Vediamo il modo in cui i buffer possono essere gestiti a livello di sistema. Prendendo come riferimento il processore Amber, supponiamo di inviare una Fast Interrupt, e quindi di mandare una segnalazione attraverso il FIRQ. Osserviamo innanzitutto sul manuale che l’handler è piazzato in RAM all’indirizzo 28 (mentre per le interruzioni normali è il 24). Questo è l’ultimo elemento del vettore delle interruzioni (le celle di memoria successive sono libere: si può iniziare un programma sequenziale senza istruzione di salto). Questo è il primo vantaggio dell’utilizzo del FIRQ (infatti sono interruzioni veloci, ed evitare un salto evita uno svuotamento della Pipeline.

Le interruzioni veloci inoltre hanno più priorità rispetto alle interruzioni normali. Inoltre Firq ha il suo banco di registri (quelli dall’8 al 12, oltre ovviamente al 13 e 14).

L’idea è quella di poter leggere un carattere alla volta dal dispositivo di input e inserirlo in un buffer di sistema: bisogna quindi tenere conto il registro memory-mapped del dispositivo di input. Faremo riferimento a tale indirizzo con il termine tty\_in. Questo registro è utilizzato con i 7 bit meno significativi destinati a contenere il codice ascii del carattere inserito da tastiera, mentre il bit di segno (31) viene usato per indicare la presenza o meno di un carattere nel registro. Per convenzione interpretiamo il valore 0 con “c’è un carattere da leggere” e 1 con “non c’è nessun carattere. Questo bit serve infatti per il processore quando cerca il dispositivo che ha sollevato l’interruzione.

Se vengono premuti più tasti prima che il processore abbia risposto all’interruzione, i caratteri precedenti vengono persi e si salva soltanto l’ultimo. La lettura del processore toglie in automatico la richiesta di interruzione da parte del dispositivo (viene trattata come un aknowledgement da esso).  
Finchè non avviene la lettura, quindi, l’interruzione rimane presente; quindi il primo ciclo di clock è utilizzato per modificare i bit di maschera del registro di stato (B, F, sono presenti nel Program Counter nel processore Amber). Questi bit quando posti a 1 non ascoltano più altre interruzioni (quindi la prima cosa da fare dopo il fetch dell’interrupt handler è il disabilitare le interruzioni portando i bit a 1).

Normalmente si porterebbe a 1 solo il bit di maschera del tipo di interruzione che si sta rispondendo (quindi nel caso della Firq la F), ma se si vuole dare alle Fast Interrupt una priorità maggiore, allora si portano a 1 entrambi i bit.

Durante la fase di bootstrap, entrambi i bit di maschera sono a 1.

Nella Pipeline del processore Amber 23 va aggiunta una fase (prima del fetch dell’istruzione successiva) che controlla per la presenza o mento di interruzioni.  
Attenzione proprio al fatto che l’architettura sia di tipo pipeline: nel momento in cui si effettua il fetch dell’interrupt handler si sta ancora facendo il decode ed execute di due istruzioni precedenti, dunque il programma non viene interrotto subito, si concedono alla nostra applicazione i due cicli di clock necessari a terminare quelle due istruzioni.

Il buffer di sistema dovrà ospitare i caratteri premuti sulla tastiera e tenerli lì per tutto il tempo che serve alla nostra applicazione per chiedere di leggere tale buffer. Di conseguenza, tenendo conto di quale potrebbe essere la velocità dell’utente nel digitare i tasti, potremmo dover aspettare una quantità di tempo piuttosto lunga relativamente alla velocità di inserimento di nuovi caratteri: il buffer dovrà essere ragionevolmente abbastanza grande.

Per rappresentare il buffer vuoto si inserisce il carattere di terminazione, ‘\0’ all’inizio di esso. Ogni volta che si estrae un carattere dal dispositivo di input, esso viene inserito in coda al buffer, ma prima del carattere di terminazione, che quindi viene shiftato a destra ogni volta. Per esempio, dopo aver scritto “ciao” si otterrebbe:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| c | i | a | o | \0 |  |  |  |  |

Questo buffer permetterebbe di leggere fino a 4 caratteri. Il buffer di sistema reale ha in realtà dimensioni abbastanza grandi. Dobbiamo in ogni caso tener conto del fatto che col tempo si potrebbe comunque arrivare alla fine del buffer: dobbiamo escogitare un metodo che ci permetta di riconoscere sempre il problema di uscita dal buffer. Se a un certo punto ciò accadesse una scelta possibile sarebbe di eliminare il nuovo carattere e lasciare quelli vecchi, mentre la seconda alternativa sarebbe inserire il nuovo carattere perdendo uno di quelli precedenti. In questo secondo caso o si sceglie di perdere l’ultimo carattere o il primo (non per forza, ma avrebbe poco senso sostituirne uno in mezzo).

Per poter supportare in modo semplice la sostituzione del primo carattere con l’ultimo che è stato digitato, sarebbe ragionevole utilizzare una struttura dati del tipo buffer circolare: quando arriva al fondo e deve andare oltre, si ritorna all’inizio. In linguaggio macchina, per semplificarci la vita conviene scegliere come dimensione di buffer una potenza di 2: questo perché così ci viene facile un’operazione di modulo (infatti basta un AND con una maschera).   
Esempio: buffer circolare di 16 bytes.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

In questo caso qualsiasi sia l’indice che proviamo a usare, ci basta fare un AND col carattere “f” (che in binario è 15): così potremmo accedere a tutti i 16 bytes in maniera ordinata e ciclica (poiché l’and si comporta da modulo).

Esempio:

FIRQ 28: Ldr R8, [R10] //il valore del registro in memoria puntato da R10 viene caricato in R8, R10 punta al dispositivo di IO (si sceglie l’8 perché i registri dall’8 al 14 sono duplicati)

Cmp R8, #0 //si confronta R8 con la costante 0 per vedere se è >= 0 (ossia se il bit di segno del registro della tastiera è 0 => c’era veramente un carattere dentro)

//a questo punto ci sono due alternative: si può considerare più probabile che sia stato letto un carattere oppure che non si sia letto niente. Nel caso di un solo dispositivo che solleva le FIRQ, come il nostro, è ovvio che il primo caso è più probabile (anzi è certo), se ce ne sono tanti invece conviene procedere con la considerazione 2.  
R9 = Ind. Buffer. R11 = Indice % 16 //

Strdbge R8, [R9+R11] //si piazza il nuovo carattere all’interno del buffer e si sposta l’indirizzo al byte successivo (incrementando in modulo 16 R11). In questo caso l’ultimo carattere è sempre scritto nel buffer, sovrascrivendo eventualmente il carattere terminatore: per evitare ciò si fa sì che si possano scrivere sempre al massimo 15 caratteri

Addge R11, R11, #1

Andge R11, R11, #15

Movge R8, #0

Strdbge R8, [R9 + R11] //L’ultimo carattere diventa SEMPRE 0

End 56: Mov PC, LP //metti nel PC il contenuto del Link Pointer (in cui è stato salvato il valore precedente del PC). Serve a ripristinare l’esecuzione che era stata interrotta precedentemente.

Esempio con 8 caratteri:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Arriva la prima richiesta di interruzione, il carattere nel registro è “a”:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| a | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

R11 diventa 1. Arrivano b, c, d, e, f, g:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| a | B | c | d | e | f | g | 0 |

E R11 è 8. Il nostro buffer è pieno. Arriva il carattere h.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | B | c | d | e | f | g | H |

A questo punto possiamo solo sostituire.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| I | 0 | c | d | e | f | g | H |

L’unica che può svuotare questo buffer è l’applicazione che richiede la lettura.

Vediamo l’esempio alternativo.   
Bisogna innanzitutto chiedersi se c’è abbastanza spazio nel buffer per ospitare il nuovo carattere.

Ldr R8, [R10]

Cmp R8, #0

Strdbge R8, [R9+R11]

Addge R11, R11, #1

Andge R11, R11, #15

Movge R8, [R9+R11]

Mov PC, LP

Al posto del Movge, bisogna inserire la verifica che il valore a cui puntano [R9+R11] sia 0 (perché sono inizializzati a 0 quando il buffer è creato), se non lo è vuol dire che si sta sovrascrivendo e bisogna evitare di farlo.

Possiamo vedere una bozza dell’applicazione di lettura. Immaginiamo di avere il buffer

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| c | i | a | o | \0 |  |  |  |  |  |  |  |  |  |  |  |

La lettura avviene tramite system call (SWI) poiché il buffer di sistema ha bisogno del permesso dell’OS per essere acceduto. Il codice di tipo di richiesta di permesso è codificato sui soliti 24 bit: immaginiamo che la richiesta di read sia 1.

Sui sistemi POSIX la READ richiede un file descriptor, l’indirizzo del buffer applicativo e le sue dimensioni (ossia quanti byte leggere dal buffer di sistema). Ipotizziamo che la read che ha codice solo 1 indichi la read con file descriptor il buffer della tastiera. Il buffer e il dim possono essere passati nei registri da 0 a 7, perché non essendo duplicati rimangono uguali quando viene avviata una SWI. Per esempio in R0 può stare l’indirizzo del buffer, in R1 le sue dimensioni e in R2 viene salvato l’intero che READ dà come valore di ritorno.

Quindi se si vuole leggere ciao dal buffer basta impostare i registri 0 e 1 in modo che il primo abbia l’indirizzo del buffer applicativo e il secondo contenga il valore 5 (legge 5 caratteri, ciao+’\0’). L’handler della SWI parte dal registro 8, ma il registro dopo (il 12) non è libero, quindi è richiesta un’operazione di salto. Considerando che fino all’indirizzo 56 è occupato dall’handler della FIRQ, il primo indirizzo libero sarà il 60. Quindi il salto dovrà essere di 44 indirizzi (8+44+8 = 60, gli altri 8 sono per via della pipeline). La prima cosa da fare è il salvataggio dei registri che dobbiamo sovrascrivere (compresi R0, R1, R2 che ci servono per parametri), per poterli poi ripristinare alla fine del programma:

60: stm SP … (elenco dei registri da salvare, generalmente conviene inserirci anche il registro LP, così nella fase di ripristino basta sostituire a LP con PC, così il ritorno avviene in automatico quando si fa la Ldm).

Una cosa che poi bisogna fare è copiare al più quanti caratteri sono presenti nel buffer, quindi se il buffer di sistema ha meno caratteri di quelli richiesti si copiano solo quelli e si segnala ciò col valore di ritorno della read (infatti restituisce come intero il numero di caratteri che sono stati effettivamente copiati).

Ogni volta che un carattere viene letto, esso viene sostituito dal valore 0 (=‘\0’).