Oggi vediamo un altro esercizio di programmazione in assembler. Il pretesto è di provare a immaginare una funzione matematica che non è già implementata a livello hardware. Vedremo quindi la divisione (tra numeri interi positivi in complemento a 2), i negativi li useremo per segnalare le condizioni di errore (come l’utilizzo di negativi e la divisione per zero). Il dividendo deve essere quindi maggiore o uguale di 0 e il divisore maggiore di 0.

Si inseriscono gli operandi nei registri 0 e 1. La divisione intera produce sia un quoziente che un resto. Pertanto il risultato, per una questione di semplicità lo inseriamo nel registro 2, mentre il resto va a finire all’interno del registro 0 (al posto del dividendo). Per poter chiamare la funzione due valori vanno inseriti nei registri r0 e r1 e poi bisogna inserire il valore del PC nel Link Pointer. A quel punto può iniziare la funzione. Finita la funzione si risposta il contenuto dell’LP nel PC.

L’idea dell’algoritmo e riprodurre quello che noi faremmo a mano (con la divisione in colonna). Per esempio se prendessimo il numero 36 e lo volessimo dividere per 5, prenderemmo la prima cifra, e vedendo che è < del divisore ci attaccheremmo anche la seconda. Poiché 36 invece è maggiore di 5 possiamo procedere e vediamo che 35 è il multiplo di 5 più grande <= 36. Il valore del quoziente (inizia da, ma in questo caso) è 7. Poiché 1 < 5, il valore che ci rimane diventa il resto.

Dobbiamo trasporre l’operazione in binario. Supponiamo di avere un valore simbolico chiamato div=64, che punta alla funzione divisione in RAM. Le operazioni che seguono sarebbero:

MOV R2, #-x (a causa di come sono codificate le istruzioni amber, non tutti i valori da 32 bit sono rappresentabili in modo diretto, bisogna cercare un valore che sia rappresentabile con lo shifter operand e che sia negativo, il -x indica che va bene qualsiasi numero che rappresenti un numero negativo: se il risultato alla fine sarà comunque negativo, allora c’è stato un errore)

CMP R0, #0

MOVLT PC, LP

CMP R1, #0

MOVLE PC, LP

MOV R2, #0

CMP R0, #0 //se il dividendo è 0 = caso particolare

MOVEQ PC, LP

//a questo punto servono altri due registri come variabili ausiliarie, però prima di poterli usare vanno salvati //nello stack (non si è fatto questo per i registri r0 e r1 perché quelli è convenzione che siano usati quando //si chiama questa operazione, infatti sono i parametri)

STM [SP], R3, R4, LP //si aggiunge LP perché quando si fa la Load Multipla per ripristinare i registri basta inserire il PC tra i registri da ripristinare e si effettua il ritorno così (evitando la MOV PC, LP)

MOV R3, #0

MOV R4, #1 //questi servono per implementare il ciclo di divisione

Primo ciclo: CMP R0, R1, lsl #1 (= R1 << 1) //tutte le istruzioni regop ammettono uno scorrimento sul secondo operando: il risultato dello shift viene utilizzato ma non sovrascrive R1

//a questo punto se il risultato è >= vuol dire che il dividendo è >= del doppio del divisore

ADDGE R3, R3, #1 (R3 = 0+1)

ADDGE R1, R1, R1 (R1 = R1+R1 = 2\*R1)

ADDGE R4, R4, R4 (R4 = 2\*R4)

BGE -24 (Branch a primo ciclo: ) //il ciclo è ripetuto finché R0 è maggiore di R1 x una certa potenza di 2.

//La potenza di 2 la troviamo in R4, mentre in R3 abbiamo il logaritmo in base 2 della potenza (ossia il numero di zeri aggiunti = numero di cicli)

Esempio: R0 = 27, R1 = 3: R1 viene moltiplicato per 2, R1 = 6, di nuovo per 2, R1= 12, poi di nuovo R1 = 24 e a questo punto poiché 48 > 27 in R1 resta 24 e R3 vale 3 e R4 vale 8. A questo punto il divisore è al massimo valore possibile. Si può entrare nel ciclo di sottrazioni.

Secondo ciclo: CMP R0, R1 //il controllo serve per quando R0 < R1

SUBGE R0, R0, R1 //in R0 ci sarà il resto

ADDGE R2, R2, R4

MOV R1, R1, asr #1 //(R1 >> 1)

MOV R4, R4, asr #1 //(R4 >> 1)

SUBS R3, #1 //la S indica che il risultato modifica I bit di condizione

BGE -32 //(-> Secondo ciclo: )

LDM [SP], R3, R4, PC

//nel nostro esempio precedente, R0 = 27, R1 = 24, R2 = 0, R3 = 3, R4 = 8:

Al primo ciclo è vera la condizione >= quindi R0 = R0 – R1 = 3. R2 = 0 + 8 = 8.

R1 diventa 12, R4 diventa 4 e R3 diventa 2.

R0 = 3 < 12 = R1, perciò non c’è sottrazione, ma R1 diventa 6, R4 diventa 2 e R3 diventa 1. R0 resta 3 e R2 resta 8.

3 < 6 dunque si procede riducendo R1 a 3, R3 a 0 (ultimo ciclo) e R4 a 1.

3 = 3 dunque R2 = R2 + R4 = 8 + 1 = 9, R0 = 3-3 = 0, R1 diventa 1 (ma non importa), R4 diventa 0 e R3 diventa -1, dunque la BGE non si attiva (-1 < 0).

Il risultato di 27/3 è dunque 9 (R2) con resto di 0 (R0). Il numero di cicli è stato 3+3 (poiché 3 è il logaritmo in base 2 del quoziente = differenza dei logaritmi in base 2 di dividendo e divisore, arrotondato per difetto).