Sorgenti del progetto di programmazione assembly a.a. 2006/07

Giacomo Ritucci, Paolo Pennestri

Indice dei file

defs.h

```
NULL = 0xffff

! file descriptor standard
STDIN = 0
STDOUT = 1
SRDERR = 2

! codici syscall
_EXIT = 1
_READ = 3
_WRITE = 4
_OPEN = 5
_CLOSE = 6
_CREAT = 8
_LSEEK = 19
_GETCHAR = 117
_PUTCHAR = 122
_PRINTF = 127
_SPRINTF = 121
_SSCANF = 125

! costanti per _READ
RD = 0
WR = 1
RDWR = 2

! costanti per _LSEEK
SEEK_SET = 0
SEEK_CUR = 1
SEEK_END = 2
```

```
      ! costanti per seekrom
      ROMEND = -1

      ! limiti
      MAXUSERS = 100
      MAXUSRLEN = 16
      PASSLEN = 8
      KEYLEN = 3
      RECORDLEN = 26
      DISPLAYLN = 6
      ! RECORDLEN * MAXUSERS
      MAXROMLEN = 2600

      ! varie
      ROOTUID = 0
```

boot.s

```
! boot.s - inizializzazione del sistema
!
! Giacomo Ritucci, Paolo Pennestri, 30/07/2007

#include "defs.h"

.SECT .TEXT
! Punto di ingresso del sistema.
main:
      ! Inizializzazione file di rom.
      CALL  openrom

      ! Caricamento file di rom in memoria e calcolo numero di utenti.
      CALL  loadrom
      CMP   AX, -1
      JE    main

      CALL creatlog

      ! Richiesta badge.
1:    PUSH  msgtitle
      CALL  askbadge
      ADD   SP, 2

      ! Richiesta password.
      PUSH  password
      PUSH  msgtitle
      CALL  askpass
      ADD   SP, 4

      ! Autenticazione.
      PUSH  password
      PUSH  (userid)
      CALL  authusr
      CMP   AX, -1
      JE    1b

      ! Gestione utente / amministratore.
      MOV   AX, (userid)
      CMP   AX, ROOTUID
      JE    3f
      CALL  serveusr
      JMP   1b
```

```
3:    CALL  serveadm
      JMP   1b


! void creatlog (void)
! Crea il file di log.
creatlog:
      PUSH  BP
      MOV   BP, SP

      ! Da specifiche, "all'avvio file di log creato vuoto"
1:    PUSH  0644
      PUSH  logpath
      PUSH  _CREAT
      SYS
      ADD   SP, 6
      CMP   AX, -1
      JNE   8f

      PUSH  callhelp
      PUSH  doorerr
      CALL  showerr
      ADD   SP, 4

      JMP   1b

8:    MOV   (logfd), AX

      MOV   SP, BP
      POP   BP
      RET


.SECT .DATA
logpath:
      .ASCIZ      "porta.log"
msgtitle:
      .ASCIZ      "CONTROLLO ACCESSI"
doorerr:
      .ASCIZ      "ERRORE PORTA"

.SECT .BSS
logfd:
      .SPACE      2

! Variabili usate nell'identificazione dell'utente autenticato via badge.
username:
      .SPACE      MAXUSRLEN+1
password:
      .SPACE      PASSLEN+1
userid:
      .SPACE      2
newusrn:
      .SPACE      MAXUSRLEN+1
newpass:
      .SPACE      PASSLEN+1
delusrn:
      .SPACE      MAXUSRLEN+1
```

adm.s

```
! adm.s - funzione per la gestione dell'amministratore.
!
! Giacomo Ritucci, Paolo Pennestri, 02/08/2007

.SECT .TEXT
! void serveadm (void)
! Presenta il menu amministratore sul display.
! Ritorna quando la scelta e' "Annulla" oppure "Apri porta", altrimenti
! continua a ciclare.
serveadm:
      PUSH  BP
      MOV   BP, SP

      PUSH  4
      PUSH  admmenu
      PUSH  admroute
1:    CALL  shwmenu
      CMP   AX, 2
      JGE   1b
      ADD   SP, 6

      MOV   SP, BP
      POP   BP
      RET

! void usrmng (void)
! Presenta il menu per la gestione degli utenti.
! Ritorna quando la scelta e' "Annulla", altrimenti cicla.
usrmng:
      PUSH  BP
      MOV   BP, SP

      PUSH  5
      PUSH  ummenu
      PUSH  umroute
1:    CALL  shwmenu
      CMP   AX, 0       ! annulla
      JNE   1b
      ADD   SP, 6

      MOV   SP, BP
      POP   BP
      RET


! void usadd (void)
! Aggiunta interattiva di un utente.
! Richiede nome utente e pass da tastiera e li aggiunge alla rom.
usadd:
      PUSH  BP
      MOV   BP, SP

      ! Controllo numero utenti.
      CMP   (numusers), MAXUSERS
      JL    1f

      PUSH  cantadd
      PUSH  errfull
      JMP   8f
```

```
      ! Richiesta nuovo nome utente.
1:    PUSH  newusrn
      PUSH  msgnewus    ! "AGGIUNTA UTENTE"
      CALL  askusrn
      ADD   SP, 4

      ! Controllo lunghezza nome utente.
      PUSH  newusrn
      CALL  strlen
      ADD   SP, 2
      CMP   AX, 0
      JG    1f

      PUSH  usleninf
      PUSH  errlen
      JMP   8f

      ! Nome utente non deve essere gia' in uso.
1:    PUSH  newusrn
      CALL  srchrom
      ADD   SP, 2
      CMP   AX, -1
      JE    2f

      PUSH  cantadd
      PUSH  errused
      JMP   8f

      ! Richiesta nuova password.
2:    PUSH  newpass
      PUSH  msgnewus
      CALL  askpass
      ADD   SP, 4

      ! Controllo lunghezza password.
      PUSH  newpass
      CALL  strlen
      ADD   SP, 2
      CMP   AX, PASSLEN
      JE    3f

      PUSH  psleninf
      PUSH  errlen
      JMP   8f

      ! Inserimento nella rom.
3:    PUSH  newpass
      PUSH  newusrn
      CALL  romusadd
      ADD   SP, 4
      JMP   9f

      ! Stampa l'errore.
8:    CALL  showerr
      ADD   SP, 4

9:    MOV   SP, BP
      POP   BP
      RET
```

```
! void usdel (void)
! Richiede il nome dell'utente da cancellare, lo cerca in romimg e, se
! presente, lo rimuove e salva la rom.
! Se l'utente specificato non esiste (o e' l'admin) stampa una schermata di
! errore.
usdel:
      PUSH  BP
      MOV   BP, SP

      ! Inizializza buffer nome utente da cancellare.
      PUSH  MAXUSRLEN+1
      PUSH  0
      PUSH  delusrn
      CALL  memset
      ADD   SP, 6

      ! Richiesta nome utente.
      PUSH  delusrn
      PUSH  msgdelus
      CALL  askusrn
      ADD   SP, 4

      ! Ricerca nome utente.
      PUSH  delusrn
      CALL  srchrom
      ADD   SP, 2

      ! Se srchrom ritorna -1, l'utente non esiste, se ritorna 0 si sta
      ! tentando di cancellare l'admin; in entrambi i casi e' errore.
      CMP   AX, 0
      JG    1f

      ! Stampa messaggio d'errore e ritorno.
      PUSH  erruser
      PUSH  msgdelus
      CALL  showerr
      ADD   SP, 4
      JMP   9f

      ! Rimozione nome utente dalla rom.
1:    PUSH  AX          ! id utente
      CALL  romusdel
      ADD   SP, 2

9:    MOV   SP, BP
      POP   BP
      RET

! void uslst (void)
! Wrapper per dolst. Inizializza l'iterazione in ordine di inserimento.
uslst:
      PUSH  BP
      MOV   BP, SP

      ! Inizializza iterazione rom.
      PUSH  0
      CALL  inititer
      ADD   SP, 2

      CALL  dolst

      MOV   SP, BP
```

```
      POP   BP
      RET


! void uslstal (void)
! Wrapper per dolst. Inizializza l'iterazione in ordine alfabetico.
uslstal:
      PUSH  BP
      MOV   BP, SP

      ! Inizializza iterazione alfabetica rom.
      PUSH  1
      CALL  inititer
      ADD   SP, 2

      CALL  dolst

      MOV   SP, BP
      POP   BP
      RET


! void dolst (void)
! Mostra il menu per l'iterazione sulla rom e per la cancellazione dell'utente
! correntemente visualizzato.
dolst:
      PUSH  BP
      MOV   BP, SP

      PUSH  BX

      ! Se c'e' solo l'admin, mostra una schermata d'errore.
1:    CMP   (numusers), 1
      JG    2f

      PUSH  nousers
      PUSH  errempty
      CALL  showerr
      ADD   SP, 4
      JMP   9f

      ! Se iterid e' 0 bisogna dare l'avvio all'iterazione chiamando
      ! subito romnext, altrimenti si salta direttamente al menu.
2:    CMP   (iterid), 0
      JNE   3f
      CALL  romnext

      ! Mostra il menu.
3:    PUSH  4
      PUSH  lsmenu
      PUSH  lsroute
      CALL  shwmenu
      ADD   SP, 6

      ! Scelto "annulla", esce dal menu.
      CMP   AX, 0
      JE    9f

      ! Se non ha scelto "elimina" torna subito a mostrare il menu.
      CMP   AX, 2
      JNE   1b
```

```
      ! Altrimenti cancellazione utente e visualizzazione successivo.
      MOV   BX, (iterid)      ! salva in BX l'id da cancellare
      CALL  romnext
      ! Ora iterid contiene il nuovo id.
      ! Se cancelliamo un id utente minore di quello che dobbiamo mostrare,
      ! va decrementato l'id superstite.
      CMP   BX, (iterid)
      JGE   4f
      DEC   (iterid)
4:    PUSH  BX
      CALL  romusdel
      ADD   SP, 2
      JMP   1b

9:    POP   BX

      MOV   SP, BP
      POP   BP
      RET


.SECT .DATA

mtumng:
      .ASCIZ      "GESTIONE UTENTI"
meusadd:
      .ASCIZ      "1. Aggiunta                "
meusdel:
      .ASCIZ      "2. Rimozione               "
meuslst:
      .ASCIZ      "3. Elenco                  "
meuslsal:
      .ASCIZ      "4. Elenco alfabetico       "

umroute:
      .WORD noop, usadd, usdel, uslst, uslstal
ummenu:
      .WORD mecancl, meuslsal, meuslst, meusdel, meusadd, mtumng

mtlst:
      .ASCIZ      "ELENCO UTENTI"
meusnxt:
      .ASCIZ      "1. Successivo              "
meusdel2:
      .ASCIZ      "2. Elimina utente          "

lsroute:
      .WORD noop, romnext, noop, noop
lsmenu:
      .WORD itrusrn, mecancl, meusdel2, meusnxt, mtlst

msgnewus:
      .ASCIZ      "AGGIUNTA UTENTE"
msgdelus:
      .ASCIZ      "RIMOZIONE UTENTE"

errfull:
      .ASCIZ      "ROM PIENA"
errused:
      .ASCIZ      "NOME UTENTE IN USO"
cantadd:
      .ASCIZ      "Impossibile aggiungere."
```

```
errempty:
      .ASCIZ      "ROM VUOTA"
nousers:
      .ASCIZ      "Nessun utente presente."

errlen:
      .ASCIZ      "LUNGHEZZA ERRATA"
usleninf:
      .ASCIZ      "da 1 a 16 caratteri."
psleninf:
      .ASCIZ      "8 caratteri."
```

drwscr.s

```
! drwscr.s - routine per la stampa dello schermo
!
! Giacomo Ritucci, Paolo Pennestri, 28/07/2007

      LINELEN = 29           ! 28 spazi piu' terminatore

.SECT .TEXT
! void drwscr (line1, line2, line3, line4, line5, line6)
! Stampa lo schermo e il tastierino numerico.
! Dentro al riquadro dello schermo stampa le stringhe passate come argomento,
! ognuna lunga al massimo 28 caratteri.
drwscr:
      PUSH  BP
      MOV   BP, SP

      ! Salvataggio registri usati.
      PUSH  BX
      PUSH  CX
      PUSH  SI

      ! Scorrimento in alto delle schermate vecchie.
      PUSH  flush
      PUSH  _PRINTF
      SYS
      ADD   SP, 4

      ! Stampa del bordo superiore del display
      PUSH  updwbrd
      PUSH  _PRINTF
      SYS
      ADD   SP, 4

      ! Stampa delle sei righe del display, passate come argomenti.
      ! A ogni iterazione viene preparata una riga di soli spazi, e se
      ! la stringa puntata dall'argomento corrente non e' NULL viene copiata
      ! sopra gli spazi.

      ! Preparazione ciclo.
      MOV   CX, 6       ! contatore
      MOV   SI, 4       ! offset argomenti: 4, 6, 8, ...
      PUSH  blkline           ! riga di soli spazi
      PUSH  line-LINELEN      ! riga da stampare

1:    ! Calcolo prossima riga da gestire.
      POP   DX
      ADD   DX, LINELEN
```

```
        PUSH  DX
        ! Se l'argomento e' -2, va stampata la riga salvata all'invocazione
        ! precedente, salta subito alla printf.
        CMP   (BP)(SI), -2
        JE    2f
        ! Altrimenti la riga e' da modificare: azzerata riempendola di spazi.
        CALL  strcpy
        ! Se l'argomento e' NULL salta subito a stampare la riga di spazi.
        CMP   (BP)(SI), NULL
        JE    2f
        ! Altrimenti stringa non nulla, va copiata nella riga da stampare
        ! usando memcpy perche' non bisogna prendere il terminatore.
        ! Calcolo lunghezza.
        PUSH  (BP)(SI)
        CALL  strlen
        ADD   SP, 2

        ! Centratura testo:
        ! spiazzamento = (spazio disponibile - spazio utilizzato) / 2
        MOV   BX, LINELEN
        SUB   BX, AX
        SHR   BX, 1
        POP   DX
        PUSH  DX
        ADD   BX, DX

        ! Copia della stringa nella riga da stampare
        PUSH  AX          ! lunghezza stringa passata come argomento
        PUSH  (BP)(SI)
        PUSH  BX
        CALL  memcpy
        ADD   SP, 6

        ! Stampa della riga
2:      PUSH  fmtline
        PUSH  _PRINTF
        SYS
        ADD   SP, 4
        ADD   SI, 2
        LOOP  1b

        ADD   SP, 4       ! toglie dallo stack blkline e line

        ! Stampa del bordo inferiore
        PUSH  updwbrd
        PUSH  _PRINTF
        SYS
        ADD   SP, 4

        ! Stampa del tastierino.
        PUSH  keybrd
        PUSH  _PRINTF
        SYS
        ADD   SP, 4

        ! Ripristino registri usati.
        POP   SI
        POP   CX
        POP   BX

        MOV   SP, BP
        POP   BP
```

```
      RET

! void drwmsg (msg, i)
! Ridisegna lo schermo precedente, sostituendo l'i-esima (1...6) riga
! dall'alto con la stringa msg.
drwmsg:
      PUSH  BP
      MOV   BP, SP

      PUSH  CX

      ! Preparazione argomenti per drwscr.
      MOV   CX, DISPLAYLN

1:    CMP   CX, +6(BP)  ! i
      JE    2f
      PUSH  -2
      JMP   3f
2:    PUSH  +4(BP)            ! errmsg
3:    LOOP  1b

      CALL  drwscr
      ADD   SP, 12

9:    POP   CX          ! ripristino

      MOV   SP, BP
      POP   BP
      RET

! void askbadge (*title)
! Stampa una schermata con titolo title e con la richiesta di inserimento del
! badge. Attende la pressione del tasto associato all'evento e usa rdbadge
! salvare il nome utente in username.
! In caso di errore ripresenta la domanda: non ritorna finche' il badge non
! e' stato letto correttamente.
askbadge:
      PUSH  BP
      MOV   BP, SP

      ! Stampa schermo iniziale.
      PUSH  NULL
      PUSH  msgbdg            ! "inserire il badge"
      PUSH  NULL
      PUSH  NULL
      PUSH  +4(BP)            ! title
      PUSH  NULL
      CALL  drwscr
      ADD   SP, 12

      ! Lettura nome utente.
1:    CALL  rdbadge
      CMP   AX, -1            ! badge male inserito
      ! In caso di errore la stampa di un messaggio esplicativo e' a carico
      ! di rdbadge quindi e' sufficiente ritentare.
      JE    1b

      MOV   SP, BP
      POP   BP
      RET
```

```
! void askpass (*title, *passbuf)
! Stampa una schermata con titolo title richiedente la digitazione di una
! password e ne attende l'inserimento. La password viene memorizzata, con
! tanto di terminatore, nel buffer passbuf specificato, che deve essere lungo
! almeno PASSLEN+1.
askpass:
      PUSH  BP
      MOV   BP, SP

      ! Stampa schermata
      PUSH  NULL
      PUSH  msgpass           ! "digitare password"
      PUSH  NULL
      PUSH  NULL
      PUSH  +4(BP)            ! title
      PUSH  NULL
      CALL  drwscr
      ADD   SP, 12

      ! Lettura password
      PUSH  PASSLEN+1
      PUSH  +6(BP)            ! passbuf
      CALL  readkbd
      ADD   SP, 4

      MOV   SP, BP
      POP   BP
      RET


! void askusrn (*title, *userbuf)
! Stampa una schermata con titolo title richiedente la digitazione di un
! nome utente e ne attende l'inserimento. Il nome utente viene memorizzato,
! con tanto di terminatore, nel buffer userbuf specificato, che deve essere
! lungo almeno MAXUSRLEN+1.
askusrn:
      PUSH  BP
      MOV   BP, SP

      ! Stampa schermata
      PUSH  NULL
      PUSH  msgusrn           ! "digitare nome utente"
      PUSH  NULL
      PUSH  NULL
      PUSH  +4(BP)            ! title
      PUSH  NULL
      CALL  drwscr
      ADD   SP, 12

      ! Lettura nome utente.
      PUSH  MAXUSRLEN+1
      PUSH  +6(BP)            ! userbuf
      CALL  readkbd
      ADD   SP, 4

      MOV   SP, BP
      POP   BP
      RET


! void showerr (*title, *msg)
! Stampa una schermata con titolo e messaggio specificati, attendendo la
```

```
! pressione di un tasto qualsiasi dall'utente.
showerr:
      PUSH  BP
      MOV   BP, SP

      ! Stampa schermata di errore
      PUSH  NULL
      PUSH  msgakey           ! "premere un tasto"
      PUSH  +6(BP)            ! msg
      PUSH  NULL
      PUSH  +4(BP)            ! title
      PUSH  NULL
      CALL  drwscr
      ADD   SP, 12

      CALL  skipln

      MOV   SP, BP
      POP   BP
      RET


.SECT .DATA
msgbdg:
      .ASCIZ      "inserire il badge..."
msgusrn:
      .ASCIZ      "digitare nome utente..."
msgpass:
      .ASCIZ      "digitare la password..."
msgakey:
      .ASCIZ      "premere un tasto..."

updwbrd:
      .ASCIZ      "********************************\n"
blkline:
      .ASCIZ      "                              "     ! 28 spazi + terminatore
fmtline:
      .ASCIZ      "* %s *\n"
keybrd:
      .ASCII      "\n\n"
      .ASCII      "|  1   | 2abc | 3def |\n"
      .ASCII      "| 4ghi | 5jkl | 6mno |\n"
      .ASCII      "| 7pqrs| 8tuv | 9wxyz|\n"
      .ASCII      "|  0_  |\n"
      .ASCIZ      "|   INVIA  | ANNULLA |\n"
flush:
      .ASCII      "\n\n\n\n\n\n\n\n\n\n\n"
      .ASCII      "\n\n\n\n\n\n\n\n\n\n\n"
      .ASCII      "\n\n\n\n\n\n\n\n\n\n\n"
      .ASCII      "\n\n\n\n\n\n\n\n\n\n\n"
      .ASCII      "\n\n\n\n\n\n\n\n\n\n\n"
      .ASCII      "\n\n\n\n\n\n\n\n\n\n\n"
      .ASCII      "\n\n\n\n\n\n\n\n\n\n\n"
      .ASCII      "\n\n\n\n\n\n\n\n\n\n\n"
      .ASCII      "\n\n\n\n\n\n\n\n\n\n\n"
      .ASCIZ      "\n\n\n\n\n\n\n\n\n\n\n"

.SECT .BSS
line:
      .SPACE      LINELEN
      .SPACE      LINELEN
      .SPACE      LINELEN
```

```
      .SPACE      LINELEN
      .SPACE      LINELEN
      .SPACE      LINELEN
```

io.s

```
! io.s - funzioni di I/O
!
! Giacomo Ritucci, Paolo Pennestri, 31/07/2007

      BADGESIM = 'x'

.SECT .TEXT
! int IN_2 (indirizzo)
! In AX ritorna:
! -1 se c'e' un errore hardware
! -2 se il badge e' stato inserito male
! >0 ovvero il carattere letto dal badge
!  0 quando non c'e' altro da leggere
! Se fallisce, DX contiene l'indirizzo del messaggio d'errore, altrimenti il
! contenuto di DX e' indefinito.
IN_2:
      PUSH  BP
      MOV   BP, SP

      PUSH  BX          ! salvataggio

      MOV   AX, -1            ! init valore di ritorno
      MOV   DX, errhw   ! init msg errore

      ! Controllo indirizzo passato come argomento.
      MOV   BX, 0x0
      CMP   BX, +4(BP)
      JNE   9f

      ! Se il fd vale -1, c'e' da iniziare la lettura di un nuovo badge.
      CMP   (badgefd), -1
      JNE   1f

      ! Attesa inserimento badge, simulata con pressione tasto.
      ! Lettura fino al '\n' per evitare che input "abbondanti" influiscano
      ! sulle _GETCHAR successive: viene considerato solo il primo carattere
      ! (quindi 'xfggf\n' viene accettato).
      PUSH  _GETCHAR
      SYS
      ADD   SP, 2
      PUSH  AX                ! salvataggio primo carattere
      CALL  skipln                  ! scarta tutto il resto fino al \n
      POP   AX                ! ripristino primo carattere

      CMPB  AL, BADGESIM            ! AL == 'x'
      JNE   8f

      ! Badge ben inserito, apertura file.
      PUSH  RD
      PUSH  bdgpath
      PUSH  _OPEN
      SYS
      ADD   SP, 6
      CMP   AX, -1
      JE    9f
```

```
      MOV   (badgefd), AX           ! salvataggio fd

      ! Lettura di un carattere dal badge. Generalmente i file hanno un \n
      ! prima dell'EOF, ma non e' certo. Viene controllato prima l'EOF poi
      ! il \n.
1:    PUSH  1
      PUSH  lettore
      PUSH  (badgefd)
      PUSH  _READ
      SYS
      ADD   SP, 8
      CMP   AX, 0             ! controllo EOF
      JE    2f

      MOVB  AL, (lettore)           ! ritorna il carattere letto
      CMPB  AL, '\n'          ! controllo newline
      JE    2f
      JMP   9f

      ! Badge letto completamente, chiusura file e valore di ritorno 0.
2:    MOV   AX, 0
      PUSH  (badgefd)
      PUSH  _CLOSE
      SYS
      ADD   SP, 4
      MOV   (badgefd), -1
      JMP   9f

      ! Errore inserimento badge (non e' stato premuto 'x')
8:    MOV   AX, -2
      MOV   DX, errbadge

9:    POP   BX          ! ripristino

      MOV   SP, BP
      POP   BP
      RET


! Apre la porta
OUT_2:
      PUSH  BP
      MOV   BP, SP

      PUSH  DI          ! salvataggio

      ! Confronto indirizzo dato.
      CMP   +4(BP), 0x3
      JNE   8f

      SUB   SP, 2       ! spazio per lunghezza username

      PUSH  username
      CALL  strlen

      ! Aggiunta di un '\n' alla fine dell'username. Si puo' modificare
      ! l'username perche' dopo l'apertura della porta c'e' il logout,
      ! quindi il contenuto del buffer non viene piu' utilizzato fino al
      ! prossimo accesso.
      MOV   DI, username
      ADD   DI, AX
```

```
      INC   AX          ! ora len conta anche il '\n'

      MOV   -4(BP), AX  ! salvataggio len nello stack

      MOVB  AL, '\n'
      STOSB

      PUSH  (logfd)
      PUSH  _WRITE
      SYS
      ADD   SP, 8

      JMP   9f

8:    MOV   AX, -1

9:    POP   DI          ! ripristino

      MOV   SP, BP
      POP   BP
      RET


! int rdbadge (void)
! Accede al lettore del badge, attende l'inserimento e prova a leggere il nome
! utente, salvandolo nella variabile username.
! Ritorna 0 se tutto ok, -1 se fallisce. In questo caso mostra un messaggio di
! errore.
! SIDE EFFECT:
! - salva il nome utente in username
rdbadge:
      PUSH  BP
      MOV   BP, SP

      PUSH  DI

      ! Inizializzazione buffer username
      PUSH  MAXUSRLEN+1
      PUSH  0
      PUSH  username
      CALL  memset
      ADD   SP, 6

      MOV   DI, username

1:    PUSH  0x0
      CALL  IN_2
      ADD   SP, 2
      CMPB  AL, 0
      JL    8f
      STOSB             ! salvataggio carattere o terminatore
      CMPB  AL, 0
      JG    1b

      ! Ricerca utente nel database
      PUSH  username
      CALL  srchrom
      ADD   SP, 2
      CMP   AX, -1
      JE    8f

      ! L'utente esiste, salva userid e ritorna zero.
```

```
      MOV   (userid), AX
      MOV   AX, 0
      JMP   9f

      ! Stampa del messaggio d'errore. DX viene riempito da IN_2 e contiente
      ! il puntatore al messaggio.
8:    PUSH  4
      PUSH  DX
      CALL  drwmsg
      ADD   SP, 4

      MOV   AX, -1            ! ritorna errore

      ! Ripristino registri usati.
9:    POP   DI

      MOV   SP, BP
      POP   BP
      RET


.SECT .DATA
bdgpath:
      .ASCIZ      "badge.txt"
errbadge:
      .ASCIZ      "Badge male inserito"
erruser:
      .ASCIZ      "Utente inesistente"
errhw:
      .ASCIZ      "Malfunzionamento lettore"
badgefd:
      .WORD -1

.SECT .BSS
lettore:
      .SPACE      1           ! buffer carattere letto dal badge
```

kbrd.s

```
! kbrd.s - tastierino numerico
!
! Giacomo Ritucci, Paolo Pennestri, 29/07/2007

      KBDSTRLN = 20
      RETCHAR = '.'
      CNLCHAR = ','

.SECT .TEXT
! void skipln (void)
! Scarta tutti i caratteri sullo standard input fino al successivo \n
! (compreso).
skipln:
      PUSH  BP
      MOV   BP, SP

      PUSH  _GETCHAR
1:    SYS
      CMPB  AL, '\n'
      JNE   1b

      MOV   SP, BP
```

```
      POP   BP
      RET


! char keypress (void)
! Simula il comportamento di un tastierino alfanumerico. Attende una serie di
! n cifre identiche terminate da un '\n' e ritorna il carattere corrispondente
! a n pressioni del tasto. In caso di errore ritorna -1.
! Es:
! "11111\n" ritorna 1
! "777\n"   ritorna 'q'
! "12\n"    ritorna -1
keypress:
      PUSH  BP
      MOV   BP, SP

      ! Variabili locali:
      ! -2: carattere ritornato da _GETCHAR
      SUB   SP, 2

      PUSH  BX
      PUSH  SI

      ! Lettura primo carattere, deve essere una cifra 0-9
1:    PUSH  _GETCHAR
      SYS
      CMPB  AL, '\n'    ! salto delle righe vuote
      JE    1b
      CMPB  AL, RETCHAR ! carattere speciale: INVIA
      JE    9f
      CMPB  AL, CNLCHAR ! carattere speciale: ANNULLA
      JE    9f
      CMPB  AL, '0'           ! carattere non valido
      JL    7f
      CMPB  AL, '9'           ! carattere non valido
      JG    7f

      ! Salvataggio carattere e conversione a intero.
      MOV   -2(BP), AX
      SUB   AX, 0x30

      ! Posizionamento di BX all'inizio della stringa kbdstr relativa al
      ! tasto premuto: BX = kbdstr + AX * KBDSTRLN
      MOV   BX, KBDSTRLN
      MUL   BX
      MOV   BX, kbdstr
      ADD   BX, AX

      MOV   SI, 0

      ! Lettura dei caratteri successivi, fino al '\n'.
      ! Se un carattere e' diverso da quello letto per primo, errore.
      ! Ad ogni carattere letto, SI = (SI + 1) % KBDSTRLN
2:    SYS
      CMPB  AL, '\n'    ! trovato "a capo", fine stringa
      JE    8f
      CMPB  AL, -2(BP)  ! carattere uguale ai precedenti
      JNE   7f
      INC   SI
      CMP   SI, KBDSTRLN
      JNE   2b
      MOV   SI, 0
```

```
      JMP   2b

7:    MOV   AX, -1            ! errore, ritorna -1
      JMP   9f
8:    MOVB  AH, 0
      MOVB  AL, (BX)(SI)      ! ok, ritorna il carattere individuato

9:    ADD   SP, 2       ! ripristina stack usato da _GETCHAR

      POP   SI
      POP   BX

      MOV   SP, BP
      POP   BP
      RET

! int readkbd (buf, buflen)
! Legge una stringa dal tastierino numerico e la salva nella locazione di
! memoria puntata da buf. Ritorna quando viene premuto il tasto INVIA
! (simulato con il carattere '.').  ! Ritorna il numero di caratteri letti.
readkbd:
      PUSH  BP
      MOV   BP, SP

      ! Salvataggio registri usati.
      PUSH  CX
      PUSH  DI

      ! Azzeramento buffer e lettura caratteri fino a ricezione INVIA o a
      ! riempimento buffer.
1:    MOV   AX, 0       ! terminatore
      MOV   DI, +4(BP)  ! buffer
      MOV   CX, +6(BP)  ! lunghezza buffer
      REP   STOSB       ! azzeramento buffer
      MOV   DI, +4(BP)
      MOV   CX, +6(BP)

      ! Stampa carattere digitato.
2:    PUSH  6
      PUSH  +4(BP)
      CALL  drwmsg
      ADD   SP, 4

      CALL  keypress

      ! Sostituzione codice di INVIO con terminatore.
      CMPB  AL, RETCHAR
      JNE   3f
      CALL  skipln            ! fine input, flush stdin
      MOV   AX, 0

      ! Salvataggio carattere nel buffer.
3:    STOSB
      PUSH  AX
      POP   AX

      ! Ricevuto un ANNULLA, tutto da rifare.
      CMPB  AL, CNLCHAR
      JE    1b

      ! Terminatore o buffer pieno, fine ciclo.
      CMPB  AL, 0
```

```
      LOOPNE     2b

      ! Se s'e' riempito il buffer, sostituzione ultimo carattere con
      ! terminatore.
      MOV   AX, 0
      DEC   DI
      STOSB

      ! Calcolo numero di caratteri letti per valore di ritorno.
4:    MOV   AX, +6(BP)
      SUB   AX, CX

      ! Ripristino registri usati.
      POP   DI
      POP   CX

      MOV   SP, BP
      POP   BP
      RET

! int readchc (n)
! Legge da tastierino un numero che rappresenta la scelta di un menu.
! Non ritorna finche' l'input da tastierino non rappresenta un numero
! compreso tra 0 e n.
readchc:
      PUSH  BP
      MOV   BP, SP

      PUSH  BX          ! salvataggio

      ! Lettura input.
1:    MOV   AX, 0
      PUSH  _GETCHAR
      SYS
      ADD   SP, 2

      ! Scarto di eventuali caratteri fino al '\n'
      MOV   BX, AX            ! salvataggio carattere letto
      CALL  skipln

      ! Conversione a intero e controllo.
      SUB   BX, '0'
      CMP   BX, 0
      JL    8f          ! errore
      CMP   BX, +4(BP)  ! n
      JG    8f          ! errore
      JMP   9f

      ! Errore input: minore di zero oppure maggiore del numero di scelte
      ! del menu. Stampa un messaggio di errore e salta a richiedere l'input
      ! nuovamente.
8:    PUSH  6
      PUSH  errchc
      CALL  drwmsg
      ADD   SP, 4
      JMP   1b

9:    MOV   AX, BX            ! ritorna scelta effettuata
      POP   BX          ! ripristino

      MOV   SP, BP
      POP   BP
```

```
      RET


.SECT .DATA
kbdstr:
      .ASCII      "0_0_0_0_0_0_0_0_0_0_"
      .ASCII      "11111111111111111111"
      .ASCII      "2abc2abc2abc2abc2abc"
      .ASCII      "3def3def3def3def3def"
      .ASCII      "4ghi4ghi4ghi4ghi4ghi"
      .ASCII      "5jkl5jkl5jkl5jkl5jkl"
      .ASCII      "6mno6mno6mno6mno6mno"
      .ASCII      "7pqrs7pqrs7pqrs7pqrs"
      .ASCII      "8tuv8tuv8tuv8tuv8tuv"
      .ASCII      "9wxyz9wxyz9wxyz9wxyz"

errchc:
      .ASCIZ      "Scelta non valida!"

.SECT .BSS
```

menu.s

```
! menu.s - menu utente e amministratore a scelta multipla.
!
! Giacomo Ritucci, Paolo Pennestri, 02/08/2007

.SECT .TEXT
! int shwmenu (routearray, menuarray, n)
! Mostra un menu e attende la scelta dell'utente, quindi esegue la funzione
! associata alla scelta. Al ritorno della funzione, ritorna il numero della
! scelta compiuta.
! routearray e' un array di puntatori a funzioni, menuarray e' un array di
! puntatori a stringhe di testo.
! NB: n deve essere minore o uguale a 5 altrimenti il menu non sta nel
! display.
shwmenu:
      PUSH  BP
      MOV   BP, SP

      PUSH  CX          ! salvataggio
      PUSH  SI

      ! Preparazione stampa a video del menu

      ! Calcolo righe vuote, non usate dal menu:
      ! vuote = disponibili - (n usate dal menu + 1 dal titolo)
      MOV   CX, DISPLAYLN
      SUB   CX, +8(BP)  ! n (menu)
      DEC   CX          ! titolo

      ! Argomenti di drwscr per le righe vuote.
      CMP   CX, 0
      JE    2f
1:    PUSH  NULL
      LOOP  1b

      ! Argomenti di drwscr per le voci del menu e il titolo
2:    MOV   CX, +8(BP)  ! n
      INC   CX
      MOV   SI, +6(BP)  ! menuarray
```

```
2:    LODS
      PUSH  AX
      LOOP  2b

      CALL  drwscr
      ADD   SP, 12

      PUSH  +8(BP)            ! n
      CALL  readchc
      ADD   SP, 2
      MOV   CX, AX            ! salva scelta in CX
      SHL   AX, 1       ! AX * 2, l'array e' di parole non di byte

      MOV   BX, +4(BP)  ! routearray
      ADD   BX, AX
      CALL  (BX)

      MOV   AX, CX            ! ritorna il numero della scelta

      POP   SI          ! ripristino
      POP   CX

      MOV   SP, BP
      POP   BP
      RET


! void noop (void)
! Non fa nulla. Associata alle voci "0. Annulla" dei menu.
noop:
      RET


.SECT .DATA
usrroute:
      .WORD noop, opendoor, chgpass
usrmenu:
      .WORD mecancl, mepass, medoor, mtuser

admroute:
      .WORD noop, opendoor, chgpass, usrmng
admmenu:
      .WORD mecancl, meusmng, mepass, medoor, mtadm

! Menu title
mtuser:
      .ASCIZ      "MENU UTENTE"
mtadm:
      .ASCIZ      "MENU AMMINISTRATORE"

! Menu entry
medoor:
      .ASCIZ      "1. Apertura porta           "
mepass:
      .ASCIZ      "2. Modifica password        "
meusmng:
      .ASCIZ      "3. Gestione utenti          "
mecancl:
      .ASCIZ      "0. Annulla                  "
```

rom.s

```
! rom.s - gestione del database utenti
!
! Giacomo Ritucci, Paolo Pennestri, 30/07/2007


.SECT .TEXT

! int authusr (id, *pass)
! Confronta la password passata come argomento con quella salvata nel record
! id della rom.
! Ritorna 0 se le password coincidono, -1 altrimenti.
authusr:
      PUSH  BP
      MOV   BP, SP

      ! Costruzione puntatore alla password salvata nella romimg.
      PUSH  +4(BP)            ! id
      CALL  getlnoff
      ADD   SP, 2
      ADD   AX, MAXUSRLEN+1
      ADD   AX, romimg

      ! Confronto con la password passata come argomento.
      PUSH  PASSLEN
      PUSH  AX          ! campo pass del record #id
      PUSH  +6(BP)            ! pass
      CALL  memcmp
      ADD   SP, 8

      ! Pass coincidono, ritorna 0.
      CMP   AX, 0
      JE    9f
      ! Altrimenti ritorna -1.
      PUSH  errpass
      PUSH  noaccess
      CALL  showerr
      ADD   SP, 4
      MOV   AX, -1

9:    MOV   SP, BP
      POP   BP
      RET


! int srchrom (*name)
! Cerca user tra i campi utente dei record della romimg.
! Se user esiste ritorna il suo id, altrimenti ritorna -1.
srchrom:
      PUSH  BP
      MOV   BP, SP

      PUSH  BX
      PUSH  CX

      MOV   CX, (numusers)
      ! DEC CX

      ! Costruzione offset al record.
1:    PUSH  CX
```

```
      CALL  getlnoff
      ADD   SP, 2
      SUB   AX, RECORDLEN
      MOV   BX, AX
      ADD   BX, romimg

      ! Confronto stringa passata come argomento con nome utente del record.
      PUSH  BX          ! romimg + offset
      PUSH  +4(BP)            ! name
      CALL  strcmp
      ADD   SP, 4
      CMP   AX, 0
      LOOPNE      1b

      ! Controllo esito: se utente e' stato trovato ritorna id.
      CMP   AX, 0
      JE    2f
      ! Altrimenti ritorna -1.
      MOV   AX, -1
      JMP   9f

2:    MOV   AX, CX            ! CX = id utente

9:    POP   CX
      POP   BX

      MOV   SP, BP
      POP   BP
      RET


! int getnumus (void)
! Ritorna il numero di utenti contenuti nel file rom.txt, dividendo la
! dimensione del file per la lunghezza di un record (RECORDLEN).
getnumus:
      PUSH  BP
      MOV   BP, SP

      PUSH  BX

      ! Con _LSEEK determina dimensione file.
      PUSH  SEEK_END
      PUSH  0
      PUSH  0
      PUSH  (romfd)
      PUSH  _LSEEK
      SYS
      ADD   SP, 10

      ! Divisione per lunghezza di un record.
      MOV   BX, RECORDLEN
      DIV   BX

      POP   BX

      MOV   SP, BP
      POP   BP
      RET


! int getlnoff (id)
! Ritorna l'offset del record numero id rispetto all'inizio della rom,
```

```
! moltiplicando la lunghezza di un record per il valore di id.
getlnoff:
      PUSH  BP
      MOV   BP, SP

      MOV   AX, RECORDLEN
      MUL   +4(BP)            ! id

      MOV   SP, BP
      POP   BP
      RET


! int openrom (void)
! Apre il file che simula la rom.
! SIDE EFFECT:
! - salva in romfd il valore del file descriptor del file della rom.
openrom:
      PUSH  BP
      MOV   BP, SP

      ! Apertura del file rom.txt in lettura e scrittura.
1:    PUSH  RDWR
      PUSH  rompath
      PUSH  _OPEN
      SYS
      ADD   SP, 6

      ! Se open fallisce mostra errore e riprova.
      CMP   AX, -1
      JNE   8f

      PUSH  callhelp
      PUSH  romerr
      CALL  showerr
      ADD   SP, 4
      JMP   1b

      ! Altrimenti, salva il file descriptor e ritorna 0.
8:    MOV   (romfd), AX
      MOV   AX, 0

9:    MOV   SP, BP
      POP   BP
      RET


! int getromsz (void)
! Ritorna il numero di byte usati in romimg, basandosi sul numero di utenti e
! sulla lunghezza di una riga della rom.
getromsz:
      PUSH  BP
      MOV   BP, SP

      PUSH  BX

      MOV   AX, (numusers)
      MOV   BX, RECORDLEN
      MUL   BX

      POP   BX
```

```
      MOV   SP, BP
      POP   BP
      RET


! int loadrom (void)
! Legge tutta la rom e ne salva l'immagine in memoria. Calcola il numero di
! utenti presenti contando i record del file.
! Ritorna 0 se riesce, -1 se fallisce.
! SIDE EFFECT:
! - salva in numusers il numero di righe lette.
! - se fallisce chiude il file della rom.
loadrom:
      PUSH  BP
      MOV   BP, SP

      ! Lettura di tutto il file.
      PUSH  MAXROMLEN
      PUSH  romimg
      PUSH  (romfd)
      PUSH  _READ
      SYS
      ADD   SP, 8

      ! Se _READ ritorna errore, chiude il file della rom e mostra un
      ! messaggio d'errore.
      CMP   AX, -1
      JNE   8f

      PUSH  (romfd)
      CALL  _CLOSE
      SYS
      ADD   SP, 2
      MOV   (romfd), -1

      PUSH  callhelp
      PUSH  romrderr
      CALL  showerr
      ADD   SP, 4

      ! Ritorna -1
      MOV   AX, -1
      JMP   9f

      ! Calcolo e salvataggio del numero di utenti.
8:    CALL  getnumus
      MOV   (numusers), AX
      MOV   AX, 0

9:    MOV   SP, BP
      POP   BP
      RET


! int saverom (void)
! Ricrea il file della rom salvando interamente l'immagine della rom in
! memoria.
! Ritorna 0 se riesce, -1 se fallisce.
! SIDE EFFECT:
! - chiude e riapre romfd.
saverom:
      PUSH  BP
```

```
      MOV   BP, SP

      ! Chiusura file descriptor della rom.
      PUSH  (romfd)
      PUSH  _CLOSE
      SYS
      ADD   SP, 4
      CMP   AX, -1
      JE    9f

      ! Riapertura di un file vuoto.
      PUSH  0644
      PUSH  rompath
      PUSH  _CREAT
      SYS
      ADD   SP, 6
      CMP   AX, -1
      JE    9f

      ! Salvataggio nuovo file descriptor.
      MOV   (romfd), AX

      ! Calcolo dimensioni effettive romimg.
      CALL  getromsz

      ! Scrittura buffer nel nuovo file.
      PUSH  AX          ! dimensione rom
      PUSH  romimg
      PUSH  (romfd)
      PUSH  _WRITE
      SYS
      ADD   SP, 8
      CMP   AX, -1
      JE    9f

      ! Riuscito, ritorna 0.
      MOV   AX, 0

9:    MOV   SP, BP
      POP   BP
      RET


! int romusadd (*newuser, *newpass)
! Aggiunge un nuovo utente a romimg e salva rom.txt
! Ritorna 0 se riesce, -1 se fallisce.
! SIDE EFFECT: incrementa numusers.
romusadd:
      PUSH  BP
      MOV   BP, SP

      PUSH  BX

      ! Calcolo dimensione romimg.
      CALL  getromsz
      MOV   BX, AX

      ! Puntatore posizionato alla fine di romimg.
      ADD   BX, romimg

      ! Inizializzazione spazio nuovo record.
      PUSH  RECORDLEN
```

```
	PUSH  0
	PUSH  BX          ! romimg + offset
	CALL  memset
	ADD   SP, 6

	! Copia del nome utente all'inizio della riga.
	PUSH  +4(BP)            ! newuser
	PUSH  BX          ! romimg + offset
	CALL  strcpy
	ADD   SP, 4

	! Puntatore posizionato all'inizio del campo password e copia di
	! newpass.
	ADD   BX, MAXUSRLEN+1
	PUSH  PASSLEN
	PUSH  +6(BP)
	PUSH  BX
	CALL  memcpy
	ADD   SP, 6

	! Incremento numero di utenti e salvataggio rom.txt.
	INC   (numusers)
	CALL  saverom

	POP   BX

	MOV   SP, BP
	POP   BP
	RET


! int romusdel (id)
! Rimuove la riga numero id da romimg, decrementa il numero di utenti e salva
! rom.txt.
romusdel:
	PUSH  BP
	MOV   BP, SP

	PUSH  BX

	! Calcolo offset riga successiva a quella da rimuovere.
	PUSH  +4(BP)            ! id
	CALL  getlnoff
	ADD   SP, 2
	MOV   BX, AX
	ADD   BX, RECORDLEN

	! Calcolo dimensione dati da spostare.
	CALL  getromsz
	SUB   AX, BX

	! Se non ci sono dati da spostare id e' l'ultima riga e si salta la
	! copia.
	CMP   AX, 0
	JE    8f

	! Costruzione puntatore alla riga.
	ADD   BX, romimg

	! Copia delle righe successive alla numero id sulla numero id.
	! Le aree di memoria si sovrappongono ma la copia e' possibile perche'
	! avviene dagli indirizzi piu' alti a quelli piu' bassi.
```

```
      PUSH  AX          ! dimensione
      PUSH  BX          ! riga successiva a quella da rimuovere
      SUB   BX, RECORDLEN
      PUSH  BX          ! riga da rimuovere
      CALL  memcpy
      ADD   SP, 6

      ! Decremento numero di utenti e salvataggio rom.txt.
8:    DEC   (numusers)
      CALL  saverom

      POP   BX

      MOV   SP, BP
      POP   BP
      RET


! void editpass (id, *newpass)
! Sovrascrive la password della riga id e salva la rom.
editpass:
      PUSH  BP
      MOV   BP, SP

      ! Calcolo offset riga.
      PUSH  +4(BP)            ! id
      CALL  getlnoff
      ADD   SP, 2

      ! Costruzione puntatore al primo carattere della password.
      ADD   AX, MAXUSRLEN+1
      ADD   AX, romimg

      ! Copia della nuova password in romimg.
      PUSH  PASSLEN
      PUSH  +6(BP)            ! newpass
      PUSH  AX          ! romimg + offset
      CALL  memcpy
      ADD   SP, 6

      ! Salvataggio romimg in rom.txt
      CALL  saverom

      MOV   SP, BP
      POP   BP
      RET


! void inititer (int mode)
! Inizializza le strutture dati che tengono traccia dell'iterazione sui
! record. Da eseguire prima di ogni nuova iterazione.
! L'argomento mode imposta il tipo di iterazione: 0 = inserimento, 1 =
! alfabetico.
inititer:
      PUSH  BP
      MOV   BP, SP

      ! Imposta modalita'.
      MOV   AX, +4(BP)
      MOV   (itmode), AX

      ! Azzera id.
```

```
      MOV   (iterid), 0

      ! Azzera il buffer.
      PUSH  MAXUSRLEN+1
      PUSH  0
      PUSH  itrusrn
      CALL  memset
      ADD   SP, 6

      ! Uno spazio come primo carattere e' utile all'inizio dell'iterazione
      ! in ordine alfabetico.
      MOVB  (itrusrn), ' '

      MOV   SP, BP
      POP   BP
      RET


! void romnext (void)
! Passa al nome utente successivo (secondo la modalita' specificata da
! inititer) nell'iterazione sui record. Il nome utente viene memorizzato nel
! buffer itrusrn, il suo id in iterid. Se l'iterazione e' giunta all'ultimo
! utente, la funzione ritorna al primo.
romnext:
      PUSH  BP
      MOV   BP, SP

      ! Controllo modalita' interazione.
      CMP   (itmode), 0
      JNE   1f

      ! Ordine di inserimento:
      ! passa al prossimo id: iterid = (iterid + 1) % numusers
      INC   (iterid)
      MOV   AX, (iterid)
      CMP   AX, (numusers)
      JL    2f

      ! Se l'iterazione era arrivata alla fine, riparte da 1.
      MOV   (iterid), 1
      JMP   2f

      ! Ordine alfabetico
1:    CALL  nxtalpha

      ! Costruzione puntatore al nome utente.
2:    PUSH  (iterid)
      CALL  getlnoff
      ADD   SP, 2
      ADD   AX, romimg

      ! Copia in itrusrn.
      PUSH  AX
      PUSH  itrusrn
      CALL  strcpy
      ADD   SP, 4

      MOV   SP, BP
      POP   BP
      RET
```

```
! void nxtalpha (void)
nxtalpha:
      PUSH  BP
      MOV   BP, SP

1:    PUSH  itrusrn     ! -2(BP) = ptr username
      PUSH  0     ! -4(BP) = ptr record candidato
      PUSH  0     ! -6(BP) = ptr record corrente

      ! Inizializzazione ptr record corrente.
      CALL  getromsz
      ADD   AX, romimg
      SUB   AX, RECORDLEN
      MOV   -6(BP), AX

      ! Se non e' ancora stato trovato un candidato...
ciclo:      CMP   -4(BP), 0
      JNE   match

      ! ... si cerca il primo utente > iterid.
      PUSH  -6(BP)            ! corrente
      PUSH  -2(BP)            ! username
      CALL  strcmp
      ADD   SP, 4
      CMP   AX, 0
      JGE   avan

      ! Trovato: candidato = corrente.
      MOV   AX, -6(BP)
      MOV   -4(BP), AX

      JMP   avan

      ! ... altrimenti si confronta il candidato con i rimanenti utenti: un
      ! utente diventa il candidato se viene prima del candidato corrente e
      ! dopo l'iterid.
match:      PUSH  -4(BP)            ! candidato
      PUSH  -6(BP)            ! corrente
      CALL  strcmp
      ADD   SP, 4
      CMP   AX, 0
      JGE   avan

      ! corrente < candidato, bisogna controllare se e' anche > iterid
      PUSH  -6(BP)            ! corrente
      PUSH  -2(BP)            ! username
      CALL  strcmp
      ADD   SP, 4
      CMP   AX, 0
      JGE   avan

      ! corrente e' il nuovo candidato
      MOV   AX, -6(BP)
      MOV   -4(BP), AX

      ! Avanti il prossimo!
avan: MOV   AX, -6(BP)
      SUB   AX, RECORDLEN
      MOV   -6(BP), AX

      CMP   AX, romimg
      JG    ciclo
```

```
      ! -4(BP) e' il puntatore al prossimo utente da visualizzare; se e'
      ! rimasto 0, siamo arrivati all'ultimo utente della lista e
      ! ricominciamo dal primo.
      MOV   AX, -4(BP)
      CMP   AX, 0
      JNE   9f

      ! Reinizializza dati e salta all'inizio di questa procedura.
      PUSH  1
      CALL  inititer
      MOV   SP, BP            ! pulizia stack
      JMP   1b

      ! Altrimenti calcola l'id utente a partire dal puntatore.
9:    SUB   AX, romimg
      PUSH  RECORDLEN
      DIV   -8(BP)

      MOV   (iterid), AX      ! salvataggio id

      MOV   SP, BP
      POP   BP
      RET



.SECT .DATA
rompath:
      .ASCIZ      "./rom.txt"
romerr:
      .ASCIZ      "ERRORE ACCESSO ROM"
romrderr:
      .ASCIZ      "ERRORE LETTURA ROM"
romwrerr:
      .ASCIZ      "ERRORE SCRITTURA ROM"
callhelp:
      .ASCIZ      "Contattare assistenza."
noaccess:
      .ASCIZ      "ERRORE AUTENTICAZIONE"
errpass:
      .ASCIZ      "Password errata."

.SECT .BSS
romfd:
      .SPACE      2
romimg:
      .SPACE      MAXROMLEN
numusers:
      .SPACE      2

! Buffer per tenere traccia dell'iterazione sui record della rom.
iterid:
      .SPACE      2
itprop:
      .SPACE      2
itnext:
      .SPACE      2
itrusrn:
        .SPACE  MAXUSRLEN+1
itmode:
      .SPACE      2
```

string.s

```
! string.s - operazioni su stringhe
!
! Giacomo Ritucci, Paolo Pennestri, 28/07/2007


.SECT .TEXT

! void memset (*buf, ch, buflen)
! Inizializza tutti i buflen byte di buf al carattere ch.
memset:
      PUSH  BP
      MOV   BP, SP

      PUSH  CX
      PUSH  DI

      MOV   CX, +8(BP)  ! buflen
      MOVB  AL, +6(BP)  ! ch
      MOV   DI, +4(BP)  ! buf

      REP   STOSB

      POP   DI
      POP   CX

      MOV   SP, BP
      POP   BP
      RET


! void memcpy (dst, src, n)
! Copia i primi n byte del buffer puntato da src nel buffer puntato da dst, che
! deve essere grande a sufficienza.
memcpy:
      PUSH  BP
      MOV   BP, SP

      ! salvataggio registri utilizzati
      PUSH  CX
      PUSH  SI
      PUSH  DI

      ! preparazione alla copia
      MOV   DI, +4(BP)  ! dst
      MOV   SI, +6(BP)  ! src
      MOV   CX, +8(BP)  ! numero di byte

      ! copia: decrementa CX finche' e' maggiore di 0
      REP   MOVSB

      ! ripristino registri utilizzati
      POP   DI
      POP   SI
      POP   CX

      MOV   SP, BP
      POP   BP
      RET
```

```
! int strlen (ptr)
! Ritorna la lunghezza della stringa puntata da ptr, terminatore escluso.
strlen:
      PUSH  BP
      MOV   BP, SP

      ! salvataggio registri usati
      PUSH  CX
      PUSH  DI

      ! preparazione al calcolo
      MOV   AX, 0       ! terminatore da cercare
      MOV   CX, -1            ! contatore iterazioni
      MOV   DI, +4(BP)  ! stringa

      ! scansione stringa (decrementa CX)
      REPNZ SCASB

      ! calcolo lunghezza
      NEG   CX
      SUB   CX, 2
      MOV   AX, CX

      ! ripristino registri usati
      POP   DI
      POP   CX

      MOV   SP, BP
      POP   BP
      RET


! void strcpy (dst, src)
! Copia la stringa puntata da src, terminatore compreso, nel buffer puntato da
! dst, che deve essere grande a sufficienza.
strcpy:
      PUSH  BP
      MOV   BP, SP

      ! lunghezza stringa src in AX, incrementato per contare anche il
      ! terminatore
      PUSH  +6(BP)            ! src
      CALL  strlen
      ADD   SP, 2
      INC   AX

      ! copia usando memcpy
      PUSH  AX          ! lunghezza src
      PUSH  +6(BP)            ! src
      PUSH  +4(BP)            ! dst
      CALL  memcpy
      ADD   SP, 6

      MOV   SP, BP
      POP   BP
      RET

! int memcmp (buf1, buf2, n)
! Confronta i primi n byte dei due buffer.
! Ritorna
! -1 se buf1 e' minore di buf2
```

```
!  0 se buf1 e buf2 sono uguali
!  1 se buf1 e' maggiore di buf2
memcmp:
      PUSH  BP
      MOV   BP, SP

      PUSH  CX          ! salvataggio
      PUSH  SI
      PUSH  DI

      MOV   AX, 0       ! init valore di ritorno

      MOV   CX, +8(BP)  ! n
      MOV   DI, +6(BP)  ! buf2
      MOV   SI, +4(BP)  ! buf1

      REPZ  CMPSB
      JE    9f          ! str1 == str2
      JG    1f          ! str1 > str2
      MOV   AX, -1
      JMP   9f

1:    MOV   AX, 1

9:    POP   DI          ! ripristino
      POP   SI
      POP   CX

      MOV   SP, BP
      POP   BP
      RET


! int strcmp (srt1, str2)
! Confronta le due stringhe, che devono avere il terminatore.
! Ritorna:
! -1 se str1 e' alfabeticamente precedente a str2
!  0 se str1 e' identica a str2
!  1 se str1 e' alfabeticamente successiva a str2.
strcmp:
      PUSH  BP
      MOV   BP, SP

      ! Calcolo lunghezza strlen
      PUSH  +4(BP)            ! str1
      CALL  strlen
      ADD   SP, 2
      INC   AX          ! terminatore incluso

      PUSH  AX          ! len
      PUSH  +6(BP)            ! str2
      PUSH  +4(BP)            ! str1
      CALL  memcmp
      ADD   SP, 6

      ! Ritorna quello che viene ritornato da memcmp.

      MOV   SP, BP
      POP   BP
      RET
```

usr.s

```
! usr.s - routine per la gestione delle scelte utente.
!
! Giacomo Ritucci, Paolo Pennestri, 02/08/2007

! void serveusr (void)
! Presenta il menu utente sul display.
serveusr:
      PUSH  BP
      MOV   BP, SP

      PUSH  3
      PUSH  usrmenu
      PUSH  usrroute
1:    CALL  shwmenu
      CMP   AX, 2       ! modifica password
      JE    1b
      ADD   SP, 6

      MOV   SP, BP
      POP   BP
      RET

opendoor:
      PUSH  BP
      MOV   BP, SP

      PUSH  0x3
      CALL  OUT_2
      ADD   SP, 2

      MOV   SP, BP
      POP   BP
      RET


! void chgpass (void)
! Richiede una nuova password e la sostituisce in rom.txt al posto della
! vecchia.
chgpass:
      PUSH  BP
      MOV   BP, SP

      ! Inizializzazione buffer.
      PUSH  PASSLEN+1
      PUSH  0
      PUSH  newpass
      CALL  memset
      ADD   SP, 6

      ! Richiesta password.
      PUSH  newpass
      PUSH  msgnewps
      CALL  askpass
      ADD   SP, 2

      ! Controllo lunghezza password.
      PUSH  newpass
      CALL  strlen
      ADD   SP, 2
```

```
      CMP   AX, 8
      JE    1f

      ! Se troppo corta, schermata di errore e uscita.
      PUSH  psleninf
      PUSH  errlen
      CALL  showerr
      ADD   SP, 2
      JMP   9f

      ! Modifica password.
1:    PUSH  newpass
      PUSH  (userid)
      CALL  editpass
      ADD   SP, 4

9:    MOV   SP, BP
      POP   BP
      RET


.SECT .DATA
msgnewps:
      .ASCIZ      "MODIFICA PASSWORD"
```