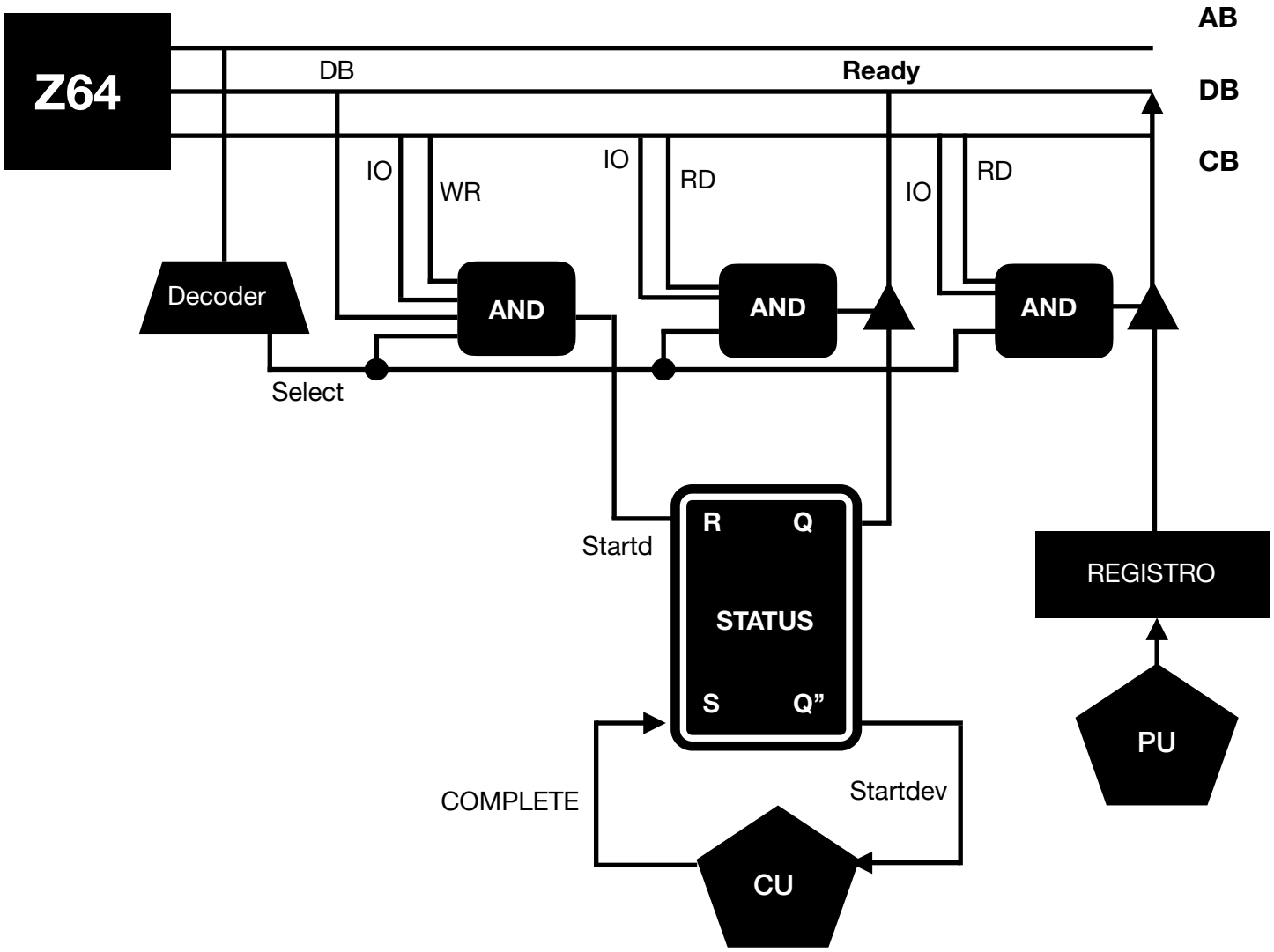
PROGETTO

Una periferica **TENSIONE** permette di acquisire informazioni sulla tensione elettrica di un contatore. La periferica produce dati di dimensione word come input per il processore z64. Scrivere il codice di una subroutine (secondo le convenzioni di chiamata System V ABI) che accetti come parametri il numero di dati (una word) da leggere dalla periferica **TENSIONE** e l'indirizzo di memoria da cui il processore z64 dovrà incominciare a scrivere i dati così acquisiti dalla periferica. Scrivere inoltre il programma che invoca la funzione chiedendo di acquisire 100 word dalla periferica **TENSIONE** e di memorizzarli in un vettore posto a partire dall'indirizzo 0x1200. **La dimensione massima del vettore è 400 word**.

Svolgimento



La periferica TENSIONE è un'interfaccia di input poiché scrive dati all'interno del processore Z64 - in modo particolare produce dati sulla tensione elettrica di un contatore. All'interno della periferica è altresì presente un registro che bufferizza queste informazioni che successivamente dovranno essere lette.
L'unità di processamento deve produrre dei dati per il processore e quest'ultimo deve essere notificato che i dati sono stati prodotti per poter essere acquisiti: questo viene risolto semplicemente leggendo il valore di Q nel flip flop di status.
Se il valore di Q = 0 i dati non sono ancora stati prodotti, viceversa, se il valore di Q = 1 i dati sono stati efficacemente prodotti dalla periferica e sono pronti per essere acquisiti.
Dal testo del problema questi dati hanno una dimensione di 16 bit - 2 byte - word.
L'interfaccia quindi è bi-settorizzata in due zone:

1. La zona relativa all'interfacciamento Busy Waiting.
2. La zona relativa alla produzione e la conseguente consegna dei dati: interfaccia di input.

La prima operazione da svolgere è quella che consente l'avvio del dispositivo.
Possiamo decidere di utilizzare l'approccio di tipo esplicito in termini di sintassi durante un'operazione di scrittura sul dispositivo.

```
movb $1,  %al
outb %al,  $TENSIONE_STATUS
```

$TENSIONE STATUS è il nome dell'interfaccia del flip flop status - ci serve per avviare il dispositivo in modo tale che il processore stabilisca un'interazione con esso.
Quindi sostanzialmente ho chiesto a tensione di produrre un nuovo dato.

```
.equ $TENSIONE_STATUS, 0x0000
```

Con l'uso di questa istruzione TENSIONE_STATUS parte dall'indirizzo **0x0000**.
Ora bisogna implementare la parte relativa al loop del Busy Waiting - il processore deve leggere continuamente il valore del flip flop di status per sapere se il dato da trasferire sulla CPU è pronto.

```
.bw1:
    inb $TENSIONE_STATUS, %al
    btb $0, %al
    jnc .bw1
```

Grazie all'ausilio di questo ciclo il processore abilita il segnale di I/O e RD per leggere il valore del flip flop di status.
Tale valore verrà inserito all'interno del registro %RAX e con un'operazione di bit-test verrà comparato con la costante zero.
Se CF = 0 torna ad eseguire il loop.
Se CF = 1 significa che i dati sono pronti.
ATTENZIONE: Quando il loop è finito il dato sarà pronto e bisognerà procedere alla sua acquisizione - Recuperiamo il dato.

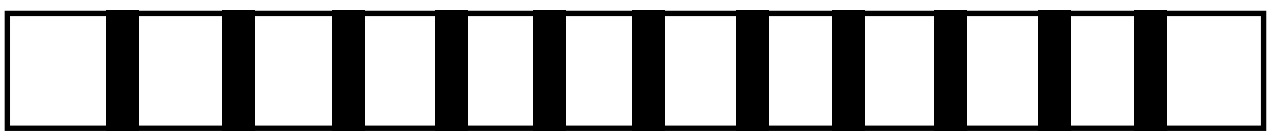
```
.acquisizione:
    inw $TENSIONE_REG, %ax
    movw %ax, (%rsi, %rbx, 2)
    addq $1, %rbx
    cmpq %rbx, %rdi
    jnz .acquisizione
```

Piccola nota: devo leggere dal registro una word per volta - motivo per il quale l'istruzione di lettura "IN" possiede il suffisso "W".
Il registro di destinazione è proprio %ax poiché abbiamo letto 16 bit.

Questo valore viene spostato all'interno di un vettore in modo particolare al seguente indirizzo:

Indirizzo iniziale del vettore + i*2

Dove ciascuna cella è 2 byte.



I è il conteggio presente all'interno del registro RBX, ci permette di spostarci di una cella alla volta.
I = 1, … , 100.
In RSI è presente l'indirizzo iniziale del vettore!
Il vettore è così dichiarato:

```
vettore: .fill 400, 2
```

Riserva uno regione di memoria composta da 400 celle di dimensione 16 bit (2 byte).

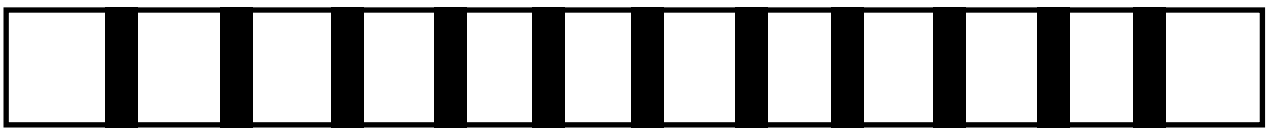
L'operatore XOR deve azzerare all'inizio l'indice del vettore. Abbiamo usato il registro RBX come indice:

```
xorq %rbx, %rbx
```

È ancora importante sottolineare l'importanza del "Location Counter".
Il vettore deve partire dall'indirizzo in memoria 0x1200, perciò è possibile modificare esplicitamente il Location Counter per indicare che sotto tale istruzione il vettore parte proprio da quell'indirizzo.

```
. = 0x1200

vettore: .fill 400, 2
```



0x1200 0x1201

4608 4609 ■ ■ ■

Di seguito il codice completo.

```
.org 0x800
.data
    .equ $TENSIONE_STATUS, 0x0000
    .equ $TENSIONE_REG, 0x0001
    . = 0x1200
    vettore: .fill 400,2
.text
main:
    movq $vettore, %rsi #caller-save
    movq $100, %rdi #caller-save
    call input
    hlt
Input:
    push %rbx
    xorq %rbx, %rbx
    .acquisizione:
        outb %al, $TENSIONE_STATUS
    .bw1:
        inb $TENSIONE_STATUS, %al
        btb $0, %al
        jnc .bw1
        inw $TENSIONE_REG, %ax
        movw %ax, (%rsi, %rbx, 2)
        addq $1, %rbx
```

```
cmpq %rbx, %rdi
jnz .acquisizione
pop %rbx
ret
```

#RBP = registro il cui contenuto deve essere preservato dalla subroutine poiché è un “registro di salvataggio del chiamato” e quindi per utilizzarne il contenuto bisogna inserirlo nello stack tramite push e alla fine ripristinarlo tramite pop.

Piccola raccomandazione: non ha senso eseguire il ciclo di acquisizione dopo aver avviato **solo una volta** il dispositivo, questo perché, in questo modo, *si legge sempre lo stesso valore prodotto dal dispositivo*.
Per far produrre un altro valore, **bisogna riavviare il dispositivo** (con una out) e aspettare che abbia prodotto il nuovo dato (con un ciclo di busy waiting).

Soltanto dopo si può accedere in input al registro di interfaccia.

Per il reset di STATUS, le linee del DB sono don't care conditions, quindi qualsiasi sia il valore in %al, verrà sempre e comunque resettato il flip flop