Un esempio di Trap per IJVM

Per capire meglio il concetto di Trap studiato nella sezione 5.6.4 del Tanenbaum, può essere utile vedere un esempio di Trap per IJVM e la sua relativa implementazione.

Se programmiamo direttamente in assembly IJVM (o IJVM puro) è chiaro che potremmo scrivere metodi sintatticamente corretti, ma che utilizzano in modo inconsistente lo Stack degli operandi. Per esempio:

```
.method (n)
BIPUSH 3
IADD
:
ecc...
.end-method
```

L'esecuzione di tale metodo porterebbe la IADD a distruggere la locazione di memoria contenente l'LV del chiamante, creando seri problemi.
Se non si riesce ad identificare il problema staticamente (a tempo di scrittura del codice), sarebbe quindi utile che a tempo di esecuzione il tentativo di prelevare un elemento da uno Stack Operandi vuoto facesse scattare una Trap: l'esecuzione del codice si bloccherebbe e verrebbe mandata in esecuzione una routine IJVM di gestione.
La routine di gestione potrebbe per esempio far comparire sullo schermo la scritta "Utilizzo scorretto dello Stack degli operandi da parte del codice IJVM".
Ovviamente la generazione della Trap deve essere implementata nei livelli sottostanti IJVM: a livello di Mic-1 oppure (parzialmente o completamente) a livello Hw.
Per il nostro esempio supponiamo che la routine di gestione sia del codice IJVM (non un vero e proprio metodo) memorizzato nella method area a partire dall'indirizzo 0x01 0x00.

Vediamo ora come realizzare la generazione della nostra Trap a livello di Mic-1.

Saranno i programmi Mic-1 che realizzano istruzioni che lavorano sullo Stack degli Operandi a controllare, prima di prelevare o eliminare un elemento dallo Stack Operandi, che quest'ultimo non sia vuoto.
Vediamo per esempio come dovrebbe essere implementata la POP.

```
pop1  MAR=LV; rd         //inizio lettura indirizzo contenente PC del chiamante
pop2                     //attendo completamento lettura
pop3  H=MDR+1            //pongo in H l'indirizzo della base dello Stack Operandi
pop4  Z=SP-H; if (Z) goto Trap1; else goto pop5  //controllo se lo Stack Operandi è vuoto
pop5  MAR=SP=SP-1;rd     //lo Stack operandi non è vuoto, procedo con la pop
pop6
pop7  TOS=MDR; goto Main1
Trap1  MAR=SP=SP+1
Trap2  MDR=PC;wr      //salvo sullo Stack il PC, nel caso mi servisse alla fine della gestione Trap
Trap3  PC=1<<8; fetch; goto Main1   //passo il controllo al codice del gestore Trap
```

La IADD invece dovrebbe essere realizzata nel seguente modo.

```
iadd1  MAR=LV; rd         //inizio lettura indirizzo PC del chiamante
iadd2                     //attendo completamento lettura
iadd3  H=MDR+1            //pongo in H l'indirizzo della base dello Stack Operandi
iadd4  H=SP-H; if (Z) goto gestTrap1; else goto iadd5  //controllo se lo Stack Operandi è vuoto
iadd5  Z=H-1; if (Z) goto gestTrap1; else goto iadd6  //controllo se lo Stack Operandi contiene un solo elem.
iadd6  MAR=SP=SP-1;rd     //lo Stack Operandi contiene almeno due elementi, procedo con la iadd
iadd7  H=TOS
iadd8  MDR=TOS=MDR+H; wr; goto Main1
```

È immediato notare come risulti molto pesante, in termini di efficienza, il controllo dello stato dello Stack Operandi. Si può migliorare l'efficienza con un aumento di costi. Si può pensare infatti ad aggiungere dell'Hw apposito per effettuare il controllo sullo stato dello Stack Operandi. Le possibilità sono molteplici.
Vediamone una.

Possiamo pensare di modificare l'Hw di Mic-1 aggiungendo un registro, BSP (Base-of-Stack Pointer). Tale registro servirà per contenere il puntatore alla base dello Stack Operandi del metodo corrente.

Ovviamente la presenza del registro aggiuntivo BSP comporterà varie altre modifiche. Per esempio la lunghezza delle istruzioni Mic-1 diventerà di 37 bit.
Occorrerà anche modificare il codice di INVOKEVIRTUAL e IRETURN, in modo che ad ogni chiamata di metodo venga salvato anche il BSP del chiamante, che verrà ripristinato alla fine del metodo da IRETURN. Ovviamente il valore da mettere in BSP quando chiamiamo un metodo sarà calcolato dal codice di INVOKEVIRTUAL. Questa ultima cosa si ottiene facilmente con una semplice modifica alla microistruzione invokevirtual19, che diventerà semplicemente

```
invokevirtual19  BSP=MAR=SP=SP+1
```

Costruiamo inoltre un circuito combinatorio (ES, ErrorStack) che restituisca 1 nel caso che si cerchi di leggere una parola su uno Stack Operandi vuoto e 0 altrimenti.
In pratica ES deve calcolare la funzione (MAR=SP) AND rd AND (SP=BSP). Infatti se MAR è uguale ad SP ed il bit rd è asserito vuol dire che la microistruzione corrente sta cercando di leggere la cima dello Stack. Il fatto che SP sia uguale a BSP indica che lo Stack Operandi è vuoto.
Inoltre in input ad MPC mettiamo l'output di un multiplexer M2 a due insiemi di ingressi, che avrà come segnale di controllo il valore prodotto da ES.
Il primo input di M2 sarà una costante: l'indirizzo del Control Store che contiene la microistuzione Trap1. Il secondo input di M2 sarà l'input che normalmente arriva in MPC.
Quindi, non appena eseguiremo una microistruzione che volesse prelevare un elemento da uno stack vuoto, si passerebbe immediatamente ad eseguire il microcodice Trap, che salva il PC e passa il controllo al codice IJVM ad indirizzo 0x01 0x00.

Ovviamente ci sono tante altre piccole cose di cui tener conto, ma ci siamo fatti un'idea di come realizzare, esclusivamente in software (Mic-1), oppure in Hw, una Trap per IJVM.