Presentiamo la seconda parte di laboratorio. Questa settimana lavoriamo su vector.c. Il programma contiene l’implementazione di una libreria di funzione che simula i vector del c++ (seppur non object-oriented) che usa malloc, realloc ecc. è a tutti gli effetti un tipo di programmazione a livello di sistema.

La realloc normalmente permetterebbe di aggiungere elementi solo in fondo all’array. Quindi per aggiungere un elemento in posizione x sarà necessario scalare tutti gli elementi successivi a quello da aggiungere.

Un altro aspetto di cui si deve tenere conto è che la chiamata di realloc è abbastanza costosa, perché può richiedere una system call. Essa è più costosa in termini di tempi di esecuzione di una normale chiamata di funzione: bisogna minimizzare il numero di volte che bisogna usare la realloc, pertanto si segue il principio di allocare più memoria di quella necessaria in un dato momento (per esempio quando serve spazio per 1 elemento la dimensione deve essere di 2, all’inserimento del 3° verrà allocato spazio per un 4°, quando arriviamo a 17 elementi lo spazio allocato dovrà essere di 32. Lo stesso principio è da seguire per quanto riguarda la cancellazione: quando si arriva a metà della dimensione allocata si chiama la realloc e si liberano gli elementi vuoti.

Torniamo a parlare del nostro processore Amber 23.

Avevamo descritto le istruzioni di tipo interne ai registri (Regop): 4 bit per il predicato di esecuzione, 2 bit = 00 (regop), 1 bit I per l’indirizzamento dello shifter operand (immediato o no), 4 bit per il codice operativo, 1 bit S che indica se vanno cambiati i flag al termine dell’esecuzione, 4 bit per Rn, 4 bit per Rd, 12 bit di shifter operand, la cui decodifica cambia a seconda del valore del bit I. Le istruzioni regop si possono trovare nella tabella 4.  
La caratteristica di queste istruzioni è di poter essere completate in un solo ciclo di clock perché lavorano solo su registri interni.

L’istruzione di moltiplicazione (MULT) è anch’essa di tipo registro, ma ha una sua peculiarità: non è completamente supportata a livello hardware dalla ALU. Quindi la moltiplicazione viene implementata usando l’algoritmo di somma e scorrimento, pertanto richiede una certa quantità piuttosto elevata per essere completata. Questo è il prezzo da pagare (di 34 cicli di clock) per aver semplificato l’unità aritmetico logica. Anch’essa è un’istruzione condizionale (come tutte le altre), quindi nel caso la condizione risulti falsa l’istruzione non è eseguita e viene perso un solo ciclo di clock, altrimenti se ne usano 34.

Ovviamente nel momento in cui stiamo eseguendo un’istruzione come la MULT che prende più cicli di clock, è necessario bloccare la Pipeline. C’è quindi un controllo logico sulle componenti della pipeline che effettuano la condizione di blocco. Questo è chiamato lo Stallo (Stall) della Pipeline.

L’istruzione SWAP ha un suo formato di codifica diverso dagli alti: anch’essa ha il predicato, però la sequenza di bit successiva ad essa deve essere 00010. C’è poi un bit B, che può avere valore 0 o 1 che indica se deve essere indirizzato un byte o un’intera parola di memoria (B=1 un solo byte, B=0 l’intera parola), e due valori costanti a 00. I registri Rn e Rd assumono un significato e posizione analoga alle regop, con l’eccezione che vengono usati in maniera un po’ particolare, i 12 bit di shifter operand diventano infine un valore costante che rappresenta la configurazione 9 su 8 bit (000001001) e un ultimo registro Rm indicizzato sugli ultimi 4 bit.  
L’idea è di effettuare lo scambio di valori tra il contenuto di un registro e quello di due celle di memoria RAM (Rd viene infilato in RAM[Rm] e RAM[Rn] viene infilato in Rd) i cui indirizzi devono essere presenti nei registri Rn e Rm (Rd è il registro del processore di cui scambiare il valore). È possibile dare a Rn e Rm lo stesso registro, in questo modo la Swap serve per scambiare due valori in una volta sola. Questa istruzione richiede più di un ciclo di clock, ma è definita come istruzione non interrompibile (una interrupt non potrà interrompere a metà questa istruzione, lo scambio o viene fatto o no). Non è interrompibile anche perché a livello di bus i due cicli di lettura e di scrittura non possano essere separati tra di loro (se io ho tanti dispositivi che possono accedere alla RAM quando un dispositivo ha finito di usare il bus, questo può essere usato da qualcos’altro: in una situazione normale potrebbe capitare che dopo il primo accesso al bus, la cpu sia bloccata nel ri-accedere al bus perché lo ha iniziato a usare qualcos’altro, questo non accade durante la SWAP).  
Questa atomicità dell’istruzione serve per implementare delle primitive di sincronizzazione tra più processori diversi, quindi di realizzare un sistema multi-programmato in cui ci sono più processori che operano contemporaneamente e si coordinano durante l’esecuzione di un programma. Questo vale tra processori che condividono la stessa memoria.  
Lo SWAP effettua: una lettura da RAM e una scrittura in RAM, la lettura richiede 2 cicli di clock (uno per inviare l’indirizzo e uno per accedere al contenuto della RAM, questo funziona grazie alla cache di primo livello), la scrittura un altro ciclo di clock (con l’eccezione che a causa del protocollo di consistenza Write-Through se ci fosse un’altra lettura/scrittura subito dopo questa verrebbe ritardata).

Dopo l’istruzione SWAP, abbiamo l’istruzione TRANS, che sta per trasferimento di dati (Tra RAM e registri). La TRANS è di fatto equivalente alla swap, fa un’istruzione di load e un’altra di store. La Store salva il contenuto di un registro in una cella di memoria RAM (quindi bisogna definire correttamente l’indirizzo a cui inviarlo), la Load fa l’opposto: Copia da RAM al registro.   
La SWAP potrebbe essere vista come una successione non interrompibile di un’istruzione TRANS Load e poi TRANS Store.

La codifica comprende il predicato da 4 bit, dopodiché abbiamo due bit che assumono il valore 01 (quindi non sono regop): Abbiamo poi il bit 25 che fa ancora da bit I (ha la stessa funzione delle regop), due bit chiamati P e U e che modificano il comportamento dell’istruzione, abbiamo poi il bit B, che ha lo stesso significato che ha nella SWAP (di distinguere tra byte e parole di memoria). Ci sono poi i bit W e L che cambiano il comportamento dell’istruzione, poi i due registri Rn e Rd, entrambi specificati su 4 bit, e infine un campo offset da 12 bit. I dettagli di codifica dell’offset sono simili a quelli dello shifter operand.   
Il bit L differenzia la Load dalla Store (L=1 load, L=0 store), Rd è il registro destinazione (nel caso della load)/contenitore (nel caso della store), il registro Rn contiene l’indirizzo della cella di memoria RAM (l’indirizzamento è indicizzato, poiché si somma al contenuto del registro Rn il valore costante, salvo I=0 che causa l’apparizione di un altro registro, presente in offset).

Si ricorda che i registri del processore sono organizzati nel seguente modo: da R0 a R7 sono fissi, così come R15 che fa da PC, da R8 a R12 ne esistono due versioni, sono duplicati, mentre i registri R13 e R14 sono copiati 4 volte. Quale copia dei registri usare è scelta attraverso lo stato del processore identificato dagli ultimi due bit del PC (00 = user, 11 = supervisore, 10 = interrupt/IRQ, 01 = fast interrupt/FIRQ).

Questi modi di funzionamento del processore servono per virtualizzarlo. Il modo di funzionamento delle applicazioni è il modo 0. Il modo di funzionamento del sistema operativo è il modo 3 (supervisor). Gli altri due modi entrano in gioco quando si attiva un gestore delle interruzioni.

Quando il processore si trova in modo privilegiato (quindi è in modalità supervisore) può essere utile che il processore abbia accesso alla versione dei registri usati in modalità utente: è quindi possibile codificare una condizione di esecuzione per le load e per le store in cui i registri acceduti sono quelli visibili in modalità utente (quindi il sistema può modificare anche i registri della modalità utente, ma non vale il viceversa): questo è deciso da una delle combinazioni possibili dei vari bit PUW. Ciò è visibile nella tabella 10 (Pag. 16) secondo l’indirizzamento “accesso alla memoria non privilegiato”.

Altro tipo di istruzione sono le istruzioni di trasferimento multiplo: caratteristica peculiare, a livello utente, dei processori ARMS. Le MTRANS permettono operazioni di Load/Store di tanti registri contemporaneamente. Qual è l’utilità di queste istruzioni (volendo si può fare anche di tutti e 16 i registri contemporaneamente)? Tornano utili quando si effettua una chiamata di funzione che ha bisogno di più registri; normalmente si copiano i valori di tali registri nello Stack prima di sporcarli con quello che vuole fare la funzione e poi si ripristinano i valori originari alla fine. MTRANS permette di effettuare tutto questo salvataggio / ripristino con una sola istruzione (è codice abbreviato, che richiede un solo ciclo di clock per il decode, ma l’execute di esso prende tanti cicli di clock quanti ne servirebbero normalmente facendo una copia per volta). L’insieme da registri da salvare/ripristinare è indirizzato in maniera interessante: gli ultimi 16 bit vengono usati per indirizzare i registri soggetti dell’istruzione (Tale campo è chiamato “Register List”) e i 16 bit vengono usati come un Bit Vector (infatti ci sono 16 registri per la cpu). L’ordine con cui vengono salvati in memoria è: il registro 0 viene salvato nell’indirizzo più piccolo e il registro 15 viene salvato nell’indirizzo più grande (quindi l’ordine in cui si trovano in RAM è crescente secondo il numero del registro, lo stesso vale per l’operazione di load solo che il processo avviene al contrario).  
Esistono più modi per indicizzare la RAM con questa istruzione: indicizzato, autoincremento e autodecremento. Autoincremento/decremento vuol dire che il valore ottenuto come indirizzo della cella di memoria individuata deve diventare il nuovo valore del registro (se si specifica Rn come registro base di un indirizzamento indicizzato, alla fine dell’istruzione il valore del registro rimane così com’è, negli altri casi invece cambia). Dell’autoincremento e autodecremento ne abbiamo la versione Pre e la versione Post: Pre vuol dire “prima modifico il registro e poi uso il risultato per indirizzare la memoria”, Pos vuol dire “uso il valore del registro per indirizzare la memoria e poi lo modifico”. Nel caso di load/store multiple l’incremento/decremento dovrà essere ripetuta una volta per ogni registro trasferito. Il registro usato come base assume però fin da subito il valore finale (gli altri indirizzi sono salvati da qualche altra parte).

MTRANS avrà i soliti 4 bit dedicati al registro Rn e Rd sparisce, perché i suoi 4 bit vengono usati per il bitset che rappresenta l’insieme dei 16 registri.

Esiste un quarto modo di indirizzamento: quello non privilegiato. È un modo indicizzato ma che utilizza sempre i registri del processore in modo 0: può essere usato dal sistema operativo per mandare in esecuzione un’operazione utente. Questo è un modo molto efficacie per effettuare quello che secondo i principi di Denning è chiamato “cambio di contesto”.

Vediamo i due tipi di istruzione BRANCH e SOFTWARE\_INTERRUPT.

Le istruzioni di tipo BRANCH hanno i soliti 4 bit di condizione, poi 101, poi un bit L (che serve per trasformare la Branch in una Branch and Link), poi un offset di 24 bit. Se la condizione è vera, l’esecuzione dell’istuzione è: si interpreta l’offset come un numero con segno, che dice se si deve sommare o sottrarre, si prende il valore corrente del PC e gli si somma il valore dell’offset. Il risultato diventa quindi il nuovo Program Counter. Il valore di offset dovrà ovviamente essere un multiplo di 4 per essere un indirizzo valido.  
Il Bit L se è 1 dice: prima di modificare il PC salvane il valore nel registro LP (così da poter ritornare a quel valore dopo l’esecuzione delle istruzioni a PC+offset); la BRANCH AND LINK serve per chiamare una funzione (e poter riprendere l’esecuzione del programma precedente al termine della chiamata). Il salvataggio dei registri per tale chiamata di funzione può essere fatto con MTRANS (e lo store multiplo), quindi un’alternativa a salvare il PC nell’LP è di salvare anche il registro 15 in RAM attraverso una Store multipla e poi ripristinarlo alla fine.

Il SOFTWARE\_INTERRUPT sarà visto giovedì prossimo.