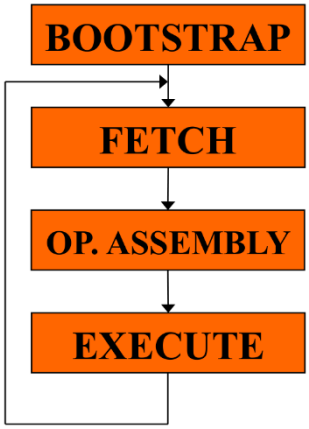
INTERRUZIONI

Abbiamo visto le problematiche delle pipeline, adesso entriamo nel dettaglio del meccanismo delle interruzioni: da un lato analizziamo il problema delle ***interruzioni all'interno di architetture basate su pipeline***, dall'altro valutiamo la possibilità di avere ***all'interno di un processore più pipeline in parallelo***. Questo diventa interessante quando le pipe sono specializzate per tipologie di istruzioni differenti, per esempio la pipe del Pentium 1 lavora su istruzioni intere con una ALU snella (riusciamo con 64 bit a fare le istruzioni intere velocemente) e su istruzioni floating point con una ALU dedicata per lavorare su informazioni in virgola mobile. *Abbiamo ALU differenti per un tipo di operazione o un'altra e possiamo cercare di mandare in parallelo più operazioni*. **Sebbene facciamo una ALU per floating point, parte del datapath è in comune con l'altra pipeline, cosa che può generare conflitto**, quindi nell'architettura a due pipeline (così come lo sarà per N pipeline in parallelo) **va gestita l'alimentazione delle due pipe affinché l'utilizzo contemporaneo delle risorse interne al processore non generi conflitto**. Dobbiamo approfondire le interruzioni precise e le architetture superscalari per le pipeline, ma partiamo da una ricapitolazione del meccanismo delle interruzioni. *Cosa sono gli interrupts e come possono le architetture gestirle?* Analizziamo una panoramica del 68k perché **nella progettazione dei driver questi fanno riferimento ad utilizzo esplicito delle interruzioni di un calcolatore, gestiamo I/O mediante interruzioni**. Dobbiamo capire cosa significa gestire le interruzioni e come farlo nel 68k per comunicare con le periferiche.

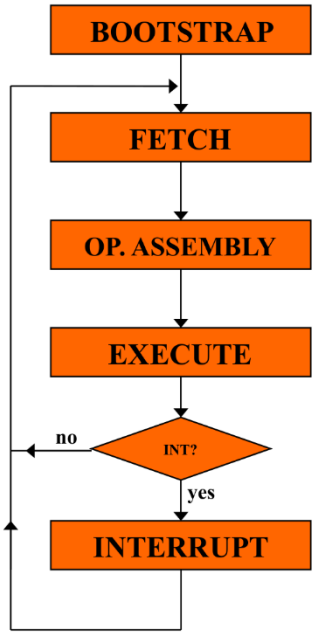
Da un lato vogliamo capire come gli interrupts si interfacciano ed integrano rispetto al ciclo fondamentale, cosa significa gestire l'interruzione e ritornare al programma principale, dall'altro come capire chi genera l'interruzione e come gestire più dispositivi che interrompono il processore contemporaneamente.

- **Aspetto 1: come si interfacciano ed integrano le interruzioni rispetto al ciclo fondamentale?**



Sappiamo che il ciclo fondamentale di VN prevede queste fasi, **dopo una fase iniziale in cui carichiamo il programma da mandare in esecuzione ed inizializzato il PC, abbiamo la sequenza continua di fetch-decode-execute**. Questa è una versione semplificata del ciclo, per esempio nel DLX sono 5 le fasi, ma il concetto non cambia. Un ciclo così fatto porterebbe a dei problemi, perché **una volta caricato il programma il processore non può più parlare con nessuno, non possiamo accettarlo perché il processore deve poter comunicare con l'esterno**. Per esempio, potrebbe accadere che un'applicazione prepotente si impadronisca di una risorsa senza rilasciarla, in un sistema multitasking non possiamo mai fare un qualcosa del genere, in questo modo **non potremmo nemmeno interrompere un'applicazione in esecuzione in un ciclo infinito**. Con questo ciclo semplificato, **il sistema operativo avrebbe un controllo limitato sul sistema**, una volta lanciato il programma ne perderebbe il controllo.

Questo ciclo, infatti, *viene modificato leggermente* **per permettere al processore di comunicare all'esterno in caso in cui si verifichino eventi eccezionali, asincroni con l'attuale programma in esecuzione sul sistema e che permettono di riportare il controllo al SO**. Ci sono situazioni in cui è fondamentale che ***il sistema operativo cambi il suo stato da modalità utente,*** *per gestire i programmi utente,* ***ad una modalità supervisore*** *in cui può fare operazioni differenti, tra cui gestire errori, comunicazione con periferiche o altri oggetti, dove la comunicazione si intende asincrona rispetto al programma in esecuzione*. Ad esempio, un errore molto noto è la schermata blu di Windows, questa si verifica quando c'è un errore di lettura dalla memoria, si genera un indirizzo di memoria pensando di prendere una pagina ma quella non è mai stata caricata, mandando tutto in stallo.



Per comunicare con l'esterno ci sono sia le interruzioni che le eccezioni. Possiamo **modificare il ciclo fondamentale del processore in modo che sia interrompibile**, **il punto in cui si interrompe il processore è una scelta architetturale ma anche di buon senso**, perché il problema reale dell'interruzione non è tanto l'interruzione in sé ma **il ripristino dello stato al termine dell'interruzione**. Se interrompiamo alla fine di un'istruzione, sappiamo che se carichiamo la prossima istruzione, manteniamo un minimo di informazioni per lo stato attuale, riusciamo a ripristinare, ma **se interrompiamo un'istruzione nel mezzo del suo ciclo potrebbe essere un problema**. Supponiamo che vogliamo interrompere subito dopo la fase di operand assembly, abbiamo acceduto alla memoria, caricato i registri interni con gli operandi, arriva un'interruzione e lasciamo il controllo a un altro programma che potrebbe utilizzare quei registri, sporcarli; ***questo comporterebbe una complessità maggiore per conservare lo stato del processore al fine di poterlo ripristinare dopo***. Questa situazione è ancora **più critica nelle pipeline**, perché se un'istruzione è finita le altre sono ancora a metà, quindi **cosa significa interrompere una pipeline**?

Finita una istruzione valutiamo se c'è o meno una richiesta di interruzione da parte di qualcuno (pending), che può essere dispositivo esterno, interno o errore interno. Se non c'è, procediamo con il ciclo fondamentale, **se c'è invece parte una routine particolare per gestirla, con il passaggio del processore dalla modalità utente a quella supervisore**.
La fase di interrupt viene eseguita quando da qualche parte si alza un segnale, che ***è un segnale fisico hardware che indica che c'è stata l'interruzione da parte di un dispositivo***. Questo evento è sintomatico del fatto che c'è un evento pendente, un qualcuno che richiede l'attenzione del processore e che deve essere servito appena possibile. **Gli eventi possono essere di diversa natura e l'interruzione è il servizio che provvede a gestire questi eventi**, infatti le interruzioni si chiamano ***ISR - Interrupt Service Routine***, che apparentemente **sembrano delle procedure, dei programmi o delle funzioni a cui possiamo saltare dal main per poter gestire il servizio ma ci sono differenze importanti** che dobbiamo mettere in evidenza. Durante la fase del processore non viene eseguito il programma principale, ma **viene eseguito un'interruzione speciale, prima di eseguirlo vanno fatte una serie di operazioni**.

Ci sono una serie di meccanismi hardware e software che preparano il processore a gestire l'interruzione: dobbiamo essere in grado dopo di poter continuare l'esecuzione del programma, **dobbiamo introdurre lo stato del processore e la necessità di ripristinare lo stato**. Per eseguire un programma software basterebbe rientrare nel ciclo fondamentale e muoversi tra le varie fasi fetch-decode-execute, nel caso di interruzioni questo non lo possiamo fare subito, lo possiamo fare quando lanciamo la ISR, prima dobbiamo prepararci per gestire l'interruzione.

La procedura di servizio può essere vista come una normale procedura, ma la differenza è che **il sottoprogramma esegue un'operazione in maniera sincrona con il programma in esecuzione**. Quando scriviamo un main sappiamo che ad un certo punto ci sarà una chiamata a sottoprogramma, quindi siamo noi a gestire il caricamento dei parametri sullo stack, il passaggio di parametri alla sottoprocedura, l'allocazione dello spazio delle variabili ed il ritorno dalla procedura; **sappiamo esattamente qual è lo stato del processore prima e dopo la chiamata alla procedura perché era previsto**. **Viceversa, un interrupt, anche se viene vista concettualmente come procedura a cui saltare, può avvenire in qualunque momento**. Se stiamo facendo la somma di elementi di un vettore, siamo al 30esimo elemento su 100 ed arriva l'interruzione, a questo punto dobbiamo salvare il fatto che siamo arrivati a 30, salvare che i primi 30 li abbiamo sommati e quindi dobbiamo salvare la variabile di conteggio, salvare l'indirizzo a cui tornare, dunque *una serie di operazioni che ci permettono di ripristinare lo stato al ritorno, problema che non abbiamo con una procedura perché sappiamo che in quel punto c'è la chiamata al sottoprogramma, adesso il salto invece può avvenire in qualunque momento*, per cui dobbiamo salvare lo stato e il ripristino dello stato opportunamente.

- **Aspetto 2: cosa causa un'interruzione e cosa significa gestire un'interruzione?**

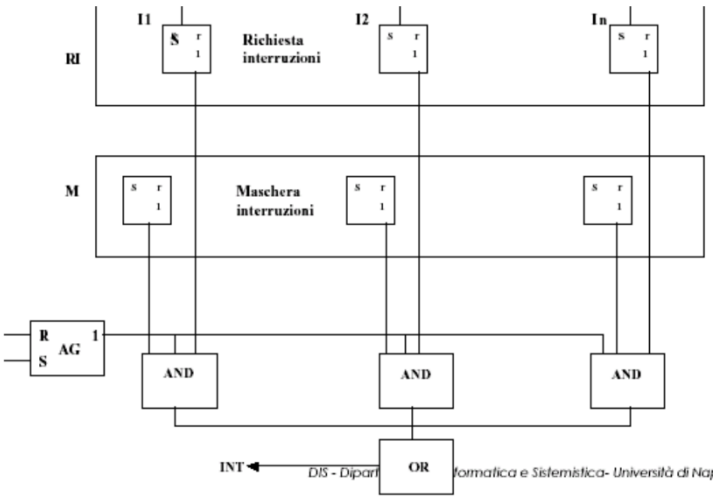
Le interruzioni possono essere di diversa natura. Si parla di esse spesso, ma per errore, quando pensiamo alle periferiche di I/O, come lo schermo, la stampante e così via, perché *questi dispositivi sono gestiti con driver particolari che comunicano mediante interruzion*i, ma possono essere tante le tipologie. Per esempio, potremmo avere un timer gestito dal sistema operativo per fare lo switch da un processo ad un altro oppure per evitare deadlock all'interno di determinati programmi, o ancora per passare il controllo ad un'altra applicazione, in questo caso quindi sono ***interruzioni periodiche gestite dal sistema operativo***. Ci possono essere interruzioni per errori, come un overflow che viene immediatamente segnalato e deve andare in esecuzione un'istruzione privilegiata (**traps o eccezioni**); **interruzioni per guasti**, un reset di un sistema che interrompe la normale esecuzione, **errore su pagina non trovata, errore su bus**; **interruzioni programmate (software interrupt)** leggermente diverse dalle altre che sono però in qualche prevedibili e quindi si può avere sincronicità con il programma correntemente in esecuzione.
Una causa di interruzione non fa partire la corrispondente routine ma fa partire una richiesta di interruzione. Soprattutto nel caso delle periferiche, si alza un segnale fisico, una linea sul bus dedicata alle interruzioni, che per esempio indica il pigiare del tasto invio della tastiera, tuttavia il fatto di aver premuto invio non significa che il buffer della tastiera viene direttamente svuotato ed inviato. **Affinché l'interruzione sia realmente attiva è necessario che il processore abbia abilitato precedentemente la periferica a parlare**; **la causa di interruzione prevede una richiesta di interruzione che bisogna verificare essere abilitata ed ha l'opportuna priorità**. Per gestire questi due aspetti (abilitazione e priorità) dobbiamo vedere il ***modello fondamentale del sistema delle interruzioni***. Quest'ultimo prevede **due registri speciali ed un FF**. Abbiamo il **registro di richiesta delle interruzioni Ri** che memorizza le richieste, il **registro di maschera M** che può abilitare singolarmente le richieste e poi **un FF di abilitazione generale del sistema Ag**.

Se c'è un nuovo evento viene alzato un bit nel registro delle richieste, se capita contemporaneamente che la maschera dell'interruzione è abilitata in corrispondenza di quel bit e l'abilitazione generale è abilitata in corrispondenza di quel bit, allora l'interruzione richiesta genererà un segnale ricevibile dal processore.

$$INT = AG \cdot \sum_i RI_i \cdot M_i$$

Algebricamente (Boole), possiamo realizzare questo sistema combinatorio come in figura: **prende in considerazione il bit i-esimo del registro richiesta in and con il bit i-esimo del registro maschera, fa la or di tutte le richieste; la or viene messa in and con l'abilitazione generale**.

L'***architettura del modello fondamentale delle interruzioni*** è questa in figura, in cui **nella parte alta c'è il registro di richiesta di interruzioni, che è collegato verso l'alto a tanti segnali**. *Se arriva la nuova richiesta viene memorizzata nel FF corrispondente al registro*. Poi **c'è il registro maschera di interruzioni, programmato di solito dal processore**, per esempio il primo lo abilitiamo perché può essere un servizio antincendio, le altre non le abilitiamo perché potrebbero riguardare un dispositivo come la stampante, di cui al momento non siamo interessati. Le linee vanno in AND con l'abilitazione generale. **La vera linea di INT viene generata se almeno una di queste AND introduce un 1**.
Supponendo che maschera e abilitazione valgono uno, se c'è una nuova richiesta, questa verrà tradotto in un segnale di interruzione che è quello che troviamo in basso.



Osservazione: la rete di priorità non è ancora visibile in questo schema, poiché viene implementata dai processori in maniera differente. *Questo è il modello fondamentale per generare un'interruzione* che viene implementata da due dispositivi. Il primo è il PIC, priority interrupt controller, la priorità verrà gestita dal processore lavorando sulla maschera. Per esempio, ***se abilitiamo la linea 1 e non le altre, di fatto stiamo dando una priorità alla linea 1***. *Il 68k, invece oltre a fare questo, ha anche un altro meccanismo fisico per cui le linee di interruzioni verso il processore sono 7 e possiamo assegnare una priorità* alle 7 linee lavorando sia sul livello della linea sia sul livello attuale dell'interruzione che stiamo servendo e che viene codificato nel registro di stato, sui 3 bit.

Il segnale di interruzione è presente se tutto il sistema è abilitato, la i-esima causa ha generato la richiesta e tale interruzione è abilitata dalla maschera. **Sia i flip flop di M sia l'abilitazione generale AG sono posizionati da opportune istruzioni in linguaggio macchina, quindi li possiamo programmare**, possiamo abilitare o disabilitare **il sistema di interruzioni**, possiamo abilitare o disabilitare assegnando un valore alla maschera **le interruzioni stesse** assegnando così una certa priorità. L'insieme di tutte le azioni elaborative svolte, da un lato dal processore nella fase di interrupt e dall'altro dalla ISR via software, viene detto ***processo di gestione delle interruzioni***.

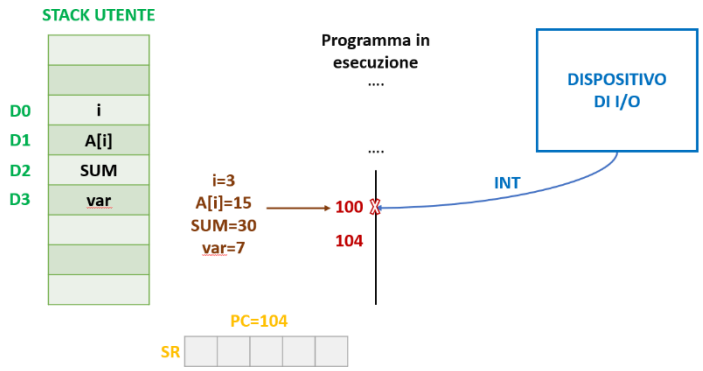
- **Aspetto 3: in cosa consiste il processo di gestione delle interruzioni?**

Si parte dall'esecuzione normale del servizio, del programma. Ad un certo punto **verrà generata una linea di interruzione, per poter servire questa linea va fatto il salvataggio del contesto che può essere fatto in hardware o in software, ma conviene fare in approccio ibrido**, cioè sia hardware e software. Per esempio, ***il 68k gestisce parte del salvataggio in hardware, quindi automaticamente alcuni registri interni del processore vengono salvati***. Non appena decidiamo di servire l'interrupt, si passa in modalità supervisore, **il 68K ha uno stack del supervisore dedicato e quindi nella fase di salvataggio del contesso verranno salvati sullo stack supervisore**, in maniera automatica e non da programma, ***il PC di ritorno, lo status register attuale***. **Questo, PC di ritorno e status register attuale, è il minimo set di informazioni che dobbiamo gestire**.
Bastano queste informazioni? Cosa definisce lo stato di un processore? **Serve anche il valore attuale dei registri.** Lo stato del processore è complesso, se decidiamo di gestire l'interruzione alla fine dell'esecuzione di un'istruzione probabilmente basterà **salvare i registri interni del processore**. Non possiamo buttare questi valori, **la ISR software andrà ad utilizzare gli stessi registri per cui dobbiamo preservarli salvandoli da qualche parte**. **In teoria, salvare lo stato del processore implicherebbe il salvataggio di tutti i registri interni**. **In realtà, si fa qualcosa di più efficiente**. *Veramente è necessario salvare tutti e 16 i registri?* **Non sapendo quand'è che arriverà l'interruzione e che registri abbiamo utilizzato fino a quel momento, possiamo salvare lo stato dei registri che la ISR andrà ad utilizzare**. Se dobbiamo chiamare l'interruzione che gestisce la stampante, che è un codice, una procedura, questa utilizzerà per esempio solo i registri D0,D1. Allora salviamo solo i registri che l'interruzione sprecherà. **Questo non viene fatto in maniera hardware prima di chiamare la ISR, perché non sappiamo la procedura quale ISR chiameremo**, sappiamo solo che c'è una linea di interruzione, dobbiamo ancora identificare il dispositivo, **sarà fatto via software alla ISR appena viene eseguita**. Se dobbiamo eseguire la ISR che deve stampare e sappiamo che essa scriverà su D0 e D1, diremo a chi ha scritto quel driver che prima di scrivere D0, D1 deve fare una copia sullo stack. **Prima di lavorare sulla procedura si salvano i registri su stack che pensano di sporcare, perché la ISR sa quali registri utilizzerà**. **Dopo aver salvato PC e SR, dobbiamo capire chi ha interrotto il processore e qual è l'indirizzo della ISR che è stata invocata, perché uno stesso dispositivo in realtà è collegato fisicamente al processore su più linee del processore**.
Per esempio, la tastiera, simulata con il 68K, avrà due interruzioni, la prima indica che l'utente ha premuto invio e vuole inviare caratteri e la seconda si verifica quando c'è il buffer full, perché la tastiera ha solo 256 caratteri, per cui se proviamo

a scrivere una frase più lunga, arrivato a 256, svuota il buffer, cosa che deve demandare al processore e quindi si genera un'interruzione. *Lo stesso dispositivo, tastiera, ha due interruzioni differenti e quindi due ISR differenti*.
Dobbiamo capire qual è il dispositivo che ci ha interrotto e qual è l'indirizzo della ISR cui saltare: *sono due aspetti critici che vengono gestiti in hardware ed in software in maniera differente in funzione dei processori in gioco*. **Capito l'indirizzo della ISR**, (è come quando facciamo la JMP subroutine) **il processore salta alla ISR e la prima cosa fatta all'interno è il salvataggio sullo stack via software di tutti quei registri che sporcheremo.** Non c'è nessun passaggio di parametri, salviamo sullo stack tutti i registri che utilizzeremo (**salvataggio del contesto software**). Eseguiamo l'interruzione vera e propria, poi, **prima di uscire dalla ISR ripristiniamo il contesto software**, per cui facciamo un pop dallo stack di tutti i registri salvati e li riscriviamo con il valore atteso dal programma, **e poi facciamo il ripristino del contesto hardware**, ovvero ripristiniamo anche il PC a cui tornare, perché non lo abbiamo nella procedura (non essendo questa una procedura).

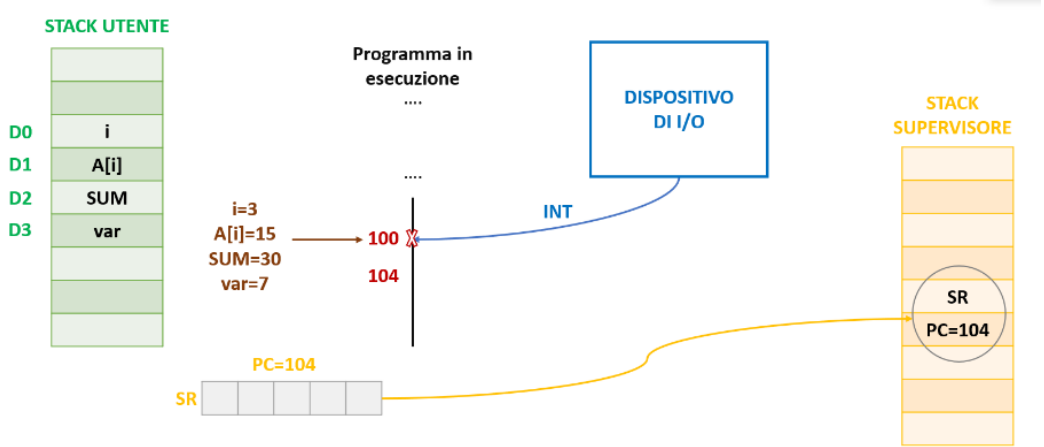
Osservazione: solo chi sviluppa la ISR sa quali registri va a sporcare. Se chi sviluppa il driver non considera questi registri, la ISR non funzionerà, quindi compriamo una stampante, installiamo i driver e questi non funzionano. La colpa è di chi produce la stampante ed i driver che sono prodotti dal produttore dalla stampante, il quale ha tutto l'interesse che i suoi driver funzionino. Lato processore va solo garantito che la ISR possa essere invocata e si possa ritornare. *Questo meccanismo però può essere una vulnerabilità del processore*, poiché nel driver si ha la responsabilità di ripristinare il valore dei registri, che sono i valori che il programma in esecuzione precedentemente si aspetta. Se i valori ripristinati non sono quelli corretti, il processore fallirà e quindi il driver non funziona.

Supponiamo che il processore stia facendo una somma, nel fare la somma sta usando questo banco di registri, *D0 che contiene i la variabile di conteggio, D1 che contiene il vettore A[i], D2 la somma parziale, D3 una variabile globale var*. Stiamo eseguendo il programma, siamo nel ciclo for e ci troviamo in un momento in cui i=3, A[i]=15, la somma parziale sum=30 e var=7. ***Siamo alla terza iterazione del ciclo, arriva un'interruzione da una periferica di I/O a questo punto*** (x rossa). Supponendo che sia 100 l'indirizzo del punto in cui avviene l'interruzione e 104 la successiva, *PC conteneva il valore della prossima istruzione da eseguire, quindi 104, lo status register avrà determinanti flag settati*, tra cui gli ultimi 5 relativi al risultato dell'operazione precedente.



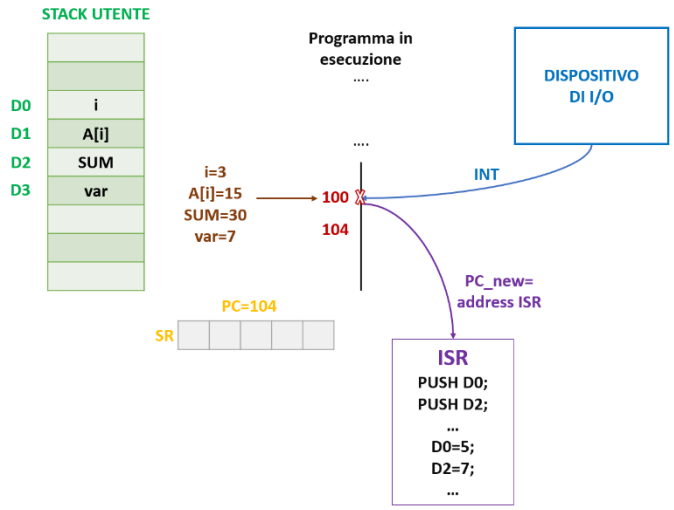
Arriva l'interruzione, eseguiamo le seguenti operazioni:

1. Passiamo da **modalità utente a modalità supervisore** mediante un bit presente nello status register.
2. **Salvataggio contesto hardware**. Lo stack modalità supervisore è separato da quello utente, facciamo il push di PC e SR che avranno determinati valori (figura sopra)



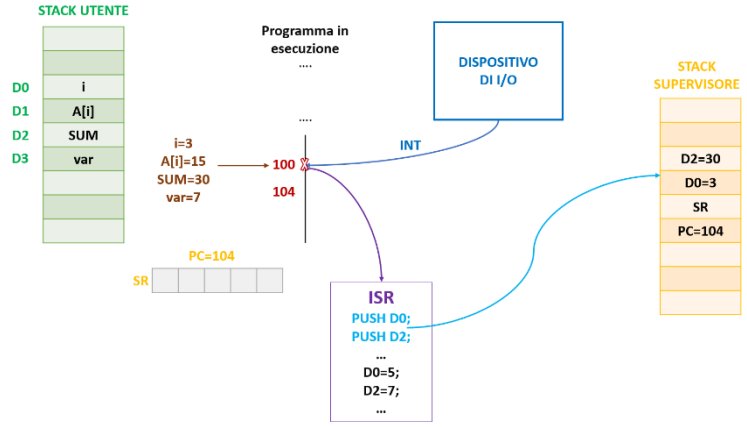
3. **Identifichiamo il dispositivo**.

4. **Identifichiamo l'indirizzo della ISR**. Ricaviamo address ISR, per cui *PC_NEW= address ISR*. Il PC salterà ad una procedura, alla ISR (figura sotto).

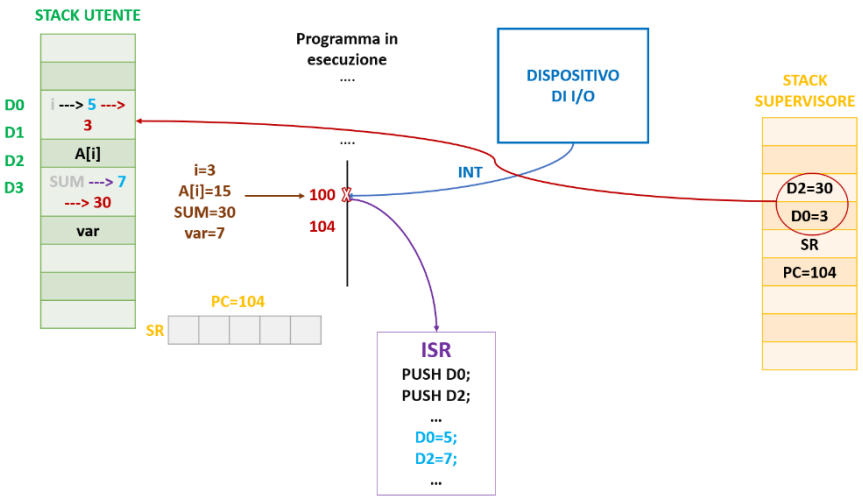


Supponiamo che *la procedura ISR scriverà D0=5 e D2=7, per cui rispetto ai 4 registri utilizzati nel programma precedente in esecuzione andrà a sovrascrivere solo il valore D0 ed il valore D2*. Gli altri non li tocca proprio. Allora quando siamo all'inizio della procedura, salviamo sullo stack D0 e D2.

5. **Esecuzione della ISR con salvataggio del contesto software e del programma secondo il ciclo fondamentale (fetch decode execute)**. All'inizio della ISR facciamo il salvataggio su stack mediante un push di D0=3 e D2=30. E' il salvataggio software perché è la procedura che lo fa. Successivamente procediamo con l'esecuzione delle istruzioni della ISR.



6. **Ripristinare il programma software**. Quando arriviamo qui, in D0 in cui troveremo 5 e in D2 troveremo 7 (*azzurro*). Eseguita tutta la ISR, *arrivati al termine facciamo il POP di D0, D2*. Cosa significa? **Ripristinare i valori vecchi, quindi prendiamo il valore di D2 su stack supervisore e lo scriviamo in D2, prendiamo il valore di D0 su stack supervisore e lo scriveremo in D0 per ripristinare i valori prima della chiamata ISR** (freccia rossa).

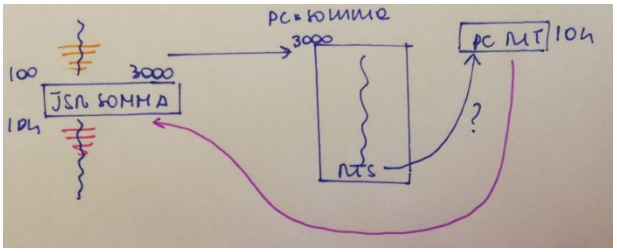


7. **Ripristino dell'hardware**. Significa ripristinare PC e SR. Se per esempio la prima istruzione di ISR ha indirizzo 5000 e termina all'indirizzo 6000, dobbiamo ripristinare PC a 104, valore che troviamo nello stack supervisore, e salvare SR precedente.

In qualunque punto ci troviamo nel programma, **con questo schema, salvataggio PC e SR in hardware, salvataggio in software dei i registri sporcati, possiamo fare il salto dentro alla sotto-procedura ed il ritorno ad essa**.

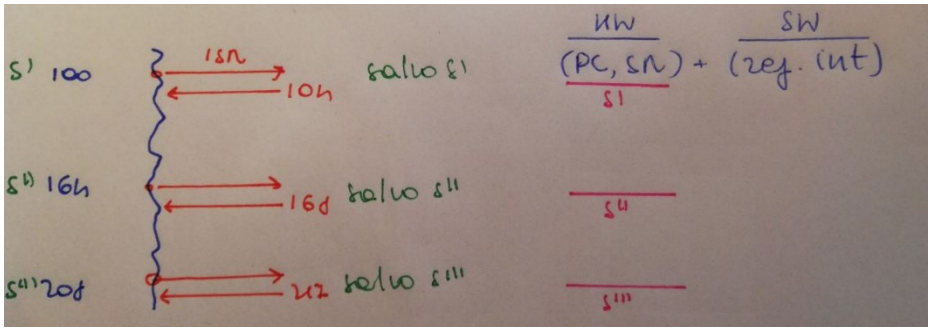
Qual è la differenza tra interruzioni e sottoprogrammi?

Abbiamo ad un certo punto JSR somma, quindi **il PC automaticamente viene inizializzato con l'indirizzo somma**. Eseguiamo



tutta la procedura. Il pc di ritorno viene salvato su stack. Terminata l'esecuzione della sotto-procedura, ricaviamo il PC OLD e saltiamo immediatamente all'indirizzo successivo all'istruzione JSR mediante RTS. **Quando abbiamo un sottoprogramma sappiamo esattamente quando verremo interrotti e sappiamo qual è esattamente l'indirizzo di somma**, che è un'etichetta, se per esempio è 3000 dobbiamo saltare all'indirizzo 3000. Con l'istruzione RTS (return from subroutine) preleviamo il vecchio PC per ritornare al punto fucsia in figura. Questo è sempre vero, sappiamo anche che se la procedura vuole dei parametri dobbiamo metterli sullo stack, prima di JSR e quindi prima di chiamare la somma, se la somma restituisce un risultato lo troveremo dopo la JSR. E' tutto **deterministico**.

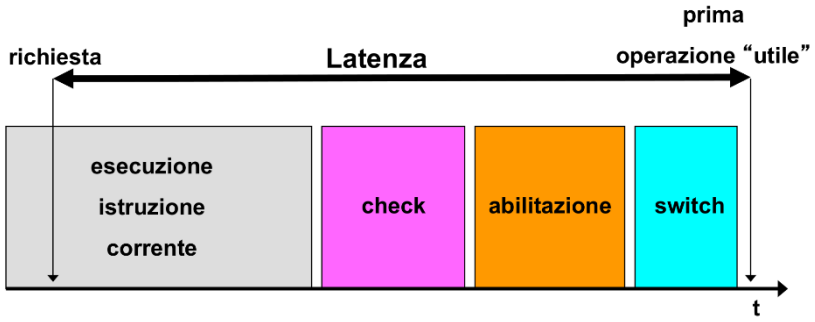
Rispetto ad un'esecuzione normale, immaginando tre punti 100, 164 e 208, **se avviene l'interruzione in uno dei 3 punti essa deve sempre prevedere salto all'indirizzo della ISR e ritorno dalla ISR** QUI (pallino cerchiato nei tre casi). Ogni volta il punto di ritorno è diverso se chiamiamo l'interruzione a 100 (è 104), lo chiamiamo a 164 (è 168) ed è diverso se lo chiamiamo a 208 (è 212). **In funzione della ISR e dell'indirizzo di ritorno cambia lo stato del processore**. Nel primo caso siamo nello stato S1, nel secondo in S2, nel terzo S3. **Indipendentemente da dove siamo, la ISR deve funzionare sempre allo stesso modo**, per cui nel primo caso salvo S1, nel secondo S2, nel terzo S3. **La procedura è sempre la stessa, salviamo prima in hardware il PC e lo ST e poi in software la ISR salva i registri interni**. **I registri interni non dipendono dallo stato, dipendono solo dalla ISR, mentre PC e SR dipendono dallo stato, quindi da S1, S2, S3 a seconda di dove viene chiamata, mentre nella procedura l'indirizzo di ritorno è sempre 104**. Apparentemente la chiamata ed il ritorno sembrano uguali, ma quello che cambia è che **l'indirizzo di ritorno viene calcolato in funzione dell'istante in cui viene chiamata l'interruzione, va salvato in modalità supervisore nello stack supervisore, non da programma con un'istruzione**, come RTS, ma avviene tutto in maniera completamente trasparente al programmatore. E' la JSR che scrive il PC di ritorno sullo stack, non abbiamo dall'altro lato una JMP ISR, ma viene chiamata dal processore perché viene generato un segnale fisico. Questa ISR genererà poi in hardware queste operazioni.



Osservazione: il salvataggio dei registri interni usati dalla subroutine (nel caso quindi della procedura, non dell'interruzione) dipende invece da una *tecnica di programmazione utilizzata per scambiare i parametri*. **Se l'unico modo per passare i parametri è su stack e che la procedura interna lavora su variabili interne, non possiamo sovrascrivere i registri utilizzati dal programma principale**. Se non possiamo fare questa ipotesi, conviene da programma mettere su stack i dati che andremo ad utilizzare. *E' legato alla modalità di passaggio di parametri ed utilizzo delle variabili locali*. Se lavoriamo solo su stack nella sotto-procedura, non dobbiamo modificare lo stato del chiamante, se invece lavoriamo su registri interni allora sì. **Le modalità di passaggio dei parametri sono tre, registri interni**. Nella procedura lavoriamo direttamente sui registri condivisi: se D0 contiene l'OP1 e D1 l'OP2, nella procedura sappiamo quale registro contiene cosa, per cui se assegniamo a sum=D0+D1, stiamo lavorando proprio sui registri del chiamante. *Se lo facciamo invece con la ISR e scriviamo D0+D1, nello stato S1, D0 e D1 significano una cosa, nello stato S2 un'altra*. **Nel caso della procedura è una tecnica di programmazione perché possiamo decidere di lavorare sui registri interni del processore perché ne consociamo il significato e quindi li elaboriamo; altrimenti esistono altre due modalità**, **passaggio di parametri con memoria condivisa oppure con stack**. Se lavoriamo con stack dobbiamo lavorare su variabili locali.

Un'interruzione potrebbe eseguire un'elaborazione completamente indipendente da quella del programma in esecuzione in quel momento, interrompendola. **La gestione delle interruzioni deve provvedere a mettere il programma interrotto nelle condizioni di poter continuare dopo l'elaborazione senza accorgersi del fatto che c'è stata l'interruzione**. A parte un ritardo, non dobbiamo accorgerci di niente in termini di eventuali malfunzionamenti. **Dobbiamo salvare prima lo stato e dopo ripristinarlo, ogni volta che questo verrà interrotto**. Tutte le informazioni che devono essere salvate comprendono PC, i flag dei codici di condizioni, se parliamo del M68K intendiamo lo SR, il contenuto di qualsiasi registro che sia usato genericamente dal programma e dalla routine di gestione dell'interruzione (abbiamo visto che questo secondo aspetto viene fatto in software dalla ISR). Le operazioni di salvataggio possono essere svolte in parte o completamente in hardware e in software. **Un'esigenza comune è salvare il numero minimo di informazioni per ripristinare lo stato, anche perché**

salvataggio significa una latenza maggiore nel trasferire i dati, per esempio da una memoria ad un'altra memoria, sullo stack. Addirittura, alcuni processori prevedevano anche una copia di registri dedicati al salvataggio per poter ripristinare velocemente lo stato. Data la frequenza con cui le interruzioni possono essere prodotte, questo rappresenterebbe un carico oneroso per il processore che deve essere tenuto minimo e quanto più efficiente possibile.
Il salvataggio dello stato incrementa il ritardo tra l'istante di ricezione della richiesta e il momento in cui viene eseguita realmente la routine, questo tempo è detto latenza dell'interruzione. Il salvataggio dello stato hardware consiste nell'avere una serie di registri che permettono le seguenti operazioni: salviamo il vecchio PC in un vecchio registro o nello stack come nel 68K, inizializziamo il PC con il primo indirizzo della ISR (indichiamo con start) e **AG=0, questo riguarda l'innesto delle interruzioni**. **Se stiamo servendo l'interruzione i ed arriva un'interruzione a priorità più elevata, possiamo servire quest'altra interruzione? Dipende**. *Se abbiamo implementato il salvataggio con hardware dedicato, non possiamo perché non abbiamo altro hardware per memorizzare questo nuovo PC*, per cui l'unica cosa che possiamo fare è porre AG=0, **l'abilitazione generale delle interruzioni viene disabilitata**. Quando stiamo servendo l'interruzione i dobbiamo disabilitare tutte le altre. *Se invece possiamo prevedere che interruzioni interrompano altre interruzioni con livello di priorità maggiore*, dobbiamo accuratamente **gestire questa fase di salvataggio**, lo facciamo in hardware ma non con hardware dedicato, per esempio lo facciamo su stack dove possiamo impilare le varie richieste di interruzioni. In software vale la stessa considerazione, nell'ISR salviamo tutti i registri necessari per la ripresa del programma interrotto. Una volta individuata la causa dell'interruzione serviamo l'interruzione stessa, ripristiniamo alla fine lo stato dei registri e riprendiamo il programma interrotto da dove era rimasto. **La latenza reale di un'interruzione è fornita da una serie di tempi. Il primo tempo è legato al completamento dell'istruzione che in quel momento stiamo eseguendo**. Abbiamo appena fatto la fetch di un'istruzione, dobbiamo prima finire quella, poi possiamo verificare se c'è stata una richiesta. Se c'è un'interruzione ed è abilitata allora possiamo prevedere lo switch vero e proprio, facciamo lo switch hardware, lo switch software e poi **dopo questo intervallo avviene la prima operazione utile della ISR vera e propria**.

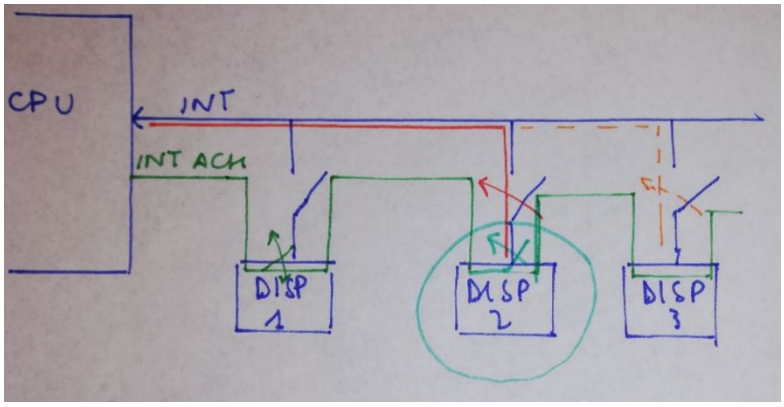


- **Aspetto 4: come si capisce chi ha generato l'interruzione e come si individua l'indirizzo del dispositivo?**

Abbiamo tralasciato due problemi: individuazione del dispositivo che fa l'interruzione e individuazione dell'indirizzo. Se andiamo nelle proprietà del sistema operativo, vediamo che in corrispondenza delle varie periferiche troveremo indicazione delle linee di interruzione relative ad un dispositivo e all'architettura che abbiamo sotto. ***Come identifichiamo i dispositivi?*** **Se ci sono più dispositivi il processore deve essere in grado di identificare il dispositivo specifico che ha generato l'interruzione, perché dispositivi differenti faranno attivare ISR differenti**. Può capitare lo scenario in cui dispositivi diversi condividono la stessa linea di interruzione oppure lo scenario in cui dispositivi diversi hanno linee differenti. Quando INT è alto dobbiamo identificare il dispositivo che ha interrotto. **Può anche capitare che più dispositivi interrompono contemporaneamente**.

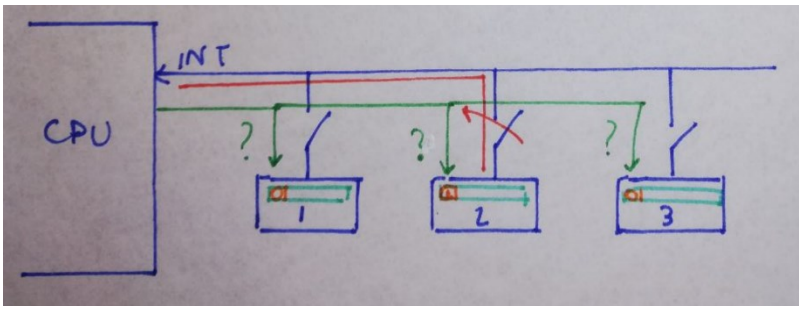
1. DAISY CHAIN

Consideriamo uno scenario di collegamento con vari dispositivi. Abbiamo il processore con **un'unica linea di interruzione che può essere condivisa tra più dispositivi di I/O**. **Possiamo immaginare un interruttore, a cui sono legati i dispositivi**, il dispositivo 1, il dispositivo 2 e il dispositivo 3. *Sono diversi, a seconda di chi interrompe dovremo fare cose completamente differenti*. **Il dispositivo 2 decide di interrompere, alza la linea di interruzione, chiude l'interruttore e il processore riceve la richiesta**. Il processore ha un'unica linea per fare la acknowledge della richiesta che si chiama **INTACK**. ***Questa linea è connessa in modalità daisy chain***. Questi dispositivi sono collegati come in rosso, c'è **una linea che attraversa i dispositivi**. Andiamo al primo dispositivo: se il primo dispositivo ha veramente interrotto, catturerà il segnale aprendo l'interruttore, se invece non è stato lui lascia la porta chiusa e il segnale passa al dispositivo successivo. **Il dispositivo 2 che ha interrotto aprirà la porta ed il segnale fisicamente non viene forwardato ai successivi, per cui l'ack viene dato al dispositivo che ha veramente interrotto**. **Daisy chain significa che il dispositivo che è fisicamente collegato più vicino al processore ha priorità maggiore**. Se anche il dispo sitivo 3 interrompe contemporaneamente al dispositivo 2? La priorità è del DISP2, quello più vicino al processore.



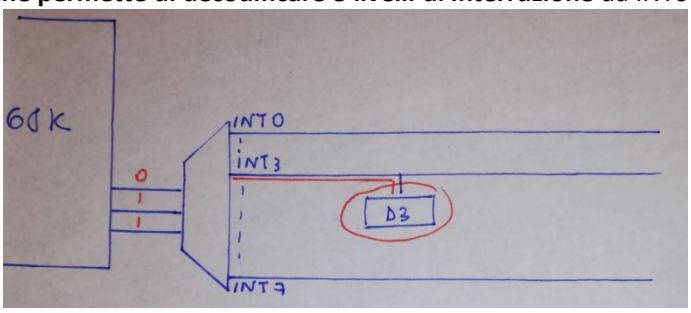
2. POLLING

Abbiamo sempre **una sola linea di interruzione, ma il processore interroga in modalità ibrida, chiamata polling**. Abbiamo tre dispositivi collegati, ma **ogni dispositivo ha al suo interno un registro di stato**. Quando il dispositivo genera un'interruzione, per esempio il 2, chiude ed alza il segnale, **metterà in un bit del registro di stato il valore 1 per indicare che ha interrotto lui, mentre gli altri, che non hanno interrotto, avranno 0**. **Il processore va ad interrogare questi registri, con un protocollo specifico**, chiedendo quale di questi dispositivi ha interrotto (freccia da processore verso i dispositivi). **Il primo che risponde avrà la ISR, sarà identificatbo**. Questa tecnica di interrogare il bit si chiama di polling, è poco efficiente perché prevede che il processore faccia operazioni specifiche con un protocollo, è più lenta.



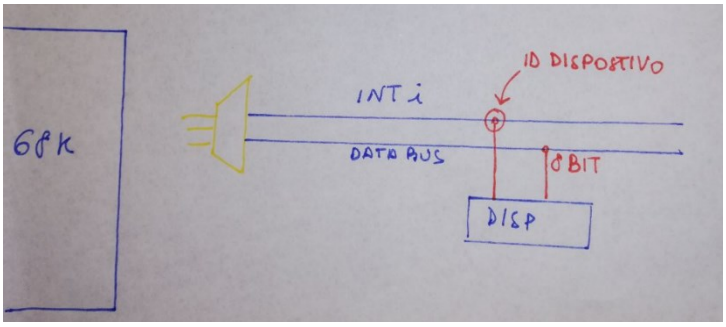
3. AUTOVETTORIZZATA

Il M68k ha **tre linee di interruzione connesse ad un decoder che permette di decodificare 8 livelli di interruzione** da INT0 ad INT7. Consideriamo la INT3 a cui è collegato un dispositivo. *Su ogni linea di INT possiamo averne più di uno, supponiamo di averne solo 1*. Se il dispositivo alza il segnale di interruzione, in maniera automatica verrà identificato perché stiamo sulla linea 3 ed il decoder fornisce il codice 011 **Riusciamo a riconoscere automaticamente il livello di interruzione ed il dispositivo collegato**. Questa nel M68K si chiama tecnica auto-vettorizzata, cioè **utilizziamo il livello della interruzione per identificare in maniera univoca la ISR e il dispositivo**.

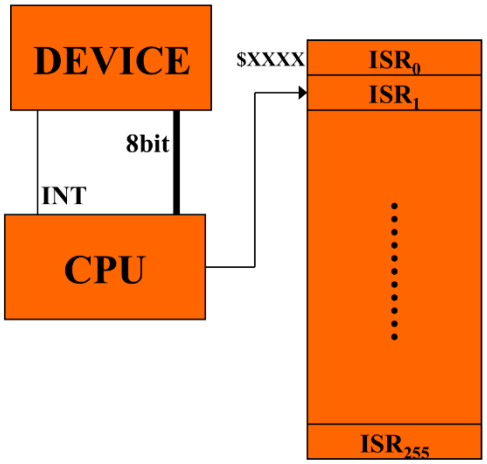


4. VETTORIZZATA

Un'altra tecnica del 68K è la vettorizzata. **Ci sono sempre le 7 linee di interruzione, però sulla linea i-esima c'è anche collegato il data bus**. **Il dispositivo, quando genera un'interruzione, da un lato alza la linea di interruzione e questa verrà forwardata al decoder che la decodificherà su un certo livello. Dall'altro manderà sul bus dati un vettore di 8 bit che servono per identificare la ISR**. *La linea INTi serve per identificare il dispositivo, il vettore su data bus viene utilizzato per identificare l'indirizzo della ISR*.



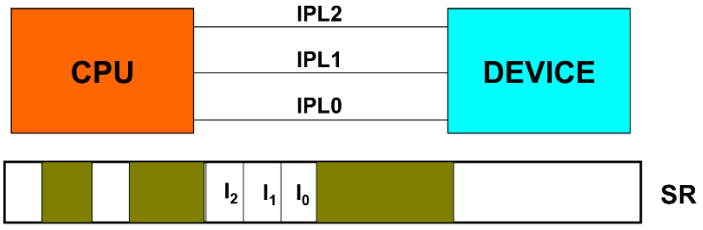
Il Motorola 68k ha un meccanismo speciale per gestire vettorizzato ed autoverizzato. Negli interrupt vettorizzati è il dispositivo stesso che fornisce un identificativo all'atto della richiesta che serve per calcolare l'indirizzo della routine che deve essere invocata. Nel 68k **abbiamo un'area di memoria ROM costituita da 256 locazioni consecutive che sono dette vettori di interrupt che contengono l'indirizzo della ISR**. In pratica, per esempio il dispositivo stampante mette sul data bus *il valore 100 che è l'indice del vettore in corrispondenza del quale avremo un indirizzo a 32 bit della prima istruzione della ISR a cui saltare*. Questo è il vettore degli indirizzi della ISR. In teoria ha 256 locazioni, in realtà **le prime 64 sono dedicate alle eccezioni, non ai dispositivi ma ad interruzioni interne, poi ci sono 7 che sono dedicate ad autovettori da 64 a 71, da 71 in poi sono tutti vettori**.



○ Aspetto 5: come si gestisce il nesting delle interruzioni?

Dobbiamo prevedere **un meccanismo delle priorità**. *Dobbiamo capire prima cosa l'ordine con cui verranno eseguite le interruzioni,* chiaramente **l'interruzione a maggior priorità verrà risolta per prima**. **Dobbiamo poi capire qual è il livello di priorità che l'interruzione sta servendo in quel momento**, per cui se arriva un'interruzione sulla linea 6 sappiamo che la linea 6 può ***interrompere e innestare tutte quelle di livello più basso ma non superiore***. Andiamo a vedere nello status register, se stiamo servendo un'interruzione di livello 7, la 6 sarà pending, la mascheriamo. Se invece stavamo servendo

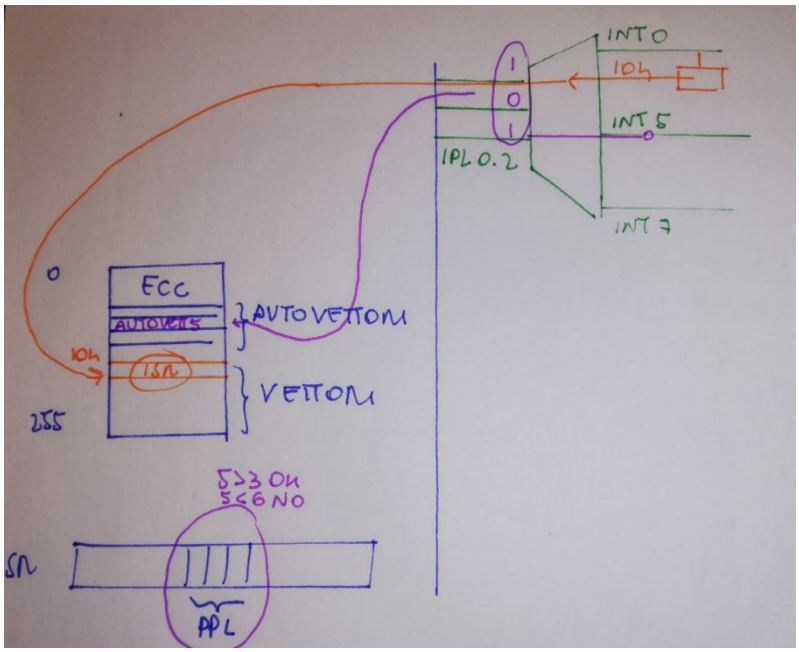
un'interruzione di livello 4, quest'ultima sarà interrotta e serviamo quella di livello 6. **Le tre linee a cui un device generico è collegato codificano il livello di priorità, si chiamano IPL – Interrupt request Priority Level**. **Nello SR, i tre bit codificano il PPL - Processor Priority Level (il livello massimo di interrupt mascherato)**. **Quando arriva l'interruzione confrontiamo l'IPL in ingresso con PPL in esecuzione, se l'IPL in ingresso è maggiore del PPL in esecuzione interrompiamo l'interruzione**, altrimenti la memorizziamo nella maschera, mettiamo ancora pending l'interruzione, finiamo quello che stavamo facendo.



Le linee a priorità 7 si dicono non mascherabili, non possiamo mai mascherare un'interruzione di livello 7 e questo è un problema se si verificano più interruzioni di livello 7 contemporaneamente. **Per gestire le differenti interruzioni**, le priorità, possiamo usare un meccanismo aggiuntivo che è costituito da un oggetto che si chiama **PIC, priority interrupt controller**, e permetterà di gestire le priorità, le chiamate innestate, il mascheramento, abilitazione e così via.

Architettura del 68k per gestire le interruzioni

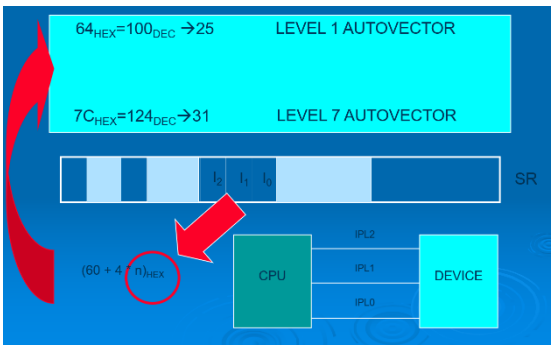
Il Motorola 68k ha lo **status register, che codifica all'interno il priority level** (PPL); ha al suo interno **il vettore delle eccezioni, fatto da 256 indirizzi**. La prima parte è dedicata alle eccezioni, poi ci sono 7 linee per gli auto-vettori, da quel punto in poi ci sono i vettori veri e propri. Poi ***abbiamo verso l'esterno le tre linee IPL, 0-1-2, collegate alle 7 linee di interruzione, con decoder oppure con PIC***. Dal decoder escono le 8 linee, da INT0 ad INT7. Nel caso di meccanismo di interruzione auto-vettorizzato (*viola*), se arriva un'interruzione dal livello 5, INT5, **questa identifica in maniera automatica l'auto-vettore 5,** per cui dal decoder INT5 viene codificata con 101 e ci porta all'auto-vettore 5 nel blocco. *Viene servita INT5?* Dipende dal valore che si trova nel PPL. Se c'è 3, allora 5>3 serviamo la INT5, se invece c'era 6, 5<6 quindi no. Nel caso del vettorizzato (*arancione*), **un qualunque dispositivo su qualunque linea manda un vettore sul bus dati**, per esempio 104. Questo 104 **viene utilizzato come indice della tabella per recuperare la ISR corrispondente**.



La differenza sostanziale è che *nell'autoverizzato non abbiamo bisogno di bus dati per mettere il vettore*. Nel caso vettorizzato, il dispositivo che interrompe si identifica fornendo un codice (**n, il vector number**) a partire dal quale possiamo risalire all'indirizzo di partenza della ISR in grado di servirlo. Nel caso del autovettorizzato il dispositivo che interrompe richiede una gestione automatica dell'interruzione e non fornisce alcun codice, in questo caso **24+IPL è il vector number**. **In ogni caso, l'indice del vettore è pari a 4*n**.

0	RESET (SSP)
1	RESET (PC)
16-23	UNASSIGNED, RESERVED
25	LEVEL 1 AUTOVECTOR
31	LEVEL 7 AUTOVECTOR
32-47	TRAP #0-15 INSTRUCTIONS
64-255	USER DEVICE INTERRUPTS

Nel dettaglio, per quanto riguarda il Motorola 68k, la tabella delle eccezioni prevede che le celle 0 a 23, quindi le prime 24, sono dedicate alle eccezioni, da 25 a 31 per auto-vettori, da 32 a 47 istruzioni di trap speciali, da 64 in poi quelle definite dall'utente. Dato che 64 in esadecimale rappresenta 100 in decimale, il primo livello è 25, mentre il livello 7 è 31. Allora se prendiamo ***(60+4*n) in esadecimale otteniamo l'indirizzo di partenza***.



Per evitare che lo stesso dispositivo interrompa si pone PPL=IPL.

Tutti zero		V7	V6	V5	V4	V3	V2	V1	V0	0	0
A23	A10	A9								A1	A0

Questo è il formato del vettore (a sinistra), **il processore traduce il numero del vettore in un indirizzo di 24 bit andando ad aggiungere due zeri nella parte meno significativa**.

Osservazione *(confronto vettorizzata ed autovettorizzata):* sono due protocolli differenti, si stabilisce un protocollo tra dispositivo e processore prima di mandare il dato sul bus. Se vogliamo usare modalità autovettorizzata basta collegare il device direttamente sulla linea, mentre se usiamo al vettorizzata dobbiamo dirlo al processore. Esiste un segnale VPA- VALID PERIFERAL ADDRESS: il device manda il segnale di interruzione, il processore dice ok, il device deve abilitare un VPA prima di mandare i dati sul bus. **Va stabilito il protocollo per indicare che sta utilizzando un vettore e non autovettore**. **Non sono modalità utilizzabili contemporaneamente**. *Con l'autovettorizzato siamo limitati a 7 ISR, quando il numero di periferiche all'interno del processore può essere anche molto più elevato, non ce la faremmo con 7 linee*. Possiamo fare un mix daisy chain ed autovettorizzato oppure non riusciamo con 7 linee, oppure si usa ***il PIC che permette di connettere in auto-verizzato in cascata fino a 64 dispositivi***.