgn}(r) \cdot \text{abs}(r)$$

Ma se avevamo $\text{sgn}(a) = \text{sgn}(r)$, allora:

$$\text{abs}(a) = (q \cdot \text{sgn}(b) \cdot \text{sgn}(a)) \cdot \text{abs}(b) + \text{abs}(r)$$

Si nota quindi che $q \cdot \text{sgn}(b) \cdot \text{sgn}(a)$ è semplicemente $\text{abs}(q)$, ergo si può rendere la divisione fra interi come la divisione fra i moduli di quegli interi, prendendo il segno separatamente (che è quello che avevamo fatto per la moltiplicazione).

Resta da trovare il valore del flag di non fattibilità `no_div`. Abbiamo, dal divisore fra naturali, che la rappresentabilità è data da:

$$\text{abs}(a) < \beta^n \cdot \text{abs}(q)$$

Questa condizione non basta, in quanto non si è ancora assicurato che l'intero in uscita sia rappresentabile su n cifre. Si prende quindi anche il flag di overflow `ow`, ricavato dalla conversione finale da MS a CR, e si mette a OR con il flag `no_div` che abbiamo ricavato dal circuito divisore.