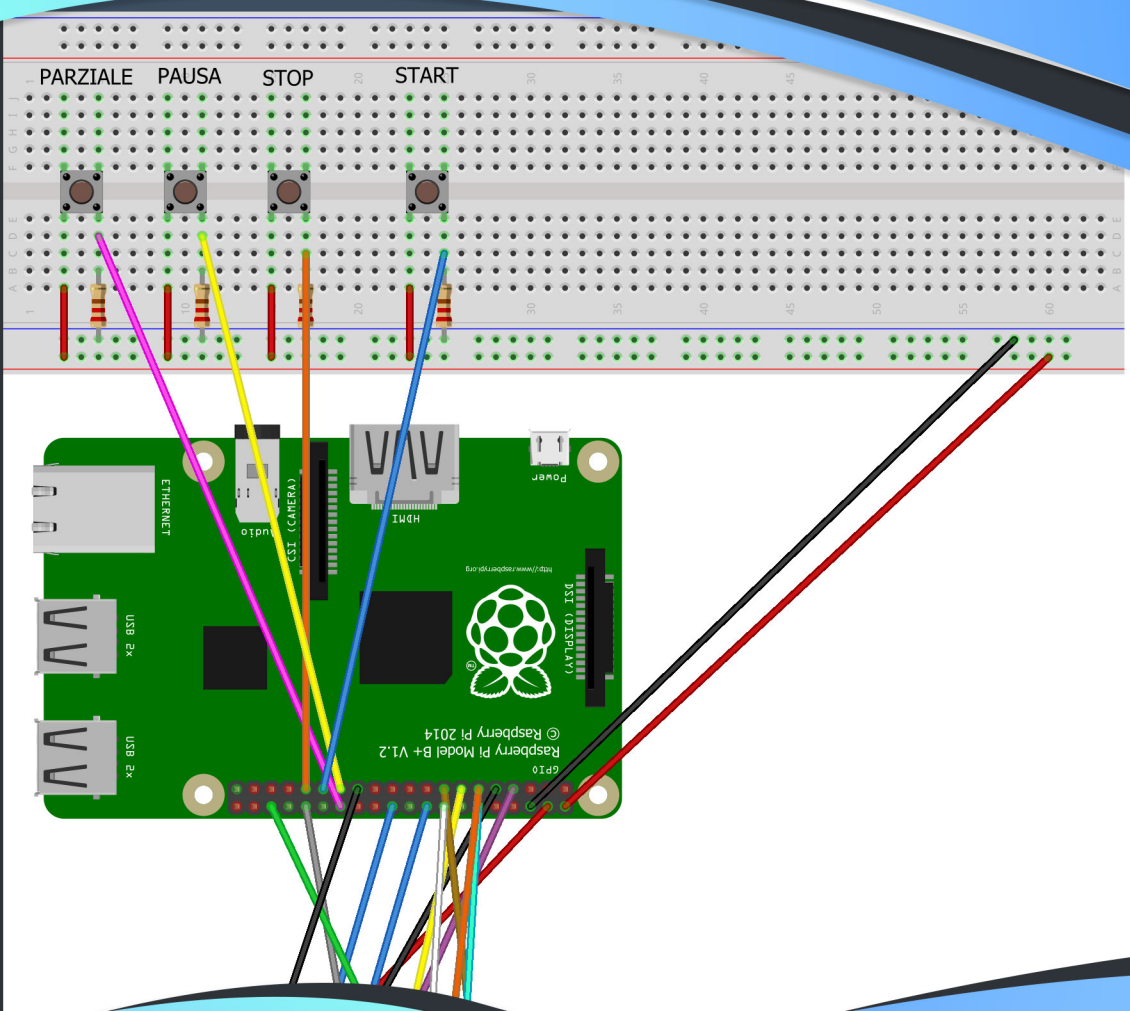


Cronometro

GAETANO DI GRAZIA - GIUSEPPE NASO

16th April

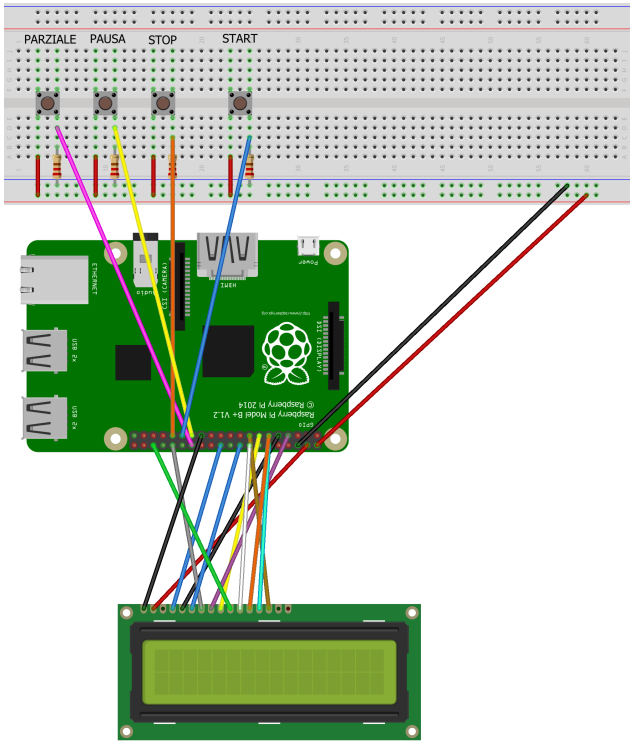
Contents

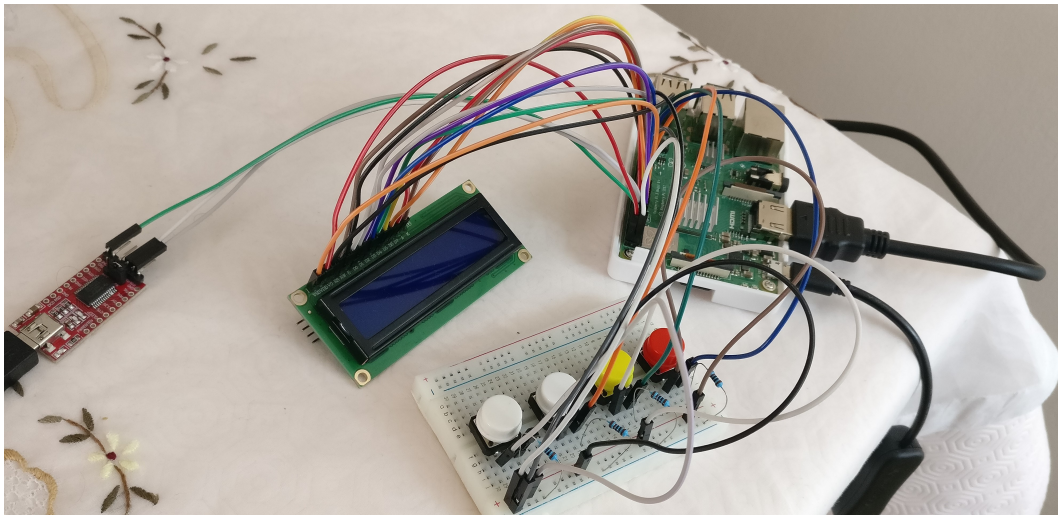
1 Introduzione

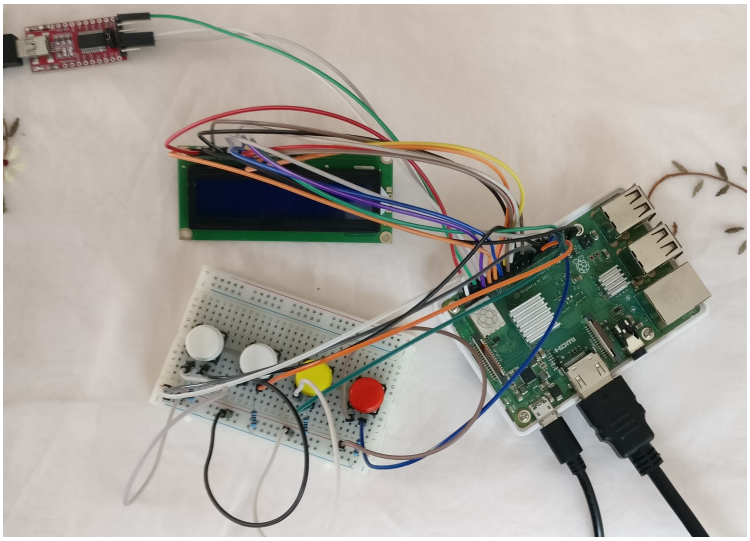
Il progetto consiste in un cronometro comandato da pulsanti che permettono di avviare, stoppare, mettere in pausa il tempo, inoltre è possibile stampare dei tempi parziali.

Il tutto viene stampato su un display LCD 16x02.

2 Schema



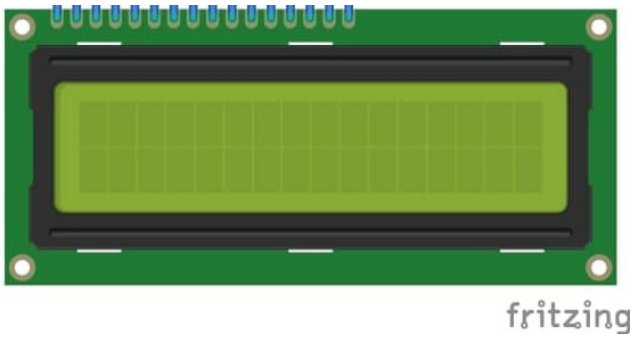




3 Hardware

- pc: un generico computer su cui abbiamo installato G-Forth, un programma di emulazione di terminale;
- breadboard: è una basetta che serve a creare prototipi di circuiti elettrici;
- Raspberry Pi 3 B+;
- interruttori;
- scheda SD: all'interno di essa è necessario caricare un sistema operativo bare metal per Raspberry Pi, basata su Jonesforth-ARM;
- UART (Universal Asynchronous Receiver – Transmitter): è un dispositivo hardware che permette il trasferimento di dati, bit in particolare, in modo seriale attraverso la USB e quindi dal computer in uso;
- LCD1602: un display avente 16 caratteri disposti su due diverse righe.

3.1 LCD16x02



Nel display LCD (ad un controller) raffigurato nell'immagine sopra sono presenti diversi pin, tra cui: VSS, VDD, VEE, RS, RW, E e otto diversi D (da D0 a D7).
Nello specifico ai pin VSS, CC e VEE vengono collegate rispettivamente massa, 5V ed un potenziometro per l'aggiustamento del contrasto.
I pin RS ed R/W possono avere valore 0 o 1, nel caso dell'RS se questo è 0 è settato a Instruction input, se ha valore 1 a Data input; per quanto riguarda R/W, invece, se ha valore 0 corrisponde a Write to LCD module altrimenti, a 1, Read from LCD module.
Il Pin E, di Enable, abilita la trasmissione dei segnali.
Ultimi, ma non per importanza, abbiamo i pin da D0 a D7, chiamati D da Data bus line, per i quali il D0 rappresenta il LSB mentre il D7 è il MSB.

3.1.1 DDRAM

La RAM di visualizzazione dei dati (DDRAM) memorizza i dati di visualizzazione rappresentati in codici di caratteri a 8 bit.
L'area nella DDRAM che non viene utilizzata per la visualizzazione può essere utilizzata come RAM dei dati generali; quindi qualunque cosa inviate sulla DDRAM viene effettivamente visualizzata sul display LCD.
Per LCD come 1x16, sono visibili solo 16 caratteri, quindi tutto ciò che viene scritto dopo i 16 caratteri si trova in DDRAM ma non è visibile all'utente.

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	← Character position (dec.)
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	← Row0 DDRAM address (hex)
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53	← Row1 DDRAM address (hex)
14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27	← Row2 DDRAM address (hex)
54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61	62	63	64	65	66	67	← Row3 DDRAM address (hex)

3.1.2 CGRAM

Come si evince dal nome, l'area CGRAM viene utilizzata per creare caratteri personalizzati nell'LCD.
Nella RAM del generatore di caratteri, l'utente può riscrivere i modelli di caratteri tramite programma. Per 5 x 8 punti, è possibile scrivere otto modelli di caratteri e per 5 x 10 punti, è possibile scrivere quattro modelli di caratteri.

3.1.3 Busy Flag (BF)

Il Busy Flag è un flag indicatore di stato per LCD; quando inviamo un comando o dati all'LCD per l'elaborazione, questo flag viene impostato (cioè BF = 1) e non appena l'istruzione viene eseguita con successo questo flag viene cancellato (BF = 0) e questo è utile nella produzione e nella quantità esatta di ritardo per l'elaborazione LCD.
Per leggere l'indicatore di occupato, è necessario soddisfare la condizione RS = 0 e R / W = 1 e l'MSB del bus dati LCD (D7) funge da indicatore di occupato. Quando BF = 1 significa che l'LCD è occupato e non accetterà il comando o i dati successivi e BF = 0 significa che l'LCD è pronto per l'elaborazione del comando o dei dati successivi.

3.1.4 Inizializzazione

Prima di utilizzare l'LCD per scopi di visualizzazione, l'LCD deve essere inizializzato dal circuito di ripristino interno o dall'invio di una serie di comandi per inizializzare l'LCD. È l'utente che deve decidere se un LCD deve essere inizializzato tramite istruzioni o tramite un circuito di ripristino interno. discuteremo entrambi i modi di inizializzazione uno per uno.

3.2 Mandare i comandi

Per inviare comandi è sufficiente selezionare il registro dei comandi; tutto è uguale a quello che abbiamo fatto nella routine di inizializzazione.
Possiamo però riassumere i passaggi comuni per metterli in una singola subroutine, di seguito sono riportati i passaggi:

- Spostare i dati sulla porta LCD;
- selezionare il registro dei comandi;
- selezionare l'operazione di scrittura;

- inviare il segnale di abilitazione;
- attendere che LCD elabori il comando.

4 Funzionamento

Il programma una volta avviato presenta una schermata di home con le informazioni riguardanti le azioni che si possono eseguire; le azioni sono: start, stop, pausa e parziale.
Una volta premuto il pulsante di start il cronometro si avvia, quando si preme il pulsante di pausa il cronometro si mette in pausa mentre con lo stop il cronometro si arresta e riparte dalla home.
Il pulsante del parziale permette di stampare a video i minuti e i secondi attuali del timer solo quando il timer è avviato altrimenti non fa nulla (o, in questo momento, fa una QUIT).

5 Pseudocodice

5.1 Logica di progetto

5.1.1 Inizio

In questa fase, nella prima per essere precisi, di progetto abbiamo sfruttato le funzionalità del registro GPLEV0 che si occupa di rilevare i livelli delle GPIO da 0 a 31, dunque dell'intera Raspberry.

```
: START WHILE(TRUE) {
    : CONTINUA WHILE(GPIO6 = 0V AND GPIO5 >
       0V) {
        COUNTER @ SCRIVISULCD
        COUNTER++
    : PARTIAL IF(GPIO0 > 0V AND COUNTERP1 ==
       0) {
           INCREMENTA COUNTERP1
           MOSTRA PARZIALE
        }
     }
: PAUSA WHILE(GPIO6 > 0V AND GPIO5 > 0V) {
        PAUSA
     }
```

```
: STOP     WHILE(GPIO5 == 0V) {
         IF(COUNTER @ != 0) {
            0 COUNTER !
         }
      }
}
```

```
: MAIN WHILE(TRUE) {
   : START WHILE(GPIO5 > 0) {
      WHILE(GPIO6==0 AND GPIO0==0) {
         COUNTER @ SCRIVISULCD
         COUNTER++
    : PARTIAL  IF(GPIO0 > 0V AND COUNTERP1
       == 0) {
            INCREMENTA COUNTERP1
            MOSTRA PARZIALE
         }
      }
   }
   : PAUSA WHILE(GPIO0 > 0) {
     WHILE(GPIO5==0 AND GPIO6==0) {
     }
   }
   : STOP WHILE(GPIO6 > 0) {
     WHILE(GPIO5 == 0) {
        IF(COUNTER @ != 0) {
           0 COUNTER !
        }
     }
   }
}
```

La logica, in tal caso, è stata organizzata mediante dei cicli while costantemente in ascolto per rilevare quelle che possano essere le variazioni di stato delle GPIO.

5.1.2 Evoluzione

Successivamente, piuttosto che utilizzare il registro GPLEV0 abbiamo analizzato i registri GPAFEN0, GPAREN0 e GPEDS0.
In particolare i nomi sono degli acronimi, ossia: GPIO Pin Async. Falling Edge Detect 0, GPIO Pin Async. Rising Edge Detect 0 e GPIO Pin Event Detect Status 0; questi sono tutti registri a 32 bit.

```
: MAIN WHILE(TRUE) {
   : START WHILE(GPEDS0 @ 20 =) {
      COUNTER @ SCRIVISULCD
      COUNTER++

   : PARTIAL  IF(GPEDS0 @ 2 =) {
            INCREMENTA COUNTERP1
            MOSTRA PARZIALE
         }2 GPEDS0 !
   } 20 GPEDS0 !

   : PAUSA WHILE(GPEDS0 @ 1 =) {
   } 1 GPEDS0 !
```

```
   : STOP WHILE(GPEDS0 @ 40 =) {
      IF(COUNTER @ != 0) {
         0 COUNTER !
      }
   } 40 GPEDS0 !
}
```

Il registro GPEDS0 deve essere settato con la somma dei bit (in esadecimale) delle GPIO delle quali vogliamo controllare gli eventi, nel nostro caso GPIO0, GPIO1, GPIO5, GPIO6, quindi avremo 00110011 che equivale a 99 in decimale o 63 in esadecimale.
Dunque, il primo registro (GPAFEN0) sta in ascolto dei fronti di onda in discesa, mentre il secondo di quelli in salita (GPAREN0).
Noi, in questo progetto abbiamo utilizzato solo il registro GPAREN0.

5.1.3 Finale

Una volta utilizzato il registro GPEDS0 piuttosto che utilizzare diversi WHILE abbiamo utilizzato degli IF poiché i primi oltre ad essere bloccanti erano poco efficienti.

```
WHILE(TRUE){
    IF(GPEDS==1){
        1 GPEDS !
    }
    IF(GPEDS ==2){
        2 GPEDS !
    }
    IF(GPEDS==32){
        START
            32 GPEDS !
    }
    IF(GPEDS ==64){
        64 GPEDS !
    }
}

: STOP QUIT ;
```

```
: PAUSA ;
: PARTIAL STAMPA ;
: STOPPAUSA IF(GPEDS PAUSA = ) PAUSA ELSE STOP ;
: START BEGIN (FINO A STOP O PAUSA) WHILE COUNTER REPEAT ;
```

A questo punto, stabilita una logica generale e le parole che compongono il programma, abbiamo continuato nella realizzazione modificando via via il codice fino a raggiungere lo stato finale riportato nelle sezioni sotto.

6 Timer: approcci

6.1 Metodo naïve

Il metodo che abbiamo utilizzato per i primi esperimenti era quello di mettere dei delay in un ciclo contatore in modo da attendere un tempo che fosse il più simile possibile ad un secondo ma, ovviamente, la strategia era solo di approccio e non definitiva in quanto mancava decisamente di precisione.

6.2 Registro CLO

```
HEX

3F003004 CONSTANT CLO
VARIABLE COMPO
F4240 CONSTANT SEC
VARIABLE COUNTER
0 COMPO !
0 COUNTER !
: INC CLO @ SEC + COMPO ! ;
: DECCOUNT COUNTER @ 1 - COUNTER ! ;
: SLEEPS HEX INC BEGIN CLO @ COMPO @ < WHILE REPEAT CR ." Passato 1 sec " DROP DECIMAL ;
DECIMAL : CICLOCONT COUNTER ! begin CR COUNTER @ U. SLEEPS DECCOUNT COUNTER @ 0 = until CR
    CR ." fine " DROP ;
```

Per quanto riguarda la gestione del tempo abbiamo utilizzato il registro a 32 bit CLO.
In particolare, definendo la costante di un secondo la logica è che viene registrato il valore di CLO, a

questo aggiungiamo 1 secondo e quando il valore attuale di CLO sarà uguale a CLO+1 secondo (CLO+F4240) allora sarà passato un secondo e potremo di nuovo prendere quel valore come riferimento.

7 Codice

7.1 start.f

```
: '\n' 10 ;
: BL 32 ;
: ':' [ CHAR : ] LITERAL ;
: ';' [ CHAR ; ] LITERAL ;
: '(' [ CHAR ( ] LITERAL ;
: ')' [ CHAR ) ] LITERAL ;
: '"' [ CHAR " ] LITERAL ;
: 'A' [ CHAR A ] LITERAL ;
: '0' [ CHAR 0 ] LITERAL ;
: '-' [ CHAR - ] LITERAL ;
: '.' [ CHAR . ] LITERAL ;
: ( IMMEDIATE 1 BEGIN KEY DUP '(' = IF DROP 1+ ELSE ')' = IF 1- THEN THEN DUP 0= UNTIL DROP
    ;
: SPACES ( n -- ) BEGIN DUP 0> WHILE SPACE 1- REPEAT DROP ;
: WITHIN -ROT OVER <= IF > IF TRUE ELSE FALSE THEN ELSE 2DROP FALSE THEN ;
: ALIGNED ( c-addr -- a-addr ) 3 + 3 INVERT AND ;
: ALIGN HERE @ ALIGNED HERE ! ;
: C, HERE @ C! 1 HERE +! ;
: S" IMMEDIATE ( -- addr len )
   STATE @ IF
      ' LITS , HERE @ 0 ,
      BEGIN KEY DUP '"'
               <> WHILE C, REPEAT
      DROP DUP HERE @ SWAP - 4- SWAP ! ALIGN
   ELSE
      HERE @
      BEGIN KEY DUP '"'
               <> WHILE OVER C! 1+ REPEAT
```

```
      DROP HERE @ - HERE @ SWAP
   THEN
;
: ." IMMEDIATE ( -- )
   STATE @ IF
      [COMPILE] S" ' TELL ,
   ELSE
      BEGIN KEY DUP '"' = IF DROP EXIT THEN EMIT AGAIN
   THEN
;
: DICT WORD FIND ;
: VALUE ( n -- ) WORD CREATE DOCOL , ' LIT , , ' EXIT , ;
: TO IMMEDIATE ( n -- )
        DICT >DFA 4+
   STATE @ IF ' LIT , , ' ! , ELSE ! THEN
;
: +TO IMMEDIATE
        DICT >DFA 4+
   STATE @ IF ' LIT , , ' +! , ELSE +! THEN
;
: ID. 4+ COUNT F_LENMASK AND BEGIN DUP 0> WHILE SWAP COUNT EMIT SWAP 1- REPEAT 2DROP ;
: ?HIDDEN 4+ C@ F_HIDDEN AND ;
: ?IMMEDIATE 4+ C@ F_IMMED AND ;
: WORDS LATEST @ BEGIN ?DUP WHILE DUP ?HIDDEN NOT IF DUP ID. SPACE THEN @ REPEAT CR ;
: FORGET DICT DUP @ LATEST ! HERE ! ;
: CFA> LATEST @ BEGIN ?DUP WHILE 2DUP SWAP < IF NIP EXIT THEN @ REPEAT DROP 0 ;
: SEE
   DICT HERE @ LATEST @
   BEGIN 2 PICK OVER <> WHILE NIP DUP @ REPEAT
   DROP SWAP ':' EMIT SPACE DUP ID. SPACE
   DUP ?IMMEDIATE IF ." IMMEDIATE " THEN
   >DFA BEGIN 2DUP
        > WHILE DUP @ CASE
      ' LIT OF 4 + DUP @ . ENDOF
      ' LITS OF [ CHAR S ] LITERAL EMIT '"' EMIT SPACE
        4 + DUP @ SWAP 4 + SWAP 2DUP TELL '"' EMIT SPACE + ALIGNED 4 -
```

```
      ENDOF
      ' OBRANCH OF ." OBRANCH ( " 4 + DUP @ . ." ) " ENDOF
      ' BRANCH OF ." BRANCH ( " 4 + DUP @ . ." ) " ENDOF
      ' ' OF [ CHAR ' ] LITERAL EMIT SPACE 4 + DUP @ CFA> ID. SPACE ENDOF
      ' EXIT OF 2DUP 4 + <> IF ." EXIT " THEN ENDOF
      DUP CFA> ID. SPACE
   ENDCASE 4 + REPEAT
   ';' EMIT CR 2DROP
;
: :NONAME 0 0 CREATE HERE @ DOCOL , ] ;
: ['] IMMEDIATE ' LIT , ;
: EXCEPTION-MARKER RDROP 0 ;
: CATCH ( xt -- exn? ) DSP@ 4+ >R ' EXCEPTION-MARKER 4+ >R EXECUTE ;
: THROW ( n -- ) ?DUP IF
   RSP@ BEGIN DUP R0 4-
        < WHILE DUP @ ' EXCEPTION-MARKER 4+
      = IF 4+ RSP! DUP DUP DUP R> 4- SWAP OVER ! DSP! EXIT THEN
   4+ REPEAT DROP
   CASE
      0 1- OF ." ABORTED" CR ENDOF
      ." UNCAUGHT THROW " DUP . CR
   ENDCASE QUIT THEN
;
: ABORT ( -- ) 0 1- THROW ;
: PRINT-STACK-TRACE
   RSP@ BEGIN DUP R0 4-
        < WHILE DUP @ CASE
      ' EXCEPTION-MARKER 4+ OF ." CATCH ( DSP=" 4+ DUP @ U. ." ) " ENDOF
      DUP CFA> ?DUP IF 2DUP ID. [ CHAR + ] LITERAL EMIT SWAP >DFA 4+ - . THEN
   ENDCASE 4+ REPEAT DROP CR
;
: BINARY ( -- ) 2 BASE ! ;
: OCTAL ( -- ) 8 BASE ! ;
: 2# BASE @ 2 BASE ! WORD NUMBER DROP SWAP BASE ! ;
: 8# BASE @ 8 BASE ! WORD NUMBER DROP SWAP BASE ! ;
: # ( b -- n ) BASE @ SWAP BASE ! WORD NUMBER DROP SWAP BASE ! ;
```

```
: UNUSED ( -- n ) PAD HERE @ - 4/ ;

: PAGE 25 BEGIN 1- CR DUP 0<= UNTIL ;

: JF-HERE  HERE ;
: JF-CREATE  CREATE ;
: JF-FIND  FIND ;
: JF-WORD  WORD ;

: HERE  JF-HERE @ ;
: ALLOT  HERE + JF-HERE ! ;

: [']  ' LIT , ; IMMEDIATE
: '  JF-WORD JF-FIND >CFA ;

: CELL+ 4 + ;

: ALIGNED  3 + 3 INVERT AND ;
: ALIGN JF-HERE @ ALIGNED JF-HERE ! ;

: DOES>CUT  LATEST @ >CFA @ DUP JF-HERE @ > IF JF-HERE ! ;

: CREATE  JF-WORD JF-CREATE DOCREATE , ;
: (DODOES-INT) ALIGN JF-HERE @ LATEST @ >CFA ! DODOES> ['] LIT , LATEST @ >DFA , ;
: (DODOES-COMP) (DODOES-INT) ['] LIT , , ['] FIP! , ;
: DOES>COMP  ['] LIT , HERE 3 CELLS + , ['] (DODOES-COMP) , ['] EXIT , ;
: DOES>INT  (DODOES-INT) LATEST @ HIDDEN ] ;
: DOES>  STATE @ 0= IF DOES>INT ELSE DOES>COMP THEN ; IMMEDIATE

DROP

: 4DROP DROP DROP DROP DROP ;
```

7.2 LCD.f

```
HEX
```

```
\ Dichiarazione dei registri utilizzati
3F200000 CONSTANT GPFSEL0
3F20001C CONSTANT GPSET0
3F200028 CONSTANT GPCLR0
3F003004 CONSTANT CLO

\ Indirizzi delle righe 1 e 2 dell'LCD
80 CONSTANT LCDLN1
C0 CONSTANT LCDLN2

DECIMAL

\ Dichiarazione delle GPIO utilizzate per connettere l'LCD
25 CONSTANT LCDRS
24 CONSTANT LCDE
12 CONSTANT LCD0
4  CONSTANT LCD1
27 CONSTANT LCD2
16 CONSTANT LCD3
23 CONSTANT LCD4
17 CONSTANT LCD5
18 CONSTANT LCD6
22 CONSTANT LCD7

\ Restituisce il valore attuale del registro CLO
( -- clo_value )
: NOW CLO @ ;

\ Setta un ritardo corrispondente al valore presente sullo stack
( delay -- )
: DELAY NOW + BEGIN DUP NOW - 0 <= UNTIL DROP ;

\ Maschera di 32 bit con 3 bit posti ad 1 (7 DECIMAL = 111 BIN ) adibita alla pulizia dei 3
    bit del registro GPFSELn che controlla il pin che ci interessa,
\ shiftando di un multiplo di 3 posti questi bit posti ad 1.
```

```
( n1 -- n2 )
: MASK 3 * 7 SWAP LSHIFT ;


\ Prende in input il numero della GPIO. Effettua un clear dei valori nel registro GPFSEL
    relativo alla GPIO che ci interessa.
\ Il registro corretto GPFSEL è ottenuto prendendo la decina della GPIO passata ed
    utilizzando questo valore come offset partendo da GPFSEL0.
\ L'unità del numero del GPIOn viene utilizzata per spostare la maschera di un multiplo di
    3 bit in modo da pulire i 3 bit del registro che
\ controllano la funzionalità della GPIOn, in modo da non alterare gli altri gruppi che
    corrispondono ad altre GPIO.
\ Sullo stack viene lasciato il valore di GPFSEL con i 3 bit puliti dalla maschera.
( GPIOn -- GPIOn addr_GPFSEL curr_value)
: SET
   DUP
   10 /MOD          ( remainder quot )
   4 *              \ moltiplico il quoziente *4 per ottenere l'offset da GPFSEL0
   GPFSEL0 + DUP @       \ ottengo GPFSELN dove N = quot.
                \ adesso mi trovo all'inizio di GPFSELN
   ROT MASK INVERT AND 2DUP SWAP ! \ abbiamo impostato i bit a zero nei tre bit che ci
       interessano
;


\ Setta la GPIOn in OUTPUT
\ L'unità del numero della GPIOn viene usata per spostarsi nella posizione relativa ai 3
    bit che controllano la GPIOn.
\ Poichè  OUTPUT = 001, spostiamo il valore 1 a sx di un multiplo di 3 bit proporzionale
    all'unità della GPIOn ottenuta con l'operatore MOD.
\ Viene quindi effettuata un'operazione logica OR tra il bit 1 shifato e il valore corrente
    che è presente sullo stack, questo per mantenere inalterati tutti gli altri valori del
    registro GPFSELN. Questo valore finale viene quindi inserito nel registro GPFSELN.
( GPIOn addr_GPFSEL curr_value -- )
: OUTPUT
   ROT
```

```
   10 MOD ( remainder )
   3 * 1 SWAP LSHIFT
   OR SWAP !
;


\ Setta il GPIOn in INPUT
\ L'unità del numero della GPIOn viene usata per spostarsi nella posizione relativa ai 3
    bit che controllano la GPIOn.
\ Poichè  INPUT = 000, spostiamo il valore 1 a sx di un multiplo di 3 bit proporzionale
    all'unità della GPIOn ottenuta con l'operatore MOD.
\ Viene quindi effettuata un'operazione logica di INVERT che trasforma 001 in 110 (mentre
    001 è circondato da 0, 110 è circondato da 1), sempre shiftato di un multiplo di 3 in
    base all'unità del numero della GPIOn. Viene quindi effettuato un'operazione di AND per
    pulire il bit meno significato di quella tripletta di bit (questo perchè  potrebbe
    essere stato posto ad OUTPUT precedentemente e quindi avere solo quel bit posto ad 1.
( GPIOn addr_GPFSEL curr_value -- )
: INPUT
   ROT
   10 MOD ( remainder )
   3 * 1 SWAP LSHIFT
   INVERT AND !
;


\ Prende il valore della GPIOn e lo shifta dello stesso numero di volte, poichè  c'è  una
    corrispondenza bitn <-> GPIOn, settando quel bit nel registro GPSET0 ad 1.
( GPIOn -- )
: ON  1 SWAP LSHIFT GPSET0 ! ;


\ Prende il valore della GPIOn e lo shifta dello stesso numero di volte, poichè  c'è  una
    corrispondenza bitn <-> GPIOn, settando quel bit nel registro GPCLR0 ad 1.
( GPIOn -- )
: OFF   1 SWAP LSHIFT GPCLR0 ! ;
```

```
\ Effettua il set dei pin dell'LCD ad OUTPUT
( -- )
: LCD-SETUP
   LCDRS SET OUTPUT
   LCDE SET OUTPUT
   LCD0 SET OUTPUT
   LCD1 SET OUTPUT
   LCD2 SET OUTPUT
   LCD3 SET OUTPUT
   LCD4 SET OUTPUT
   LCD5 SET OUTPUT
   LCD6 SET OUTPUT
   LCD7 SET OUTPUT
;

\ Abilita l'enable
( -- )
: LCDEON LCDE ON ;

\ Disabilita l'enable
( -- )
: LCDEOFF LCDE OFF ;


HEX

\ Controlla il bit n2 del valore n1 restituendo TRUE(-1) se quel bit è diverso da zero,
    quindi quel bit è posto ad 1, FALSE(0) se invece è uguale zero, quindi quel bit è posto
    a zero.
( n1 n2 -- flag )
: CHECKBIT AND 0<> ;
```

```
\ Controlla il flag. Se è TRUE(-1), setta il pin corrispondente ad ON, se è FALSE(0), setta
    il pin corrispondente ad OFF.
( flag pin -- )
: ?LCDSET SWAP IF ON ELSE OFF THEN ;


\ Prende in input dei valori esadecimali che, trasformati in binario, corrispondono alla
    sequenza di bit corrispondente ad una funzione che vogliamo passare all'LCD.
\ Scrive in modalità 8 bit.
\ Se RS=1 viene mandato un comando (n<0x100). Viceversa, vengono mandati dati.
\ Ogni input è rappresentato da un valore esadecimale che corrisponde al settaggio di bit
    ad 1 o a 0.
\ Questi bit nel comando vengono controllati uno ad uno.
\ Viene effettuato un controllo sul bit dell'input relativo al pin, per vedere se è 0 o 1.
    In base al suo valore, si mette in HIGH se il bit è 1, in LOW se è 0.
( n -- )
: LCDWRITE
   DUP 100 CHECKBIT LCDRS ?LCDSET LCDEON
   DUP 80 CHECKBIT LCD7 ?LCDSET
   DUP 40 CHECKBIT LCD6 ?LCDSET
   DUP 20 CHECKBIT LCD5 ?LCDSET
   DUP 10 CHECKBIT LCD4 ?LCDSET
   DUP 08 CHECKBIT LCD3 ?LCDSET
   DUP 04 CHECKBIT LCD2 ?LCDSET
   DUP 02 CHECKBIT LCD1 ?LCDSET
   DUP 01 CHECKBIT LCD0 ?LCDSET
   DROP LCDEOFF
;


\ Pulisce il display scrivendo il carattere SPAZIO su tutta la DDRAM e riportando il
    puntantore all'inizio.
( -- )
: LCDCLR 1 LCDWRITE ;
```

```
\ Effettua l'inizializzazione dell'LCD settando le impostazioni desiderate.
( -- )
: LCD-INIT
   LCD-SETUP
   \ Function Set: 8-bit, 2 Line, 5x8 Dots
   \ Il valore 38 è il valore in HEX che corrisponde in binario ad una configurazione della
       Function Set suddetta
   \ In binario 38 = 0011 1000, dove:
   \ - i bit alla posizione 0 e 1 sono arbitrari: la loro scelta non influisce
   \ - il bit 2 (F) rappresenta il bit di controllo del formato del display. LOW=5x8,
       HIGH=5x11
   \ - il bit 3 (N) rappresenta il bit di controllo per decidere il °n di linee.LOW=1,
       HIGH=2
   \ - il bit 4 (DL) rappresenta la modalità a 4 o 8 bit. LOW=4-bit, HIGH=8-bit.
   \ - il bit 5 è 1 di default.
   38 LCDWRITE
   500 DELAY

   \ Entry Mode Set
   \ Setta la direzione di movimento del cursore e abilita/disabilita il display.
   \ Il valore 6 è il valore in HEX che corrisponde in binario ad una configurazione della
       Entry Mode Set
   \ In binario 6 = 0110, dove:
   \ - bit 0 (S): shifta l'intero display. LOW=NoShift, HIGH=Shift
   \ - bit 1 (I/D): incrementa/decrementa l'indirizzo della DDRAM e sposta il cursore.
       LOW=MoveToSx-Decrement, HIGH=MoveToDx-Increment
   \ - il bit 2 è 1 di default.
   6 LCDWRITE
   500 DELAY

   \ Display ON/OFF Control
   \ Setta alcune modalità visive del display
   \ Il valore C è il valore in HEX che corrisponde in binario ad una configurazione del
       Display ON/OFF Control
   \ In binario C = 1100, dove:
   \ - bit 0 (B): Blink del cursore. LOW=NoBlink, HIGH=BlinkSet
```

```
   \ - bit 1 (C): Cursore presente o no. LOW=NoCursor, HIGH=CursorSet
   \ - bit 2 (D): Display ON/OFF. LOW=OFF, HIGH=ON
   \ - bit 3: 1 per default.
   C LCDWRITE
   500 DELAY

   LCDCLR
;


DECIMAL


\ Stampa un numero sull'LCD.
\ Il numero viene scritto partendo dall'unità e scrivendo verso sx. Questo è possibile
    grazie alla ripetizione dell'operatore /MOD.
\ Il numero 304 in DECIMAL corrisponde a 130 in HEX = 1 0011 0000, che corrisponde al
    comando di scrittura nell'LCD.
\ In particolare, i 5 bit ùpi significativi vengono lasciati sempre inalterati poichè
    quale numero viene scritto dipende dai 4 bit meno significativi.
\ (Basta guardare il datasheet per rendersi conto che il primo 1 indica la scrittura in
    DDRAM, che 0011 è comune a tutti i numeri da 0 a 9 e che
\ gli ultimi 4 bit corrispondono al valore binario effettivo del numero).
( n -- )
: LCDNTYPE BEGIN 10 /MOD SWAP 304 + LCDWRITE DUP 0= UNTIL DROP ;


HEX


\ Stampa una stringa sull'LCD.
\ Con c@ preleviamo il carattere all'interno dell'indirizzo della stringa che, sommato a
    100 HEX, dà 1xx, dove xx è il valore ASCII del carattere
\ da scrivere, dando così il comando Write data to RAM all'LCD.
( addr_string lenght_string -- )
: LCDSTYPE OVER + SWAP BEGIN 2DUP <> WHILE DUP c@ 100 + LCDWRITE 1+ REPEAT 2DROP ;
```

```
\ n=0 restituisce l'indirizzo della prima linea dell'LCD (LCDLN1), altrimenti quello della
    seconda linea (LCDLN2)
( n -- addr_line )
: LCDLN 0 = IF LCDLN1 ELSE LCDLN2 THEN ;


\ Chiama la Set DDRAM address mandando all'LCD il comando 8x, dove x è l'offset (80 inizio
    linea, 81 spostati di una cella, e così via...). Setta l'indirizzo della DDRAM da cui
    cominciare a scrivere.
\ IndirizzoLinea + Offset = Indirizzo di partenza per la scrittura -> LCDWRITE manda il
    comando.
( n -- )
: LCDLN! LCDLN + LCDWRITE ;


\ Manda il comando all'LCD per far spostare il cursore e il puntatore alla DDRAM da sx a dx
    (incrementando il valore della DDRAM)
( -- )
: LCDCURL>R 6 LCDWRITE ;


\ Manda il comando all'LCD per far spostare il cursore e il puntatore alla DDRAM da dx a sx
    (decrementando il valore della DDRAM)
( -- )
: LCDCURL<R 4 LCDWRITE ;


\ Stampa un numero nella linea (0=linea 1, !0=linea 2) spostato dal lato sx dell'LCD di un
    certo offset.
\ In base alla linea, lascia sullo stack l'indirizzo della linea desiderata + offset.
( number offset line -- )
: LCDNUMBER LCDLN! LCDCURL<R LCDNTYPE ;
```

```
\ Stampa una stringa nella linea (0=linea 1, !0=linea 2) spostata dal lato sx dell'LCD di
    un certo offset.
\ In base alla linea, lascia sullo stack l'indirizzo della linea desiderata + offset.
( addr_string lenght_string offset line -- )
: LCDSTRING LCDLN! LCDCURL>R LCDSTYPE ;


LCD-INIT
```

7.3 HDMI.f

```
HEX

\ Dichiarazione dei colori in esadecimale (formato ARGB)
00FFFFFF CONSTANT WHITE
00000000 CONSTANT BLACK
00FF0000 CONSTANT RED
00FFFF00 CONSTANT YELLOW

\ Dichiarazione del base address del framebuffer
3E8FA000 CONSTANT FRAMEBUFFER

VARIABLE DIM

\ Contatore utilizzato nei cicli per disegnare le linee orizzontali
VARIABLE COUNTERH
: RESETCOUNTERH 0 COUNTERH ! ;
: +COUNTERH COUNTERH @ 1 + COUNTERH ! ;
RESETCOUNTERH

\ Contatore utilizzato nei cicli per tenere traccia del numero di riga corrente
VARIABLE NLINE
: RESETNLINE 1 NLINE ! ;
: +NLINE NLINE @ 1 + NLINE ! ;
RESETNLINE
```

```
\ Restituisce l'indirizzo del punto centrale dello schermo. Coordinata ((larghezza-1)/2,
    (altezza-1)/2)
( -- addr )
: CENTER FRAMEBUFFER 200 4 * + 180 1000 * + ;

\ Colora, con il colore presente sullo stack, il pixel corrispondente all'indirizzo
    presente sullo stack,
\ dopodiché punta al pixel a destra
( color addr -- color addr_col+1 )
: RIGHT 2DUP ! 4 + ;

\ Colora, con il colore presente sullo stack, il pixel corrispondente all'indirizzo
    presente sullo stack,
\ dopodiché punta al pixel in basso
( color addr -- color addr_row+1 )
: DOWN 2DUP ! 1000 + ;

\ Colora, con il colore presente sullo stack, il pixel corrispondente all'indirizzo
    presente sullo stack,
\ dopodiché punta al pixel a sinistra
( color addr -- color addr_col-1 )
: LEFT 2DUP ! 4 - ;

\ Ripristina il valore di partenza dell'indirizzo a seguito di COUNTERH * 4 spostamenti a
    destra
( addr_endline_right -- addr )
: RIGHTRESET COUNTERH @ 4 * - ;

\ Ripristina il valore di partenza dell'indirizzo a seguito di COUNTERH * 4 spostamenti a
    sinistra
( addr_endline_left -- addr )
: LEFTRESET COUNTERH @ 4 * + ;


\ Disegna una linea verso destra di dimensione pari a 48 pixel
```

```
: RIGHTDRAW
   BEGIN COUNTERH @ DIM @ < WHILE +COUNTERH RIGHT REPEAT RIGHTRESET RESETCOUNTERH ;
\ Disegna una linea verso sinistra di dimensione pari a 48 pixel
: LEFTDRAW
   BEGIN COUNTERH @ DIM @ < WHILE +COUNTERH LEFT REPEAT LEFTRESET RESETCOUNTERH ;

\ Disegna il simbolo di pausa (due pipe distanziate)
: DRAWPAUSE
   30 DIM !
   \ Disegna la prima linea della seconda pipe
   WHITE CENTER RIGHTDRAW

   \ Disegna la prima linea della prima pipe, spostandosi di 32 pixel a sinistra
   WHITE CENTER 80 - LEFTDRAW

   \ Ciclo che disegna due pipe, distanziate, di altezza 105 pixel
   BEGIN NLINE @ 70 <
   WHILE
      \ Disegna l'n-esima linea della prima pipe
      DOWN LEFTDRAW
      2SWAP

      \ Disegna l'n-esima linea della seconda pipe
      DOWN RIGHTDRAW
      2SWAP

      +NLINE
   REPEAT
   4DROP RESETNLINE
;

\ Disegna o il simbolo di stop o pulisce la porzione di schermo su cui disegnamo.
\ Partendo da CENTER-32px-48px, quindi CENTER-80px, e poichè  ogni spostamento di 1px su
    una riga
\ vale 4, abbiamo 320 in dec e cioè  140 in hex
: DRAWSQUARE
```

```
   80 DIM !
   CENTER 140 - RIGHTDRAW

   \ Ciclo che disegna un simbolo di stop di altezza 105 pixel
   BEGIN NLINE @ 70 <
   WHILE
      DOWN RIGHTDRAW
      +NLINE
   REPEAT
   2DROP RESETNLINE
;

\ Disegna il simbolo di start.
: DRAWSTART
   RED CENTER 80 -
   BEGIN NLINE @ 70 <=
   WHILE
      \ Permette di rappresentare le linee del triangolo superiore del simbolo start
      NLINE @ 37 <= IF
         NLINE @ DIM !
      \ Permette di rappresentare le linee del triangolo inferiore del simbolo start
      ELSE
         70 NLINE @ - DIM !
      THEN
         \ Disegna una linea verso destra di dimensione variabile e dipendente dal numero di
             riga     \ memorizzato in NLINE.
         DOWN RIGHTDRAW
         +NLINE
   REPEAT
   2DROP RESETNLINE
;

: DRAWSTOP YELLOW DRAWSQUARE ;
: CLEAN BLACK DRAWSQUARE ;
```

7.4 Main.f

```
\ Ordine di inclusione dei file
\ start.f
\ lcd.f
\ HDMI.f
\ main.f

HEX

\ Dichiarazione dell'indirizzo del registro GPEDS0 come costante
3F200040 CONSTANT GPEDS0

\ Dichiarazione dell'indirizzo del registro GPAREN0 come costante
3F20007C CONSTANT GPAREN0

\ Dichiarazione della costante che indica un secondo e che ha valore 1 000 000 usec in
    decimale o
\ F4240 in hex
F4240 CONSTANT SEC



\ Variabile che memorizza il valore del tempo da visualizzare su lcd
VARIABLE COUNTER

\ Variabile che memorizza il valore attuale del CLO + 1 secondo
VARIABLE COMP0

\ Contatore circolare per i parziali
VARIABLE COUNTERP

\ Flag che permette di stabilire se siamo passati o meno dallo stop
VARIABLE FLAG

\ Flag per disegnare lo start sull'hdmi
VARIABLE DSFLAG
```

```
\ Inizializzazione delle variabili utilizzate
0 COMPO !
0 COUNTER !
0 COUNTERP !
1 FLAG !
1 DSFLAG !

\ Rendiamo sensibili ai fronti di salita le GPIO0, GPIO1, GPIO5, GPIO6, a cui sono
    associati i pulsanti, quindi
\ avremo 0110 0011 che equivale a 99 in decimale o 0x63 in esadecimale
: GPAREN! 63 GPAREN0 ! ;

\ Stampa sull'LCD la home con le possibili azioni
\ ( -- )
: PRINTHOME S" 1.START" 0 0 LCDSTRING S" 3.PAUSE" 9 0 LCDSTRING S" 2.STOP" 0 1 LCDSTRING S"
    4.PART." 9 1 LCDSTRING ;

\ Queste word ci permettono di stampare i secondi, i minuti e le ore ed eventualmente gli
    zero ùpi significativi del \ cronometro
\ (sec -- )
DECIMAL : PRINTSEC DUP 15 0 LCDNUMBER 10 < IF 0 14 0 LCDNUMBER THEN ;
\ (min -- )
DECIMAL : PRINTMIN DUP 12 0 LCDNUMBER 10 < IF 0 11 0 LCDNUMBER THEN ;
\ (hour -- )
DECIMAL : PRINTHOUR DUP 9 0 LCDNUMBER 10 < IF 0 8 0 LCDNUMBER THEN ;


\ Queste word ci permettono di stampare i secondi, i minuti ed eventualmente gli zero ùpi
    significativi dei parziali
\ (sec -- )
DECIMAL : PRINTSECPART DUP 4 COUNTERP @ LCDNUMBER 10 < IF 0 3 COUNTERP @ LCDNUMBER THEN ;
\ (min -- )
DECIMAL : PRINTMINPART DUP 1 COUNTERP @ LCDNUMBER 10 < IF 0 0 COUNTERP @ LCDNUMBER THEN ;

\ La Mod Swap Word restituisce il MOD 60 di un numero passato, in particolare quello del
    cronometro, il cui valore
```

```
\ è inizialmente espresso in secondi. La MSW quindi pone sullo stack il resto e il quoto
    della divisione per 60
\ ed effettua successivamente uno swap.
\ (n1 -- n3 n2)
: MSW 60 /MOD SWAP ;

\ Stampiamo nell'ordine prima i secondi, poi i minuti, poi le ore e infine le colon del
    cronometro
\ ( -- )
DECIMAL : PRINTCOUNT COUNTER @ MSW PRINTSEC MSW PRINTMIN PRINTHOUR S" :" 13 0 LCDSTRING S"
    :" 10 0 LCDSTRING DECIMAL ;

\ Memorizza il valore attuale del CLO + 1 secondo in COMP0
: INC NOW SEC + COMP0 ! ;

\ Stampiamo nell'ordine prima i secondi, poi i minuti e infine le colon dei parziali
\ ( -- )
DECIMAL : PRINTPARTIAL COUNTER @ MSW PRINTSECPART MSW PRINTMINPART S" :" 2 COUNTERP @
    LCDSTRING DECIMAL ;

\ Questa word azzera il timer quando abbiamo premuto stop
: CHECKSTOP COUNTER @ 0 <> IF 0 COUNTER ! THEN ;

\ Segnala ogni qual volta è passato un secondo confrontando CLO con COMP0
: SLEEPS HEX INC BEGIN NOW COMP0 @ < WHILE REPEAT CR ." Passato 1 sec " DROP DECIMAL ;

: INCCOUNT COUNTER @ 1 + COUNTER ! ;

\ Aggiorna il valore della riga su cui stampare il parziale corrente
: LINEA COUNTERP @ 2 MOD COUNTERP ! ;

\ Si occupa di stampare il valore del contatore, di attendere un secondo e successivamente
    di incrementare
\ il valore di COUNTER
: CONDCOUNT PRINTCOUNT SLEEPS INCCOUNT ;
```

```
\ Quando si preme il pulsante del parziale viene stampato a video il parziale.
\ Controlla se è stato premuto il pulsante START collegato alla GPIO5 (32) e,
    successivamente, il pulsante
\ per il parziale collegato alla GPIO1 (2)
: PARTIAL GPEDS0 @ 34 = IF LINEA PRINTPARTIAL COUNTERP @ 1 + COUNTERP ! THEN 2 GPEDS0 ! ;

\ Set e reset delle flag
: SETFLAG 1 FLAG ! ;
: RESETFLAG 0 FLAG ! ;
: SETDSFLAG 1 DSFLAG ! ;
: RESETDSFLAG 0 DSFLAG ! ;

\ Stampa la home se flag = 1, quindi all'avvio di MAIN ovvero se siamo passati dallo stop
\ (così diversifichiamo dalla pausa)
: HOME
   FLAG @ 1 = IF PRINTHOME THEN
;

: START
   BEGIN
   \ Controlla se è stato premuto il pulsante START collegato alla GPIO5
   GPEDS0 @ 32 = WHILE

      \ Permette di effettuare la pulizia dell'hdmi e l'inserimento del simbolo start
      \ una sola volta
      DSFLAG @ 1 = IF CLEAN DRAWSTART RESETDSFLAG THEN

      \ Permette di effettuare la pulizia dell'LCD solo se il programma è appena stato
          avviato
      \ o se si proviene dallo stop
      FLAG @ 1 = IF LCDCLR SLEEPS RESETFLAG THEN

      CONDCOUNT

      \ In questo modo "rimaniamo in ascolto" nel caso in cui l'utente dovesse premere il
          pulsante
```

```
      \ del parziale
      PARTIAL
   REPEAT
   \ Resettiamo il valore di GPEDS0
   32 GPEDS0 !
;

: PAUSA
   \ Controlla se è stato premuto il pulsante PAUSA collegato alla GPIO0
   GPEDS0 @ 1 = IF
      \ Clean dello schermo HDMI
      CLEAN
      \ Disegna il simbolo di pausa sull'HDMI
      DRAWPAUSE
      \ Indica che dopo la pausa si vuole ristampare il simbolo di start
      SETDSFLAG
   THEN
   \ Resettiamo il valore di GPEDS0
   1 GPEDS0 !
;

: STOP
   \ Controlla se è stato premuto il pulsante STOP collegato alla GPIO6
   GPEDS0 @ 64 = IF
      CHECKSTOP
      \ Indica che proveniamo dallo stop e che quindi la prossima cosa da visualizzare è la
          home
      SETFLAG
      \ Clear dell'LCD
      LCDCLR
      \ Clean dello schermo HDMI
      CLEAN
      \ Disegna lo stop sull'HDMI
      DRAWSTOP
      \ Indica che dopo lo stop si vuole ristampare il simbolo di start
      SETDSFLAG
```

```
   THEN
   \ Resettiamo il valore di GPEDS0
   64 GPEDS0 !
;

\ La quit e' una funzionalita' aggiuntiva.
\ Questo comando ci permette di non far nulla quando premiamo il pulsante del parziale
\ mentre il cronometro non è avviato (con la quit presente usciamo dal ciclo infinito)
: DISABLEPART GPEDS0 @ 2 = IF 2 GPEDS0 ! QUIT THEN ;

\ Ciclo principale
: MAIN
   BEGIN
   TRUE WHILE
   HOME DISABLEPART START PAUSA STOP
   REPEAT
;


GPAREN!
MAIN
```

8 Conclusione

Allo stato attuale il progetto risulta essere un valido strumento al fine della conoscenza di molti di quelli che sono gli ambiti della programmazione baremetal e, in particolare, del linguaggio Forth. Siamo stati in grado, infatti, di configurare tutto ciò che ci serviva nel dispositivo general purpose (Raspberry) per raggiungere gli scopi che ci siamo prefissati.

8.1 Sviluppi futuri

Per quanto concerne gli sviluppi futuri si potrebbe pensare di implementare una sorta di "autovelox" utilizzando due fotoresistori per avviare/stoppare il cronometro e calcolare quindi la velocità dell'oggetto che li ha attraversati (una mano per esempio); o, ancora, si potrebbe pensare ad un orologio digitale.