Scambio dati con il master

Dobbiamo realizzare contemporaneamente:

- Routine di scambio dati
  - Emissione interruzione al master
  - Gestione scambio dati sincronizzato tra slave e master
- Interfaccia Master-Slave
  - Interfaccia per gestire l'interruzione lanciata dallo slave
  - Interfaccia per lo scambio dati

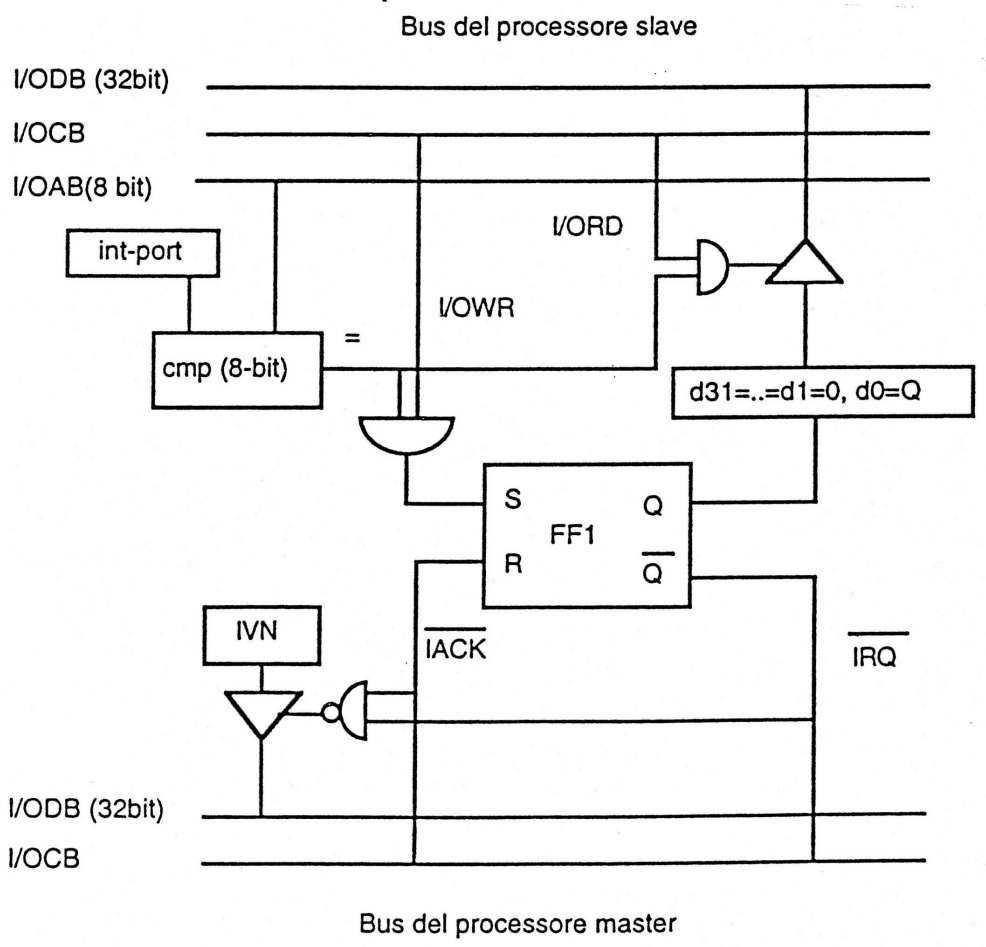
Circuito per la gestione dell'interruzione lanciata dallo slave

Passi:

1. OUT slave → setta FF1 → Q → IRQ
2. IACK master→ master rileva l'IVN e resetta FF1
3. Slave rileva stato FF1 in busy-waiting tramite IN sull'indirizzo dell'interfaccia, prima di iniziare invio dati

Sezione routine scambio-dati per lo slave:

```
        outl int-port, r3
loop:   inl int-port,r3
        cmpbl #0,r3
```



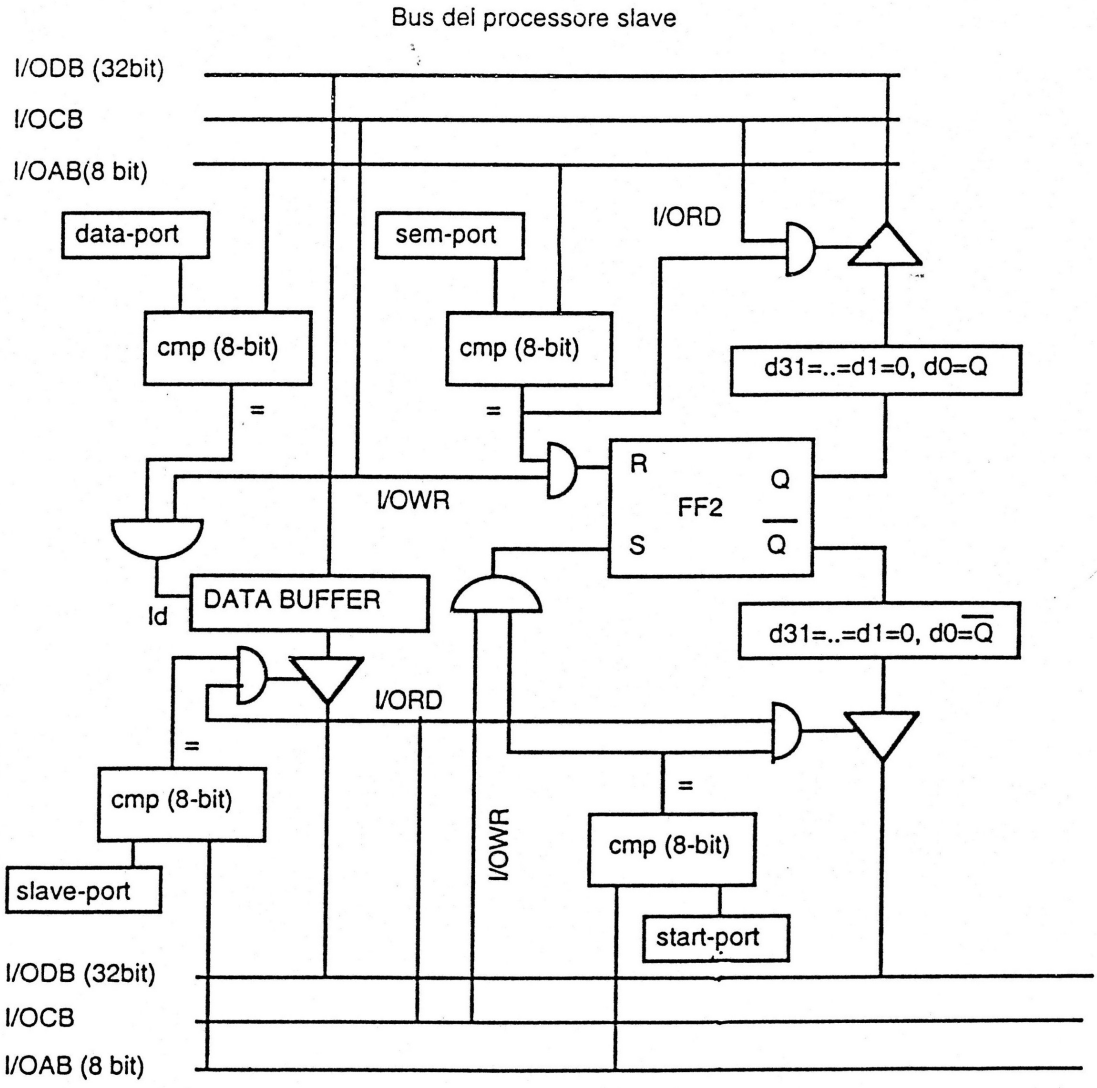
Circuito per lo scambio dati
e relative sezioni di routine di gestione

Processore Slave

```
ciclo-tx:    OUTl data-port,Rx
             OUTl sem-port,Rx
reset-FF2:INl sem-port,Rx
             CMPl #1,Rx
             JNZ reset-FF2
             JMP ciclo-tx
```

Processore Master

```
ciclo-rx:  INl start-port,R3
           CMPl #1,R3
           JNZ ciclo-rx
           INl slave-port,R3
           OUTl start-port,R3
           JMP ciclo-rx
```



Interfaccia Master-Slave per gestione superamento valore soglia

- unione dei due circuiti precedenti
- Software composto dalla routine (per lo slave) e driver (per il master) seguenti

SLAVE
Routine scambio-dati

```
        clri
        outl sem-port,R3
        outl int-port,R3
loop:   inl int-port,R3
        cmpl #0,R3
        jnz loop
        xorl R0,R0
        movb #10h,r1
ciclo-tx: inl sem-port,R3
        cmpl #1,R3
        jnz ciclo-tx
        movl AD2(R0),R3
        outl data-port,R3
        outl sem-port,R3
        addl #4,R0
        subb #1,r1
        jz esci
        jmp ciclo-tx
esci:   seti
        ret
```

MASTER
Device Driver

```
        push R0
        push R3
        push R1
        xorl R0,R0
        movb #10h,R1
        outl start-port,R3
ciclo-rx: inl start-port,R3
        cmpl #1,R3
        jnz ciclo-rx
        inl slave-port,r3
        outl start-port,R3
        movl R3,ADM(R0)
        addl #4,R0
        subb #1,r1
        jz esci
        jmp ciclo-rx
esci:   pop R1
        pop R3
        pop R0
        reti
```

Interfaccia Master-Slave per gestione interruzione da parte del master per richiesta periodica (ogni 10 s) del set di dati

aggiungiamo:

- un circuito per la gestione delle interruzioni dal master allo slave (speculare a quello proposto)
- Una routine di scambio dati (per il master) ed un driver (per lo slave)

Processore SLAVE

Device Driver

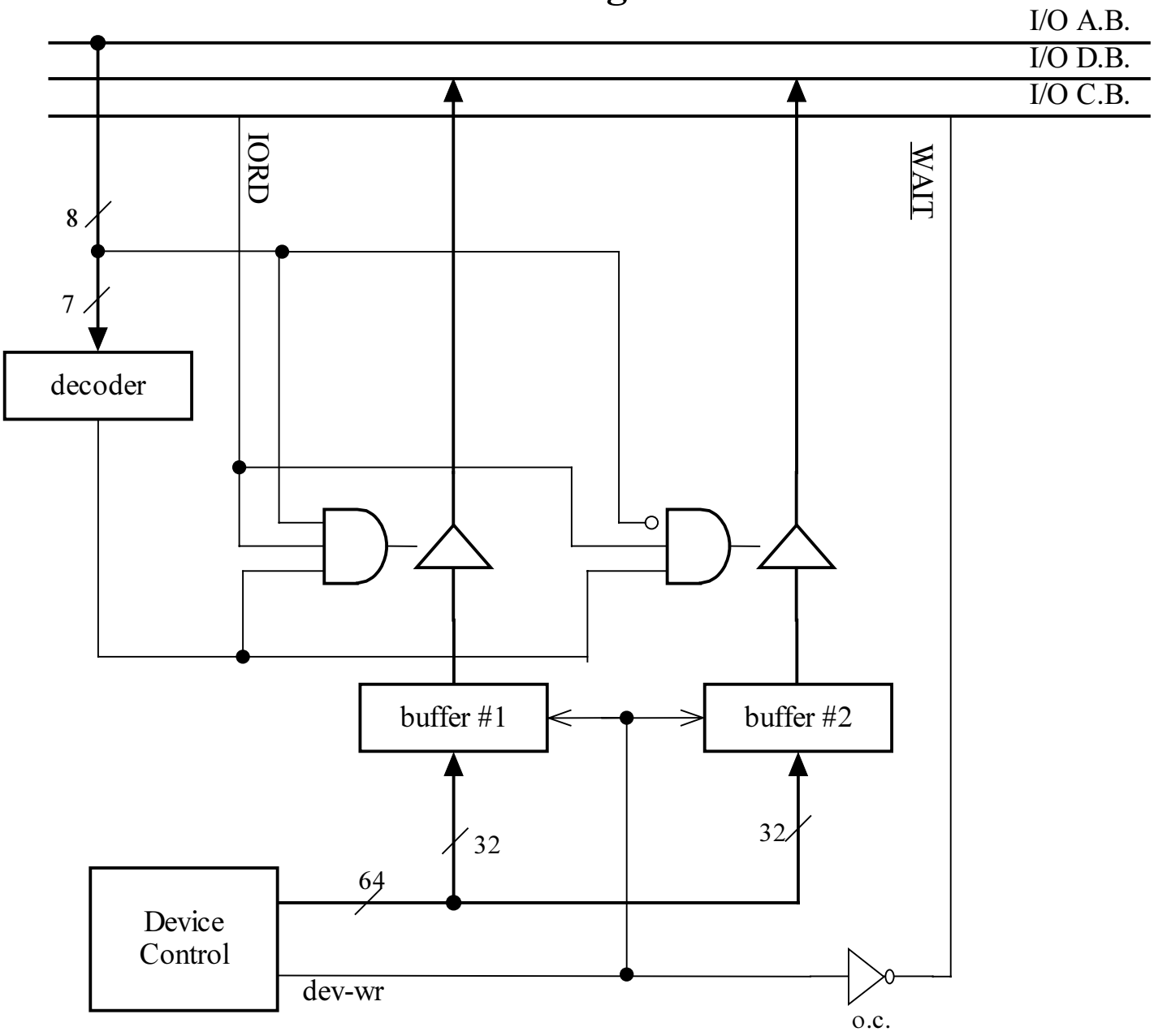
```
            push R0
            push R3
            push R1
            xorl R0,R0
            movb #10h,R1
ciclo-tx:   inl sem-port,R3
            cmpl #1,R3
            jnz ciclo-tx
            movl AD3(R0),R3
            outl data-port,R3
            outl sem-port,R3
            addl #4,R0
            subb #1,R1
            jz esci
            jmp ciclo-tx
esci:       pop R1
            pop R3
            pop R0
            reti
```

Processore MASTER

Routine: scambio-dati

```
            clri
            outl int-port,R3
loop:       inl int-port,R3
            cmpl #0,R3
            jnz loop
            xorl R0,R0
            movb #10h,R1
            outl start-port,R3
ciclo-rx:   inl start-port,R3
            cmpl #1,R3
            jnz ciclo-rx
            inl slave-port,R3
            outl start-port,R3
            movl R3,ADM(R0)
            addl #4,R0
            subb #1,R1
            jz esci
            jmp ciclo-rx
esci:       seti
            ret
```

Interfaccia PD32 Slave – periferica interazione con segnale di WAIT



Routine di accesso ai dati

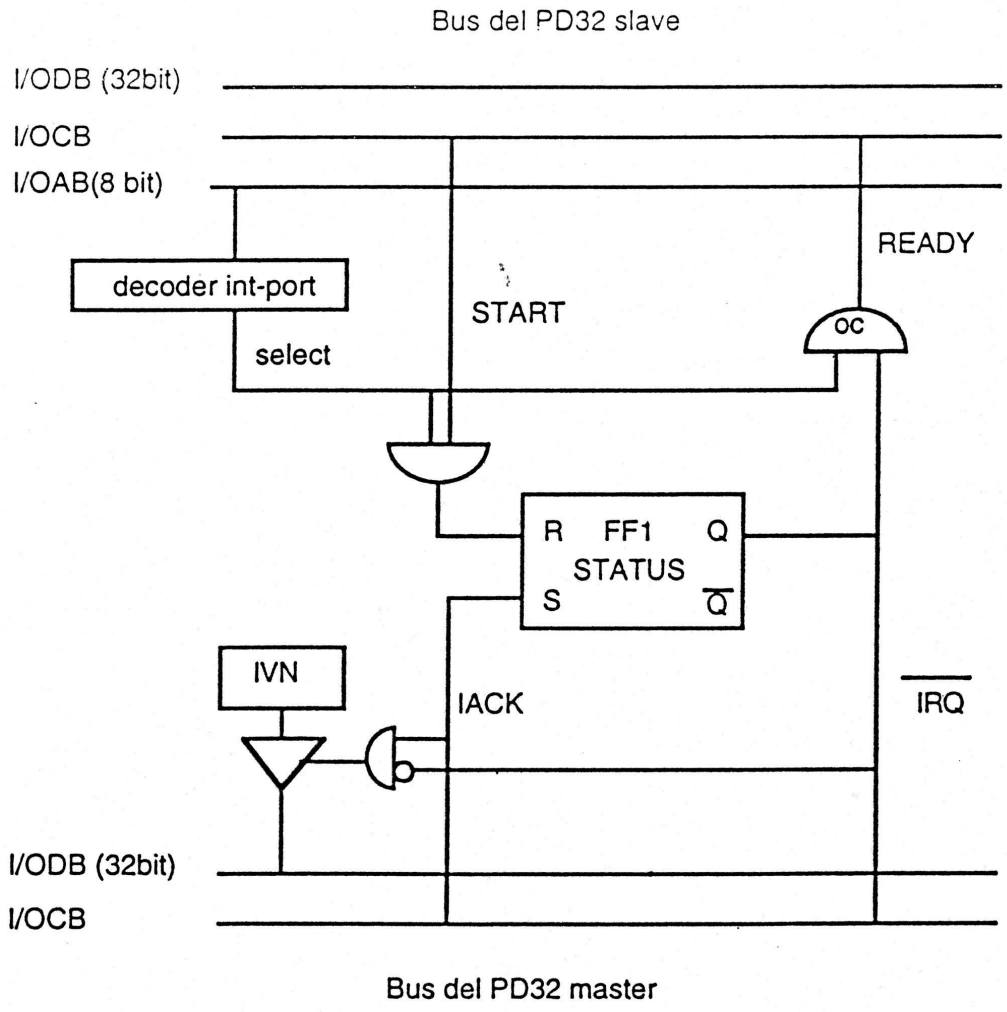
```
ac-dati:  xorl r0, r0              …
          xorl r2, r2              addl #1, r0
ciclo:    inl r0, AD1(r2)          cmpb #16, r0
          addl #4, r2              jnz ciclo
          …
```

Soluzione con processori PD32

Circuito per la gestione dell'interruzione lanciata dal PD32 slave

Abbiamo istruzioni dedicate per fare busy-waiting:

```
        start int-port
loop:   jnr int-port, loop
```

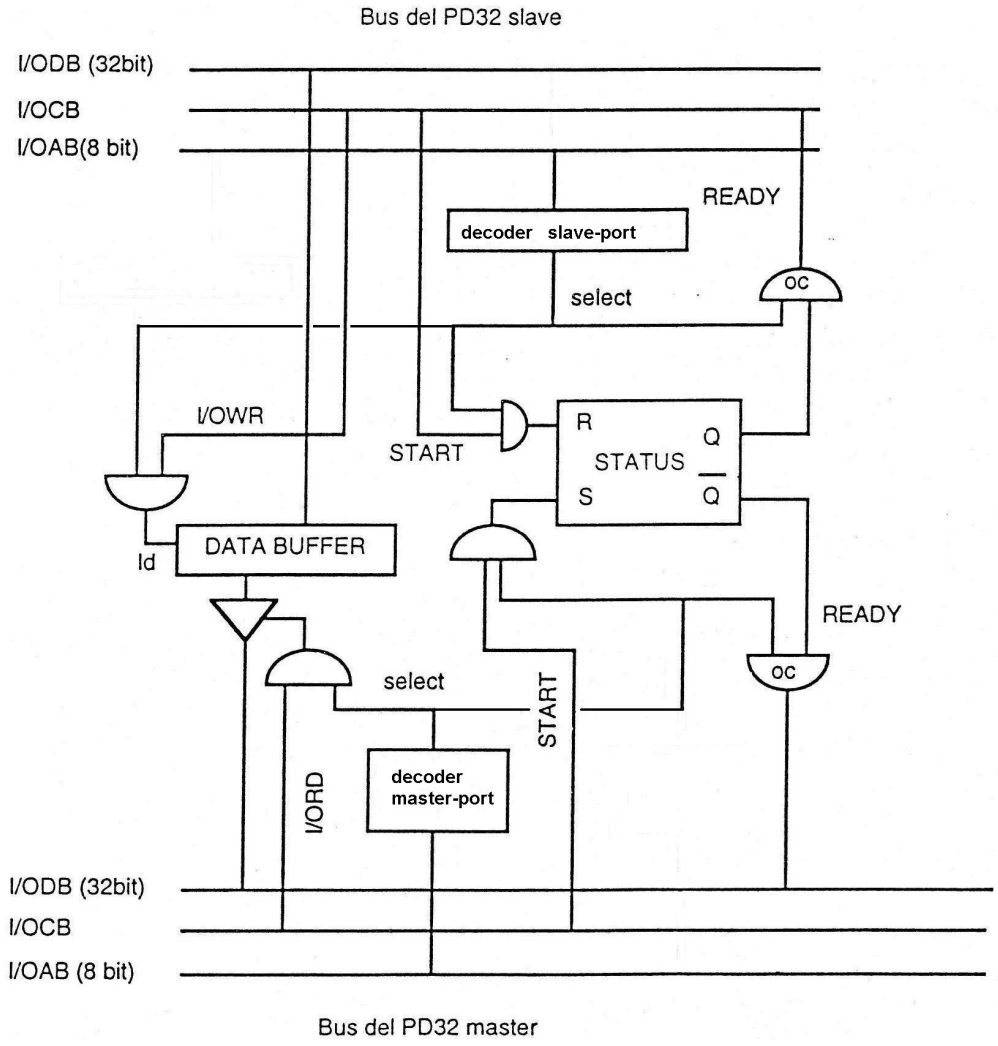


In questo contesto abbiamo IACK attivo alto e
FF1 commutante su fronte di discesa

Soluzione con processori PD32

Circuito per lo scambio dati e relativa sincronizzazione tra master e slave

- (slave) OUT slave-port→carica dato su DATA BUFFER
- (slave) START slave-port → reset FF STATUS
    - Sblocca master bloccato in busy waiting
- (master) carica dato in memoria
- (master) START master-port → set FF STATUS
    - Master si riblocca in busy waiting
    - Slave ottiene permesso per produrre nuovo dato



Soluzione con processori PD32

Software per lo scambio dati tra master e slave

SLAVE

Routine scambio-dati

```
        clri
        start slave-port
        start int-port
loop:   jnr int-port,loop
        xorl r0,r0
        movb #10h,r1
ciclo-tx: jnr slave-port,ciclo-tx
        movl AD2(r0),r3
        outl r3,slave-port
        start slave-port
        addl #4,r0
        subb #1,r1
        jz esci
        jmp ciclo-tx
esci:   seti
        ret
```

MASTER

Device Driver

```
        push r0
        push r3
        push r1
        xorl r0,r0
        movb #10h,r1
        start master-port
ciclo-rx: jnr master-port,ciclo-rx
        inl master-port,r3
        start master-port
        movl r3,ADM(r0)
        addl #4,r0
        subb #1,r1
        jz esci
        jmp ciclo-rx
esci:   pop r1
        pop r3
        pop r0
        reti
```

Esercizio #5 – 15/07/1999

Una periferica utilizza una sezione della memoria del PD32 per appoggiare dati temporanei secondo la disciplina LIFO. Ciascun dato a 32 bit viene depositato e successivamente recuperato dalla periferica tramite interruzione al processore. Una coppia di FF con significato di stack vuoto e stack pieno vengono settati dal micro per prevenire eventuali richieste anomale da parte della periferica. Definire l'interfaccia HW di I/O della periferica ed i moduli del programma assembler.

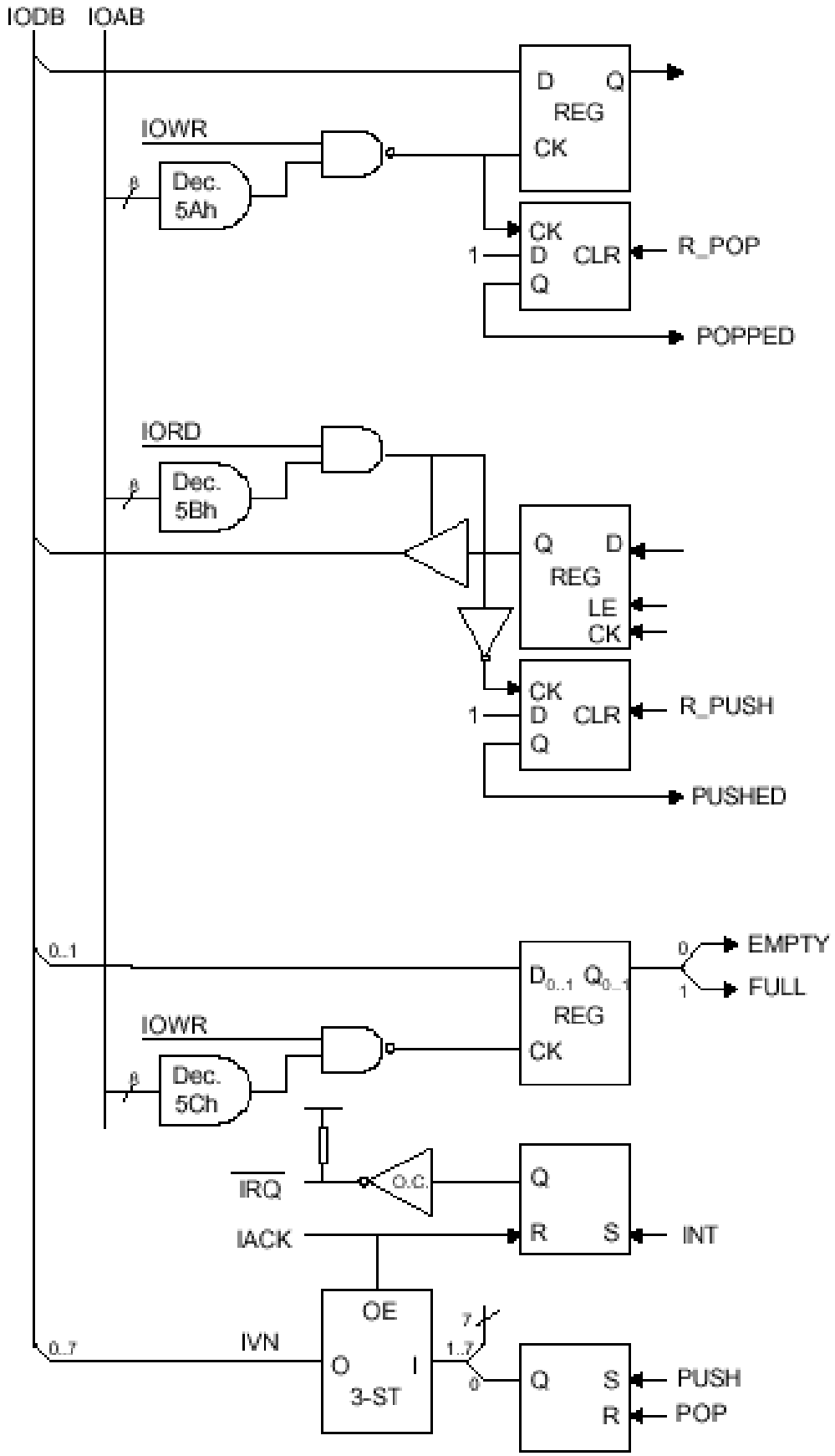
Analisi

- La periferica deve poter eseguire le operazioni di PUSH e POP
- Queste 2 operazioni coinvolgono lo scambio di un dato su bus IO (su base interruzione) e la lettura/scrittura su una sezione della memoria gestita a stack (diversa dallo stack del PD32)

HW, l'interfaccia di I/O comprende:

- Registro di input (push) + FF handshake
- Registro di output (pop) + FF handshake
- Registro (2 bit-FULL,EMPTY) sui cui il proc.scrive lo stato dello stack; non occorre aggiungere FF handshake, aggiorniamo registro prima di ogni operazione di lettura/scrittura (push/pop) sui registri
- Interfaccia di richiesta e riconoscimento interruzione, codifichiamo tipologia richiesta (push/pop) direttamente nell'IVN, altrimenti dovremmo prevedere un indirizzo aggiuntivo di IO per leggere il FF che indica la richiesta

#5 – 15/07/1999 --- Interfaccia IO



Routine Software (1/3)

```
                    org 500h
    pushreg equ 05Ah          ;indirizzo reg. push (in)
    popreg equ 05Bh           ;indirizzo reg. pop (out)
    fullmpty equ 05Ch         ;indirizzo reg. 2 bit (out)
    inivn  equ 000h      ;indirizzo driver periferica lifo in (push)
    outivn equ 001h      ;indirizzo driver periferica lifo out (pop)
    indriver equ 1000h     ;indirizzo driver periferica lifo in (push)
    outdrive equ 1100h     ;indirizzo driver periferica lifo out (pop)
    stackbot equ 500h     ;base stack periferica (256 byte)
    stacktop equ 400h     ;cima stack periferica (dall'alto al basso)
    stackpnt dl 000h

                    code
main:               movl #stackbot,stackpnt        ;stackpnt
                            ;puntatore all'area del messaggio out
                    outl #01h,fullmpty  ;EMPTY → reg. LIFO
                    seti
mainloop:           jmp mainloop
```

Routine Software (2/3)

```
; ************************
;lifo in (push)
            driver inivn, indriver
            push r0
            movl stackpnt,r0      ;test su disponibilità residua dello
             cmpl #stacktop,r0    ; stack
            jz exitin             ;ignora richiesta di push (stack pieno)
            subl #4,r0            ;aggiorna stack pointer (predecrem.)
            movl r0,stackpnt
            cmpl #stacktop,r0
            jnz nofull
            outb #02h,fullmpty    ; scrivi 1-0 nei bit full-empty del reg.
                                  ; fullmpty
            jmp pushdato
nofull:     outb #00h,fullmpty          ;scrivi 0-0 nei bit
                                        ;full-empty del reg.fullmpty
            jmp pushdato
pushdato:   inl pushreg,(r0)      ;leggi dato e caricalo nello stack
exitin:     pop r0
            rti
;************************
```

Routine Software (3/3)

```
;************************
;lifo out (pop)
            driver outivn, outdrive
            push r0
            movl stackpnt,r0     ;test su disponibilità residua dello
            cmpl #stackbot,r0    ;stack
            jz exitout           ;ignora richiesta di pop (stack vuoto)
            cmpl #stackbot-4,r0
            jnz noempty
            outb #01h,fullmpty           ;scrivi 0-1 nei bit
                                         ;full-empty del reg.fullmpty
            jmp popdato
noempty:    outb #00h,fullmpty ; scrivi 0-0 nei bit full-empty
                               ; del reg. fullmpty
            jmp popdato
popdato:    outl (r0),popreg   ; estrai dato dallo stack e passalo alla
                               ; perifer.
            addl #4,r0         ; aggiorna stack pointer (postincrem.)
            movl r0,stackpnt
exitout:    pop r0
            rti
;************************

            end
```