Versione H: I/O

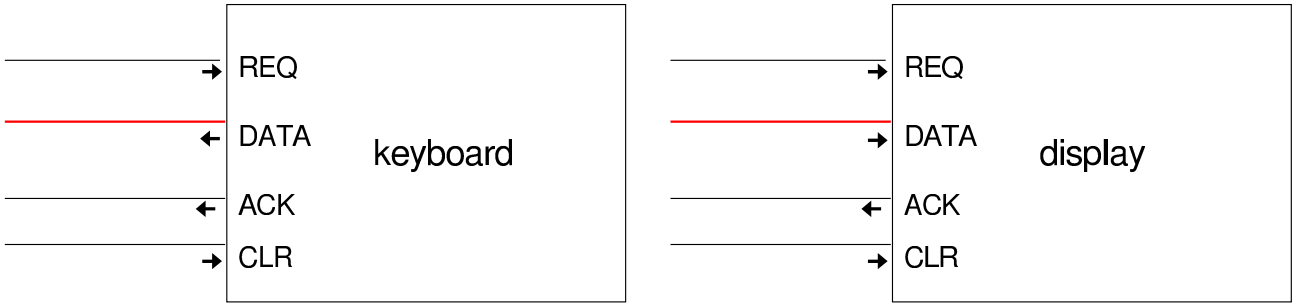
La versione precedente è abbastanza completa per dimostrare le funzionalità principali di una CPU; in questa versione si passa a estendere il progetto iniziale, per consentire il collegamento con dispositivi di input-output. Per fare questo viene aggiunto un bus dati e un bus indirizzi per l'I/O (input-output), inoltre, si aggiungono alcune linee nel bus di controllo (***CON***), da utilizzare per i componenti che costituiscono l'interfaccia con i dispositivi di I/O. Il progetto della CPU dimostrativa si basa su Tkgate e nelle sezioni successive si mostra anche il codice utilizzato per realizzare i dispositivi in questione con questo simulatore.

Generalizzazione della comunicazione con i dispositivi

«

I dispositivi di I/O sono normalmente asincroni rispetto alla CPU (nel senso che non sono regolati dal clock che amministra la CPU), pertanto si rende necessario un protocollo per richiedere un'azione al dispositivo e per recepirne il risultato, tipico dei componenti asincroni. I dispositivi di I/O utilizzati in questo progetto dimostrativo hanno esteriormente le connessioni che si possono vedere nella figura successiva.

Figura u113.1. Un dispositivo di input e un dispositivo di output.

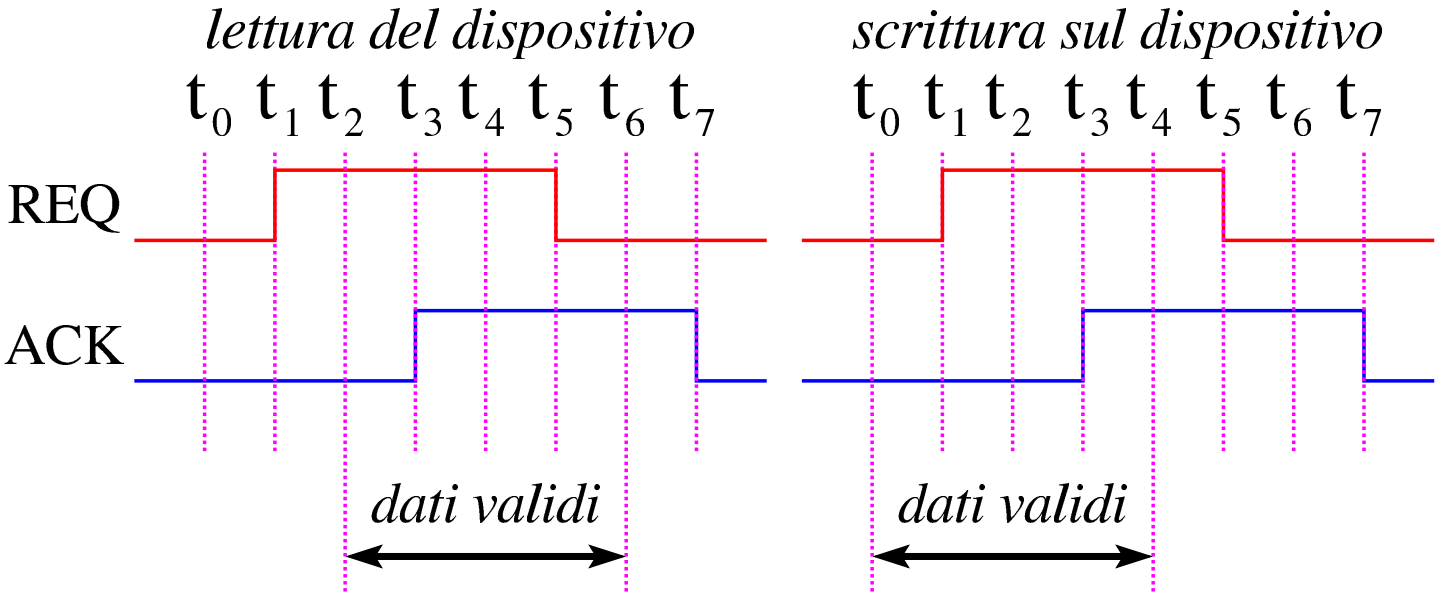


L'ingresso o l'uscita ***DATA*** consente di movimentare l'informazione che serve al dispositivo o che viene fornita dallo stesso. Gli ingressi ***REQ*** (*request*) e ***ACK*** (*acknowledge*) servono a negoziare il movimento dell'informazione (l'ingresso ***CLR*** serve ad azzerare il dispositivo). Vanno fatti due esempi, a seconda che si debba leggere un'informazione dal dispositivo, oppure che vi si voglia scrivere.

Per comunicare con un **dispositivo di input**, dal quale si deve leggere un'informazione, si comincia attivando l'ingresso ***REQ*** (t_1), con il quale si intende richiedere il dato. Il dispositivo riceve la richiesta e predispone l'uscita ***DATA*** con l'informazione (t_2); subito dopo, attiva l'uscita ***ACK*** (t_3). A questo punto, trovando attiva l'uscita ***ACK*** si può leggere il valore dall'uscita ***DATA*** (t_4) e al termine si può

disattivare l'ingresso ***REQ*** (t_5). Il dispositivo, osservando la disattivazione dell'ingresso ***REQ*** sa che l'informazione è stata recepita, quindi smettere di fornire l'informazione richiesta (t_6) e disabilita a sua volta l'uscita ***ACK*** (t_7).

Figura u113.2. Fasi della lettura da un dispositivo di input e della scrittura su un dispositivo di output.



Per comunicare con un **dispositivo di output**, nel quale si deve scrivere un'informazione, si comincia fornendo l'informazione sull'ingresso ***DATA*** (t_0); subito dopo, si attiva l'ingresso ***REQ*** (t_1) per informare il dispositivo della disponibilità dell'informazione. Quindi il dispositivo recepisce l'informazione (t_2) e poi dà conferma attivando l'uscita ***ACK*** (t_3). Ricevendo la conferma, non è più necessario trattenere l'informazione disponibile nell'ingresso ***DATA*** (t_4), quindi viene disattivato l'ingresso ***REQ*** (t_5) e dopo di questo il dispositivo disattiva l'uscita ***ACK*** (t_7) concludendo l'operazione.

Realizzazione dei dispositivi di I/O

«

In questo progetto, per il momento, si realizzano soltanto due dispositivi di I/O: una tastiera e uno schermo. Dato che il progetto è sviluppato con Tkgate, i dispositivi vanno dichiarati attraverso codice Tcl/Tk. Il codice che viene mostrato qui è stato ottenuto modificando un dispositivo di esempio che fa parte della distribuzione di Tkgate.

Listato u113.3. File ‘`share/tkgate/vpd/kbd.tcl`’ per simulare l’input di una tastiera. Il programma si limita a mostrare una piccola finestra vuota, selezionando la quale è possibile digitare da tastiera qualcosa che deve essere recepito dal dispositivo virtuale corrispondente.

```
VPD::register KBD
VPD::allow KBD::post

namespace eval KBD {
    # Dichiarazione delle variabili pubbliche. $kbd_w è un array
    # di cui si utilizza solo l'elemento $n, il quale identifica
    # univocamente l'istanza dell'interfaccia in funzione.
    variable kbd_w
    variable KD
    # Funzione richiesta da Tkgate per creare l'interfaccia.
    proc post {n} {
        variable kbd_w
        # Crea una finestra e salva l'oggetto in un elemento dell'array
        # $kbd_w.
        set kbd_w($n) [VPD::createWindow "KBD $n" -shutdowncommand "KBD::unpost $n"]
        # Collega la digitazione della tastiera, relativa all'oggetto
        # rappresentato da $kbd_w($n), alla funzione sendChar.
        bind $kbd_w($n) <KeyPress> "KBD::sendChar $n \"%A\""
        # Apre un canale di scrittura, denominato «KD».
        if {[info exists ::tkgate_isInitialized]} {
            VPD::outsignal $n.KD KBD::KD($n)
        }
    }
    # Funzione che recepisce la digitazione e la immette nel canale
    # denominato «KD», relativo all'istanza attuale dell'interfaccia.
```

```
    proc sendChar {n key} {
        variable KD
        if { [string length $key ] == 1 } {
            binary scan $key c c
            set KBD::KD($n) $c
        }
    }
    # Funzione richiesta da Tkgate per distruggere l'interfaccia.
    proc unpost {n} {
        variable kbd_w
        destroy $kbd_w($n)
        unset kbd_w($n)
    }
}
```

Listato u113.4. File ‘`share/tkgate/vpd/scr.tcl`’ per simulare l’output su schermo a caratteri. Il programma mostra una finestra sulla quale possono poi apparire i caratteri trasmessi. Nel codice si fa riferimento al file ‘`textcurs.b`’ che è già disponibile nella distribuzione di Tkgate.

```
image create bitmap txtcurs -file "$bd/txtcurs.b"

VPD::register SCR
VPD::allow SCR::post
VPD::allow SCR::data

namespace eval SCR {
    # Dichiarazione delle variabili pubbliche: si tratta di array
    # dei quali si utilizza solo l'elemento $n, il quale identifica
    # univocamente l'istanza dell'interfaccia in funzione.
    variable scr_w
    variable scr_pos
    # Funzione richiesta da Tkgate per creare l'interfaccia.
    proc post {n} {
        variable scr_w
        variable scr_pos
        # Crea una finestra e salva l'oggetto in un elemento dell'array
        # $scr_w.
        set scr_w($n) [VPD::createWindow "SCR $n" -shutdowncommand "SCR::unpost $n"]
        # Per maggiore comodità, copia il riferimento all'oggetto nella
        # variabile locale $w e in seguito fa riferimento all'oggetto
        # attraverso questa seconda variabile.
```

```
        set w $scr_w($n)
        text $w.txt -state disabled
        pack $w.txt
        # Mette il cursore alla fine del testo visualizzato.
        $w.txt image create end -image txtcurs
        # Apre un canale di lettura, denominato «RD», associandolo
        # alla funzione «data».
        if {[info exists ::tkgate_isInitialized]} {
            VPD::insignal $n.RD -command "SCR::data $n" -format %d
        }
        # Azzera il contatore che tiene conto dei caratteri visualizzati
        # sullo schermo.
        set scr_pos($n) 0
    }
    # Funzione richiesta da Tkgate per distruggere l'interfaccia.
    proc unpost {n} {
        variable scr_w
        destroy $scr_w($n)
        unset scr_w($n)
    }
    # Funzione usata per recepire i dati da visualizzare.
    proc data {n c} {
        variable scr_w
        variable scr_pos
        # Per maggiore comodità, copia il riferimento all'oggetto che
        # rappresenta l'interfaccia nella variabile $w.
        set w $scr_w($n)
        catch {
            # La variabile $c contiene il carattere da visualizzare.
            if { $c == 7 } {
                # BEL
                bell
                return
            } elseif { $c == 127 || $c == 8 } {
                # DEL | BS
                if { $scr_pos($n) > 0 } {
                    # Cancella l'ultimo carattere visualizzato, ma solo
                    # se il contatore dei caratteri è maggiore di zero,
                    # altrimenti sparirebbe il cursore e la
                    # visualizzazione verrebbe collocata in un'area
                    # non visibile dello schermo.
                    $w.txt configure -state normal
                    $w.txt delete "end - 3 chars"
                    $w.txt see end
                    $w.txt configure -state disabled
                    set scr_pos($n) [expr {$scr_pos($n) - 1}]
```

```
                }
                return
            } elseif { $c == 13 } {
                # CR viene trasformato in LF.
                set c 10
            }
            # Converte il numero del carattere in un simbolo
            # visualizzabile.
            set x [format %c $c]
            # Visualizza il simbolo.
            $w.txt configure -state normal
            $w.txt insert "end - 2 chars" $x
            $w.txt see end
            $w.txt configure -state disabled
            # Aggiorna il contatore dei caratteri visualizzati.
            set scr_pos($n) [expr {$scr_pos($n) + 1}]
        }
    }
}
```

I due programmi Tcl/Tk servono a fornire due moduli software, a cui poi si fa riferimento attraverso del codice Verilog. Nei listati successivi si vedono i moduli **keyboard** e **display** che graficamente si mostrano esattamente come nella figura u113.1.

Listato u113.5. Codice Verilog per il modulo **keyboard**.

```
module keyboard(DATA, REQ, ACK, CLR);
output ACK;
output [7:0] DATA;
input REQ;
input CLR;
reg ready;
reg [7:0] key;

  initial
    begin
      ready = 0;
      key = 0;
```

```
    end

  always
    begin
      @(posedge CLR)
      ready = 0;
      key = 0;
    end

  initial $tkg$post("KBD","%m");

  always
    begin
      @ (posedge REQ);
      # 5;
      key = $tkg$recv("%m.KD");
      # 5;
      ready = 1'b1;
      # 5;
      @ (negedge REQ);
      # 5;
      ready = 1'b0;
    end

  assign DATA = key;
  assign ACK = ready;

endmodule
```

Listato u113.6. Codice Verilog per il modulo **`display`**.

```
module display(DATA, REQ, ACK, CLR);
output ACK;
input [7:0] DATA;
input REQ;
```

```
input CLR;
reg ready;

  initial
    begin
      ready = 0;
    end

  initial $tkg$post("SCR","%m");

  always
    begin
      @(posedge CLR)
      ready = 0;
    end

  always
    begin
      @(posedge REQ);
      # 5;
      $tkg$send("%m.RD",DATA);
      # 5;
      ready = 1'b1;
      # 5;
      @(negedge REQ);
      # 5;
      ready = 1'b0;
    end

  assign ACK = ready;

endmodule
```

Fino a qui, i dispositivi descritti funzionano in modo asincrono, ma

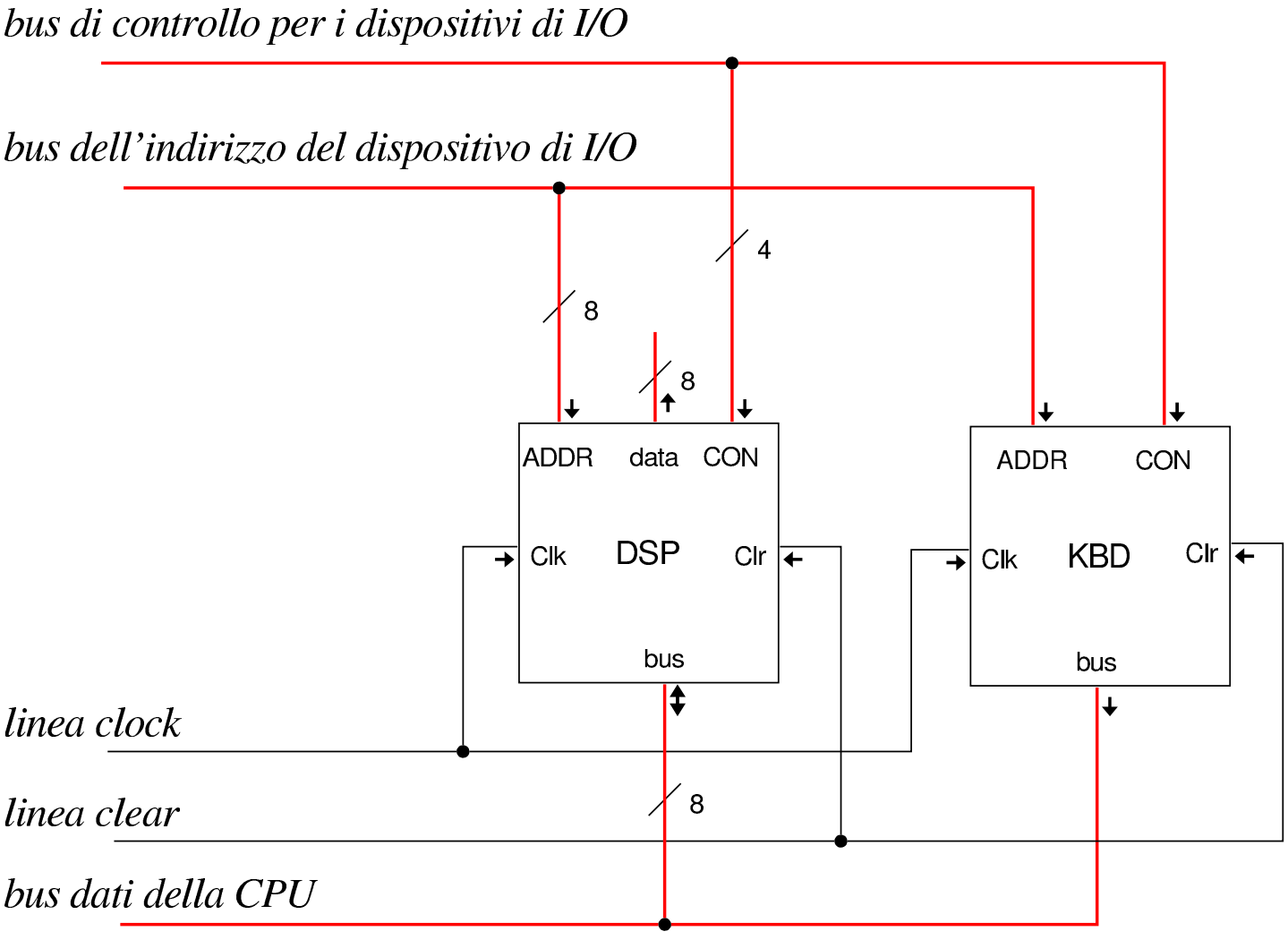
per essere utilizzati devono essere adattati per poter funzionare in modo sincrono.

Aspetto e funzionamento esteriore delle interfacce sincrone

«

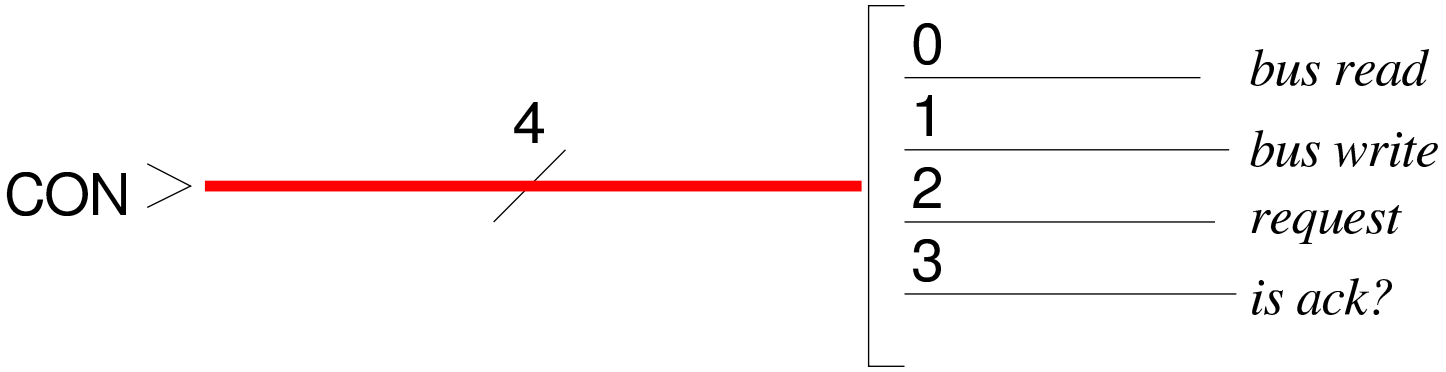
Le interfacce sincrone dei dispositivi di I/O devono potersi collegare al bus della CPU come gli altri moduli già presenti; tuttavia, per semplificare il cablaggio, invece di disporre di linee di controllo personali, viene creato un bus aggiuntivo, in cui indicare l'indirizzo del modulo a cui si vuole fare riferimento.

Figura u113.7. Connessione dei moduli di I/O.



Come si vede dal disegno, il bus connesso agli ingressi ***ADDR*** serve a selezionare il dispositivo, mentre il bus connesso agli ingressi ***CON*** serve a uniformare le linee di controllo per tutti i moduli di I/O. In pratica, viene mostrato in seguito che questi due bus aggiuntivi provengono dallo stesso bus di controllo complessivo.

Gli ingressi ***CON*** sono uniformi, ma ogni modulo utilizza solo ciò che gli serve, ignorando il resto:

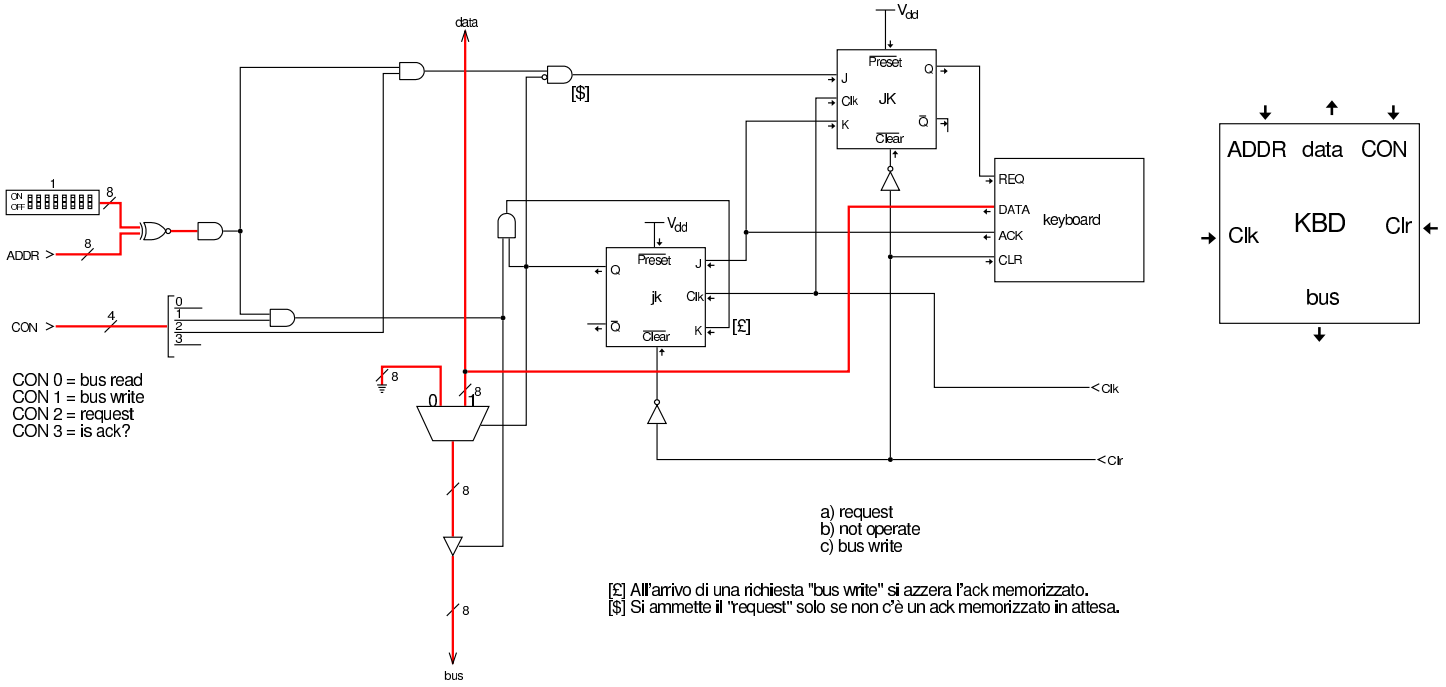


Le linee ***bus read*** e ***bus write*** sono le stesse degli altri moduli già descritti, riferendosi all'ordine di leggere o di scrivere sul bus della CPU. La linea ***request*** serve a richiedere l'operazione di lettura o scrittura del dispositivo, ma senza la necessità di mantenerla attiva come nel protocollo di comunicazione asincrona. La linea ***is ack?*** serve a ottenere, in qualche modo, l'informazione sul fatto che sia stata ricevuta la conferma da parte del dispositivo.

Interfaccia sincrona della tastiera

Il modulo di interfaccia della tastiera utilizza solo due delle quattro linee di controllo: ***bus write*** e ***request***.

Figura u113.9. Modulo **KBD** che collega la tastiera al bus della CPU.



Per comunicare con il modulo della tastiera, è necessario inizialmente un segnale ***request*** che all'arrivo dell'impulso di clock viene memorizzato nel flip-flop JK superiore, attivandolo; tale flip-flop ha il compito di trasferire e fissare tale valore sull'ingresso ***REQ*** del dispositivo. Il secondo flip-flop JK, in posizione centrale, ha invece il compito di memorizzare l'esito emesso dall'uscita ***ACK*** e, se tale valore risulta memorizzato, non permette la ricezione di una nuova richiesta. Dopo la ricezione di un segnale ***request*** acquisito correttamente, nella migliore delle ipotesi, il dispositivo ha già accumulato un carattere digitato da tastiera e risponde quasi subito mettendo tale valore nella sua uscita ***DATA*** e poi attivando la sua uscita ***ACK***. Al secondo impulso di clock, il flip-flop che manteneva attivo l'ingresso ***REQ*** si azzera e invece si attiva il secondo flip-flop al centro del disegno; tuttavia, il dispositivo mantiene il valore dell'uscita ***DATA***. A questo punto si riceve il segnale ***bus write*** che consente di im-

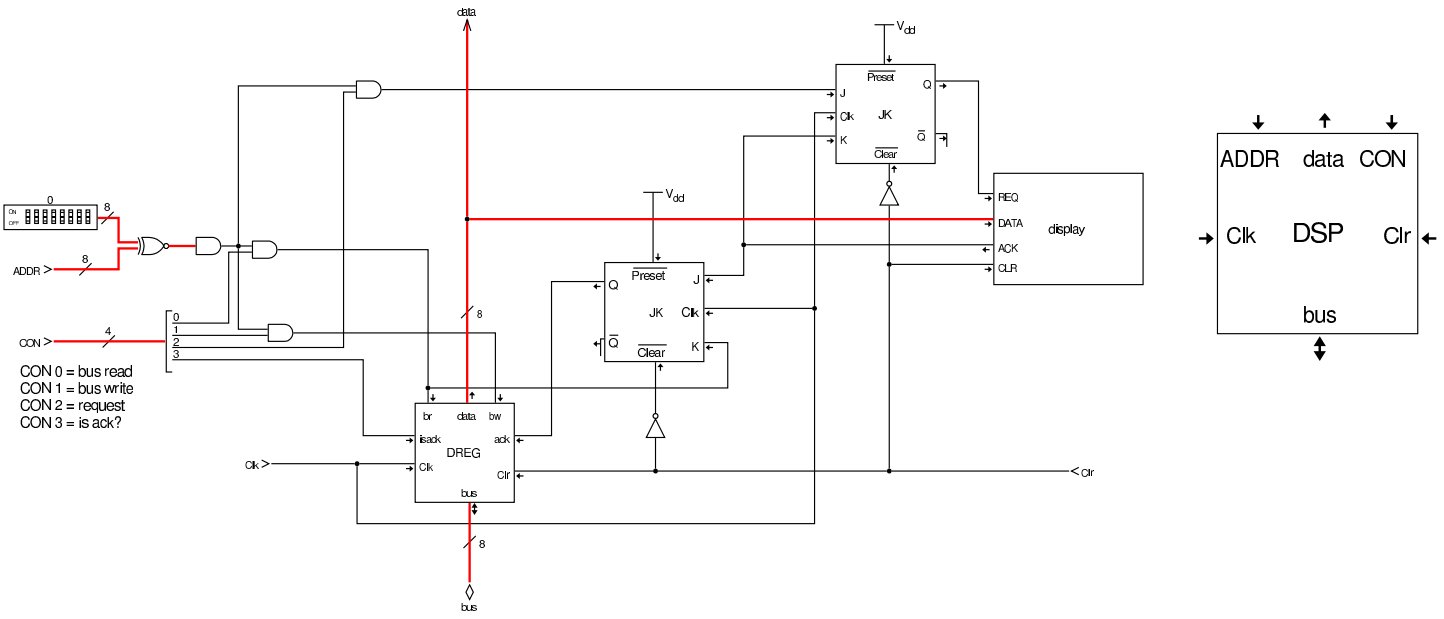
mettere il valore ricevuto dalla tastiera nel bus della CPU, azzerando contestualmente il flip-flop centrale. Se invece non si riesce a ottenere un carattere dalla tastiera, nel tempo stabilito, si ottiene il valore nullo (00_{16}), dato che manca l'attivazione del flip-flop che conserva lo stato di ***ACK***.

A livello di macrocodice, se la lettura della tastiera produce un carattere nullo, significa che non c'è alcun carattere pronto e occorre ripetere la lettura.

Interfaccia sincrona dello schermo

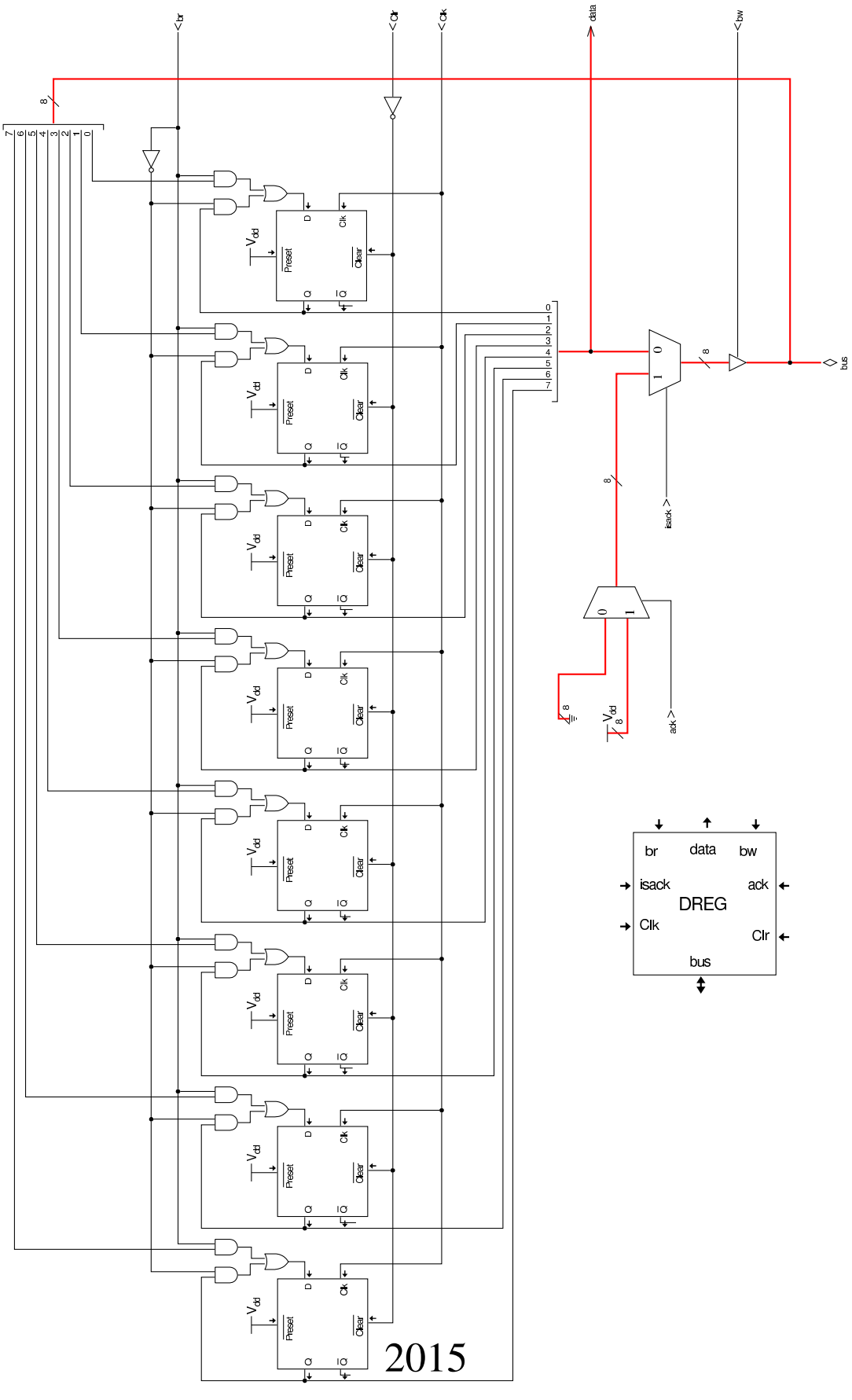
Il modulo **`DSP`** utilizza tutte le linee dell'ingresso di controllo, in quanto deve poter leggere dal bus della CPU, quando si vuole visualizzare un carattere sullo schermo, ma deve anche poter scrivere sul bus, per fornire un codice di conferma della riuscita della visualizzazione.

Figura u113.10. Modulo **`DSP`** che collega lo schermo al bus della CPU.



Il modulo **`DSP`** utilizza un registro modificato che serve principalmente per memorizzare il carattere da rappresentare sullo schermo; tuttavia, quando si attiva il suo ingresso ***isack***, può immettere nel bus della CPU l'esito della visualizzazione: 00_{16} vuol dire che questa non è ancora stata confermata, mentre FF_{16} indica una rappresentazione avvenuta correttamente.

Figura u113.11. Registro **DREG** che accumula il carattere da visualizzare.



Per visualizzare un carattere sullo schermo, si comincia attivando la linea ***bus read***: all'arrivo dell'impulso di clock il registro accumula il carattere leggendolo dal bus della CPU, mentre il flip-flop JK centrale si azzera, azzerando così l'ingresso ***ack*** del registro ***DREG***. Quindi deve essere attivata la linea ***request*** e all'arrivo dell'impulso di clock questo valore viene memorizzato nel flip-flop JK superiore, il quale attiva così l'ingresso ***REQ*** del dispositivo. A quel punto, ritardo di propagazione permettendo, il dispositivo mostra il carattere già presente nel suo ingresso ***DATA*** (proveniente dal registro ***DREG*** e a un certo punto risponde attivando la sua uscita ***ACK***. Quando l'uscita ***ACK*** del dispositivo si attiva e sopraggiunge un impulso di clock, il registro che manteneva il segnale ***REQ*** si azzera, mentre si attiva il flip-flop JK centrale, attivando di conseguenza l'ingresso ***ack*** del registro ***DREG***.

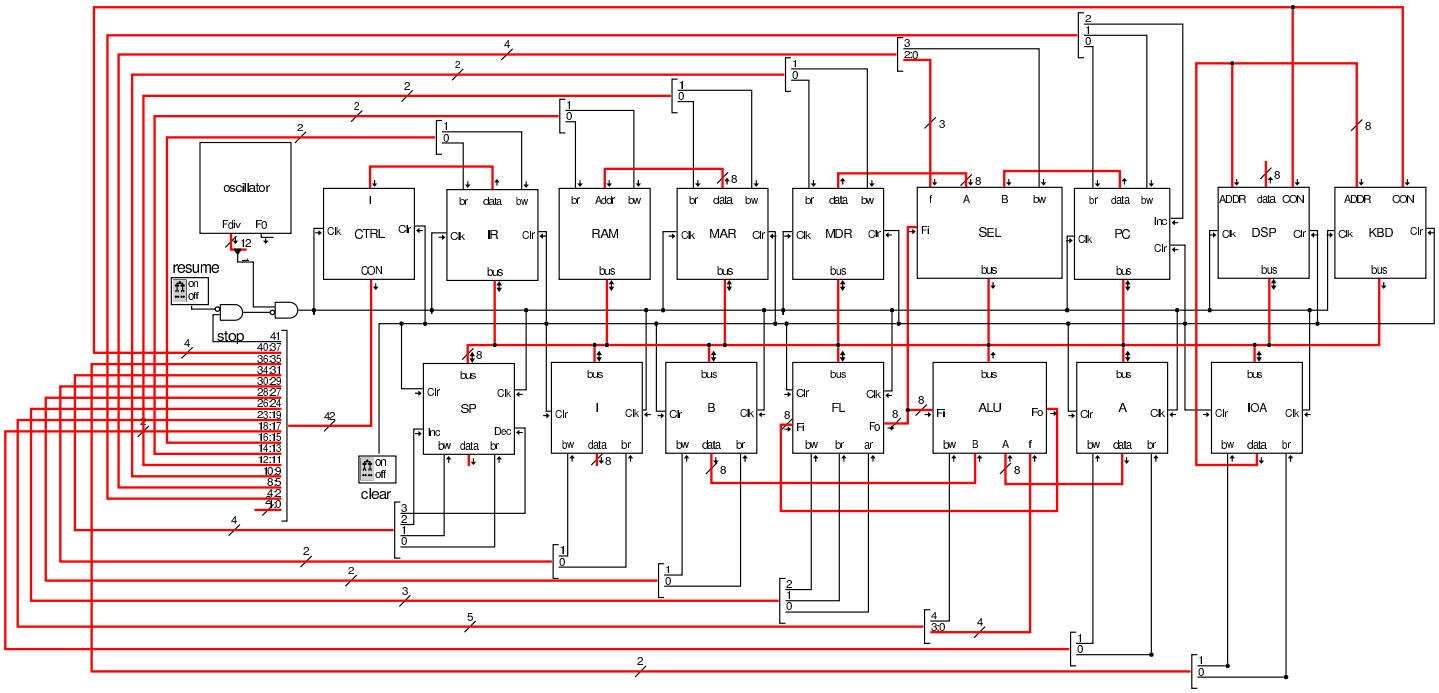
Per la visualizzazione di un carattere, sono sufficienti i due cicli di clock iniziali, ma per verificare che la visualizzazione sia avvenuta effettivamente, occorre intervenire nuovamente con un'interrogazione. In tal caso si attiva la linea ***is ack?*** e ***bus write***, in modo da immettere nel bus della CPU il valore che può essere 00_{16} o FF_{16}, a seconda del fatto che non sia ancora stata ottenuta la conferma oppure che invece questa ci sia stata.

Il bus della CPU con i dispositivi di I/O

«

Nel bus della CPU, oltre ai moduli dei dispositivi di I/O, si aggiunge un registro, denominato ***IOA***, con lo scopo di conservare l'indirizzo del dispositivo di I/O con il quale si vuole comunicare.

Figura u113.12. Il bus della CPU con l'aggiunta del registro ***IOA*** e dei moduli di I/O.



Dal disegno si può vedere che il bus di controllo complessivo è stato modificato, inserendo delle linee per il controllo del registro ***IOA*** e le quattro linee necessarie a controllare i dispositivi di I/O, spostando di conseguenza la linea usata per fermare il segnale di clock. Pertanto, nel codice della dichiarazione delle memorie e in quello che descrive i campi del bus di controllo, si apportano le modifiche seguenti:

```
map         bank[7:0]    ctrl.m0;
microcode   bank[31:0]   ctrl.m1;
microcode   bank[41:32]  ctrl.m2;
macrocode   bank[7:0]    ram.m3;
…
field ioa_br[35];          // IOA <-- bus
field ioa_bw[36];          // IOA --> bus
field io_br[37];           // I/O <-- bus
field io_bw[38];           // I/O --> bus
field io_req[39];          // I/O request
field io_isack[40];        // I/O is ack?
```

```
field stop[41];          // stop clock
```

Nell'elenco dei codici operativi si aggiungono istruzioni nuove e lo stesso poi nella descrizione del microcodice; in particolare, si ammettono macroistruzioni che richiedono due argomenti:

```
operands op_2 {
    //
    // [........][mmmmmmmm][nnnnnnnn]
    //
    #1,#2 = { +1=#1[7:0]; +2=#2[15:8]; };
};
…
op in {
  map in : 48;                    // read input from I/O bus
  +0[7:0]=48;
  operands op_1;
};
op out {
  map out : 49;                   // write output to I/O bus
  +0[7:0]=49;
  operands op_1;
};
op io_is_ack {
  map io_is_ack : 50;
  +0[7:0]=50;
  operands op_2;
};
```

```
begin microcode @ 0
…
in:
    mar_br pc_bw;               // MAR <-- PC
    pc_Inc;                     // PC++
    mdr_br ram_bw;              // MDR <-- RAM[mar]
    ioa_br mdr_bw;              // IOA <-- MDR
    io_req;                     //
    io_br;                      // non fa alcunché
```

```
    a_br io_bw;                    // A <-- I/O
    ctrl_start ctrl_load;          // CNT <-- 0

out:
    mar_br pc_bw;                  // MAR <-- PC
    pc_Inc;                        // PC++
    mdr_br ram_bw;                 // MDR <-- RAM[mar]
    ioa_br mdr_bw;                 // IOA <-- MDR
    io_br a_bw;                    // IO <-- A
    io_req;                        //
    ctrl_start ctrl_load;          // CNT <-- 0

io_is_ack:
    mar_br pc_bw;                  // MAR <-- PC
    pc_Inc;                        // PC++
    mdr_br ram_bw;                 // MDR <-- RAM[mar]
    ioa_br mdr_bw;                 // IOA <-- MDR
    //
    mar_br pc_bw;                        // MAR <-- PC
    pc_Inc;                              // PC++
    mdr_br ram_bw;                       // MDR <-- RAM[mar]
    a_br io_bw io_isack;                 // A <-- I/O is ack
    a_br alu_f=not_a alu_bw fl_ar;       // A <-- NOT A
    a_br alu_f=not_a alu_bw fl_ar;       // A <-- NOT A
    pc_br sel_f=if_not_zero sel_bw       // PC = (not_zero ? MAR : PC)
    ctrl_start ctrl_load;                // CNT <-- 0
…
end
```

Figura u113.16. Corrispondenza con il contenuto della memoria che rappresenta il microcodice (la coppia ***m1*** e ***m2*** dell'unità di controllo).

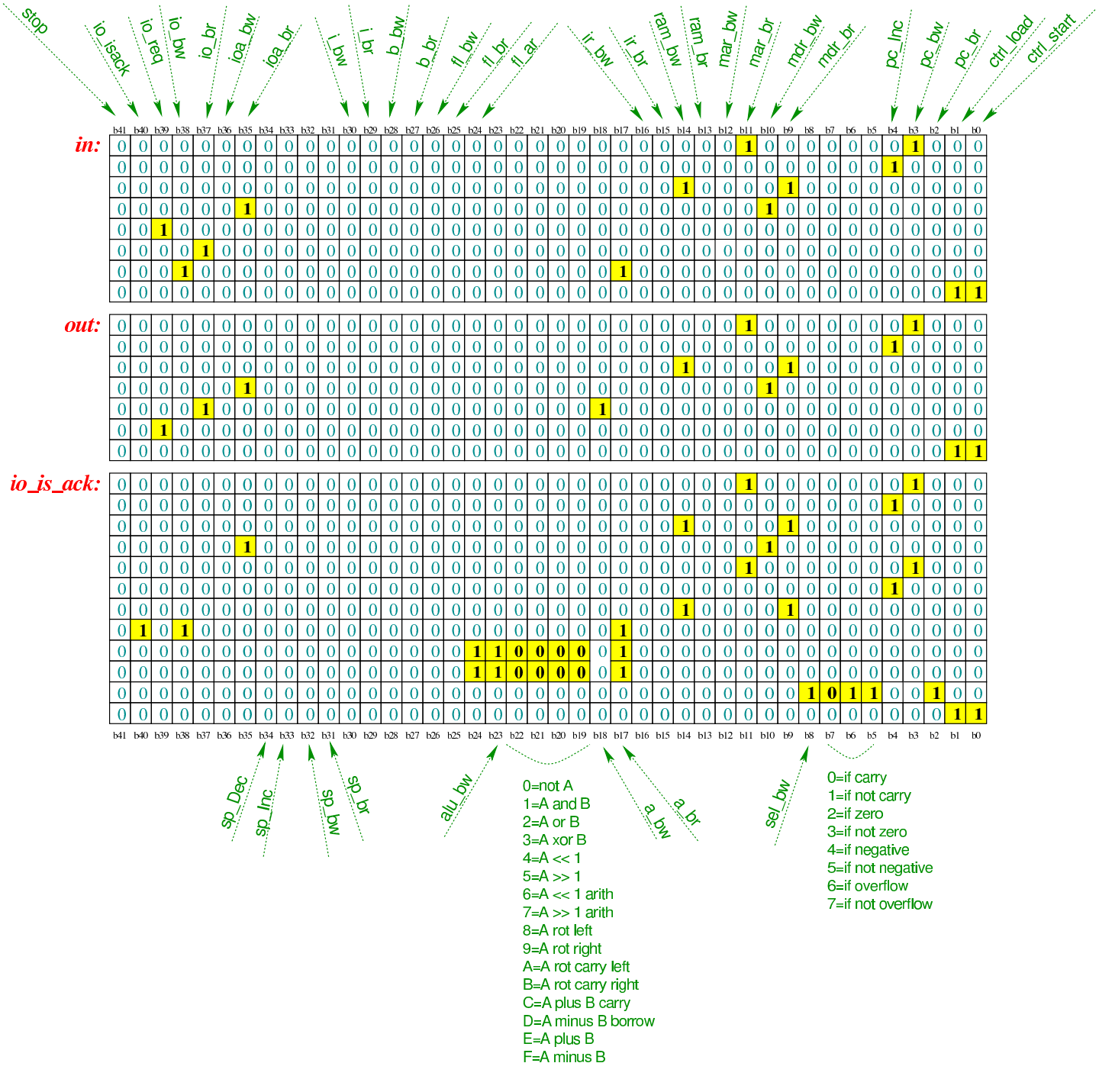


Tabella u113.17. Elenco delle macroistruzioni aggiunte in questa versione della CPU dimostrativa.

Sintassi	Descrizione
`in` ***indirizzo_io***	Legge un byte dal dispositivo a cui corrisponde l'indirizzo.
`out` ***indirizzo_io***	Scrive un byte sul dispositivo a cui corrisponde l'indirizzo.
`io_is_ack` ***indirizzo_io indirizzo***	Legge dal dispositivo indicato dal primo argomento, un codice che consente di verificare il ricevimento della conferma (*acknowledge*); se l'esito è valido, salta all'indirizzo indicato come secondo argomento.

Istruzione «out»

«

In questa sezione viene mostrato l'uso della macroistruzione '**`out`**', per visualizzare un carattere attraverso il dispositivo **`DSP`**. Nel listato successivo si mostra l'uso di '**`out`**' e poi anche '**`io_is_ack`**' per verificare che il carattere da visualizzare sia stato effettivamente mostrato. Il programma si limita a mostrare la lettera «H», ripetutamente, senza fermarsi.

Listato u113.18. Macrocodice per sperimentare l'istruzione **`out`** e **`io_is_ack`**: si vuole visualizzare la lettera «H», ripetutamente, controllando ogni volta il completamento dell'operazione. Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile presso *allegati/circuiti-logici/scpu-sub-h-out.gm* .

```
begin macrocode @ 0
start:
        load_imm #carattere
        move_mdr_a
        out 0                       // display
check_ack:
        io_is_ack 0, #start
        jump #check_ack
stop:
        stop
carattere:
        .byte 0x48                  // 'H'
end
```

Anche per questo esempio è disponibile un video: *http://www.youtube.com/watch?v=S9XqmTMYAj4* .

Istruzione «in»

«

In questa sezione viene mostrato l'uso della macroistruzione '**in**', per recepire la digitazione da tastiera, attraverso il modulo **KBD**. Nel listato successivo si usa anche l'istruzione '**out**', usata per riemettere il carattere ricevuto.

Listato u113.19. Macrocodice per sperimentare l'istruzione **in**: si vuole recepire la digitazione da tastiera, la quale viene riemessa attraverso l'istruzione **out** sul dispositivo **DSP**. Il file completo che descrive le memorie per Tkgate dovrebbe essere disponibile presso *allegati/circuiti-logici/scpu-sub-h-in.gm* .

```
begin macrocode @ 0
start:
        in 1
        not
```

```
        not
        jump_if_zero #start
        out 0
        jump #start
stop:
        stop
end
```

Video: *http://www.youtube.com/watch?v=JhGoQhssWQM* .

2024