POLLING

È simile al Busy Waiting ma a differenza di quest'ultimo il processore non aspetta più che il singolo device finisca di processare/produrre (a seconda se siamo in presenza di interf. di input o di output) i dati.

Si tratta di una "Verifica circolare" con lo scopo di trovare la prima periferica pronta ad interagire con il processore.

Questo metodo di interazione viene gestito completamente lato software perché l'architettura fisica può essere la stessa al Busy Waiting - stiamo semplicemente cambiando un metodo di approccio "lato software".

Il processore legge il Flip Flop di status del primo dispositivo ed effettua il controllo del bit meno significativo con un'operazione di "bit test" discussa sul pdf Busy Waiting.
Se CF = 1 significa che quel dispositivo ha finito di processare i dati che il processore gli aveva mandato (periferica di output) oppure che ha finito di produrre i dati che il processore deve andare a leggere (periferica di input).

```
.poll :
    movw $STATUS_DEV1, %dx
    inb $STATUS_DEV1, %al
    btb $0, %al
    jc .dev1
    inb $STATUS_DEV2, %al
    btb $0, %al
    jc .dev2
    # ...
    jmp .poll

. devX:
    # ...
    jmp .poll
```

Con la seconda istruzione leggo il valore del Flip flop di status e lo inserisco dentro AL.

Controllo se CF = 0 o CF = 1.

Se CF = 1 il dispositivo ha finito e posso interagire con lui e salto dentro la label .devX, viceversa se CF = 0 procedo ad interrogare il dispositivo successivo.

Il polling è **sub-ottimale**.
Il povero processore deve chiedere continuamente (testando il valore di status) se i dispositivi hanno finito di svolgere la propria attività.

Codice difficile da manutenere: cosa succede se si aggiunge un dispositivo?
Rischio di starvation: le ultime periferiche possono non essere servite mai.
Difficile gestire le priorità.