

MICROCONTRÔLEURS MICROCONTRÔLEURS ET SYSTÈMES NUMÉRIQUES TRAVAIL PRATIQUE NO 8

MT GROUPE A	MT Groupe B	EL		
No du Groupe	Premier Etudiant	Second Etudiant	Evaluation	Visa Correcteur
093	Nathann Morand	Felipe Ramirez		

8. INTERFACE UART (RS232) ET CHAINES DE CARACTÈRES

Les communications séries constituent une importante catégorie parmi les méthodes de transmission généralement utilisées par les microcontrôleurs. Ce travail pratique voit l'étude du protocole RS232 par l'établissement d'une transmission entre le PC et la carte STK-300 sur l'exemple de transmission de caractères ASCII. Dans une deuxième partie, des opérations sur les chaînes de caractères sont étudiées.

8.1 LE PROTOCOLE RS232

Un programme terminal est nécessaire à établir une communication avec le PC par l'interface série. Les applications PuTTY et RealTerm utilisées sous Windows prennent les mêmes paramètres de configuration de la transmission.

Dans les deux cas, la configuration de transmission est la suivante:

Bits par seconde: 9600, bits de données: 8, parité: aucune,
bits d'arrêt: 1, contrôle de flux: aucun.

Le cable blanc FTDI est utilisé qui permet de connecter le PC par un port USB vers le connecteur UART de la carte STK300. Ce cable doit être connecté avant de démarrer le terminal.

Par défaut, RealTerm sera utilisé, et configuré comme présenté en Figure 8.1. Chercher RealTerm sous Windows en sélectionnant la fonction de recherche de programme et insérant le nom complet.

La connection physique utilise un port USB de l'ordinateur. Ainsi, le port qui doit être configuré dans RealTerm peut varier suivant la méthode d'installation (software, drivers). La procédure suivante peut être appliquée (en DLL ou sur un ordinateur privé) afin de trouver le port.

- sous l'onglet Port, sélectionner de champ Port puis une recherche de tous les ports à disposition par "Exhaustive Search by Opening;"
- l'installation en DLL semble être sur le Port \VCP0; La Figure 8.1 présente une configuration testée en DLLs.

les autres ports peuvent être essayés si le port choisi ne fonctionne pas; afin de tester, on peut charger le programme uart1.asm ou uart2.asm et émettre des touches de clavier, et les observer à l'oscilloscope, suivant la procédure décrite ci-dessous.

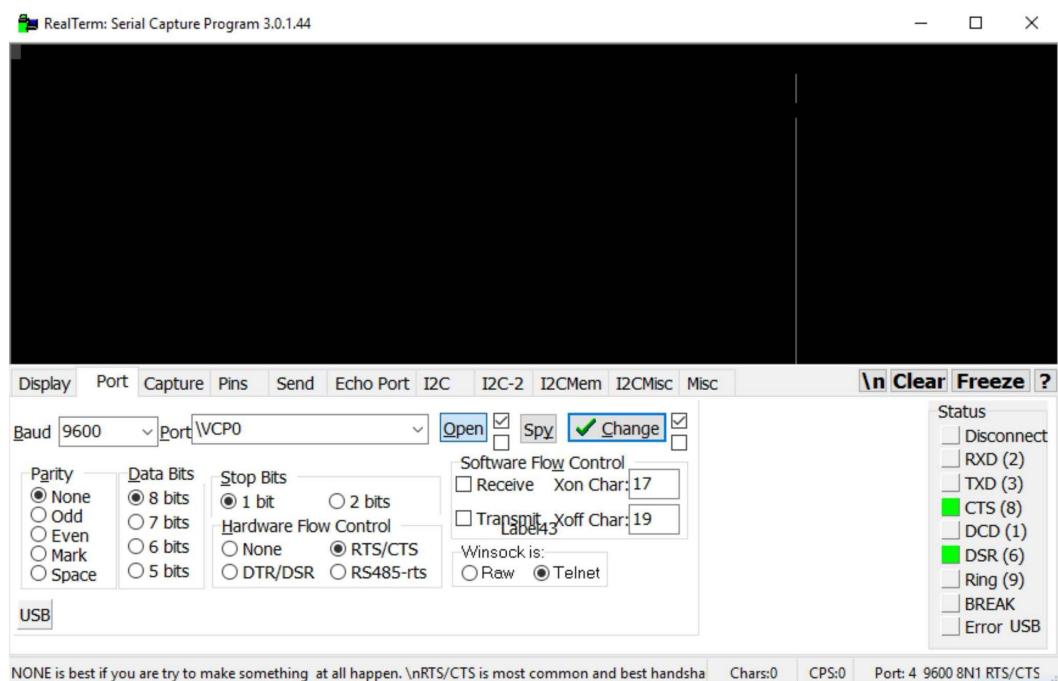
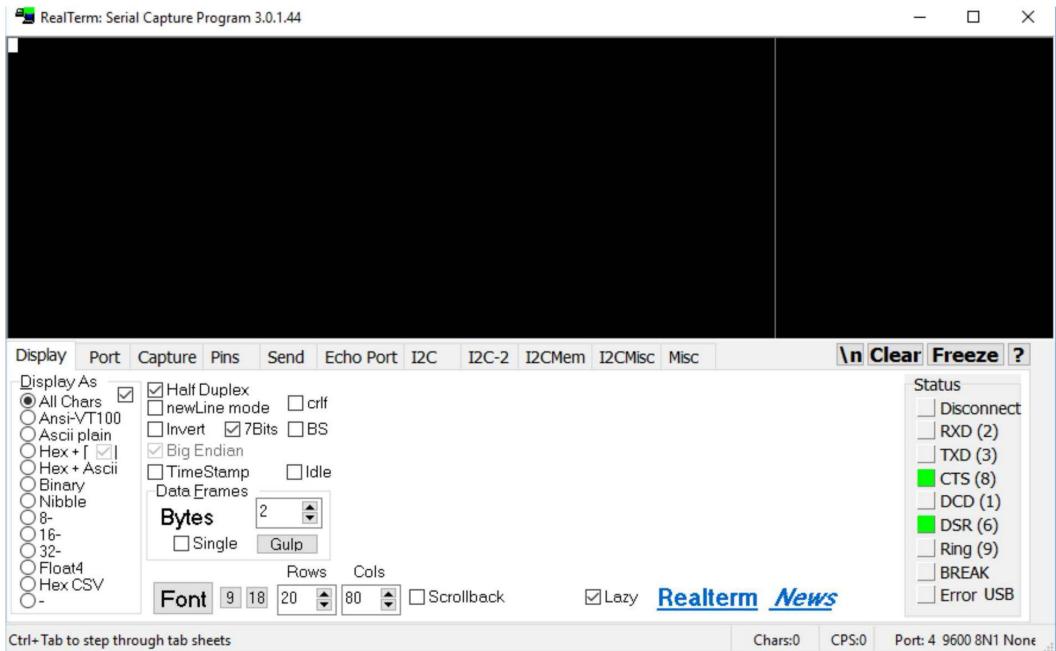


Figure 8.1: Configuration de RealTerm.

Le choix de Data Bits à 7 ou 8 bits a une influence sur la présence d'un bit 8 comme MSB qui est à un car le code ASCII (non-étendu) n'utilise que moins de 128 caractères. Si les LEDs de status sur la droite ne sont pas conformes, cliquer sur Change. Si le port ne semble pas actif, cela est signalé par un message situé au fond à droite "Port: Closed;" alors, il faut reconfigurer et trouver le bon port et si nécessaire cliquer sur Open. La transmission et réception peuvent être observées sur les LEDs de status TXD et RXD. Il est nécessaire de cliquer dans la zone d'édition (noire) avant d'entrer un caractère pour transmission.

PuTTY peut aussi être utilisé, comme un terminal simple et robuste. Configuration de PuTTY (Windows 7):

- au lancement, choisir une transmission série (Figure 8.2),
- par un click sur le bouton de droite de la souris dans le menu, puis Change settings, on peut reconfigurer la partie Connection→Serial (Figure 8.3.b),
- puis, l'écho des touches est forcé à l'affichage sous Terminal→Local Echo→Force On (Figure 8.3.a)

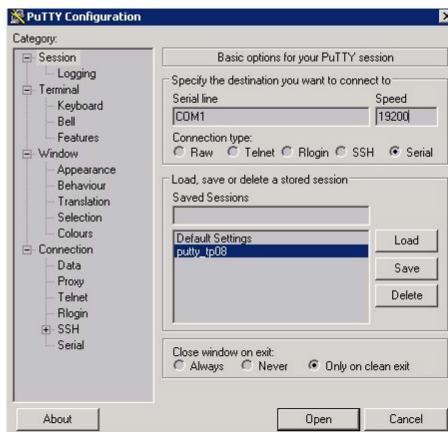


Figure 8.2: Configuration du port COM1: au lancement de l'application PuTTY.

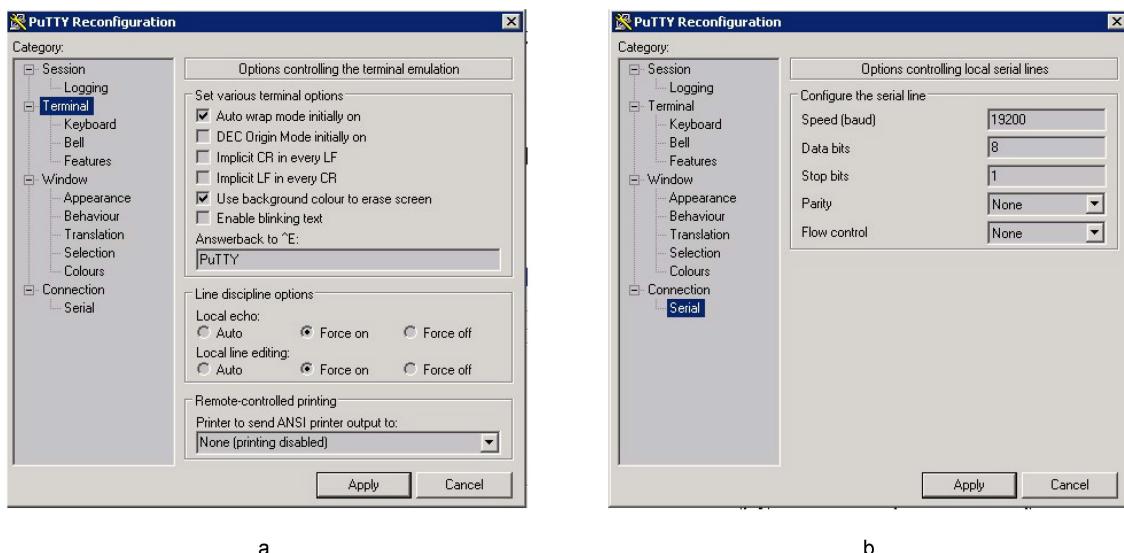


Figure 8.3: Configuration de PuTTY.

Observez, et reproduisez sur la Figure 8.4 les signaux émis lorsque vous affectez une entrée par le terminal de votre PC vers la carte utilisant le protocole RS232. Placez la probe de l'oscilloscope sur la pin PE0 (Rx). Configurez l'oscilloscope de la façon suivante:

- CH1: DC, 10X, Invert off, 5V/DIV;
- AUTO/NORM, afin d'échantillonner une seule trame et garder l'écran figé sur celle-ci;
- TRIGGER: Edge, Falling, CH1, Normal, DC, 2.5V;
- HORIZONTAL: Main, Level, 250us/DIV.

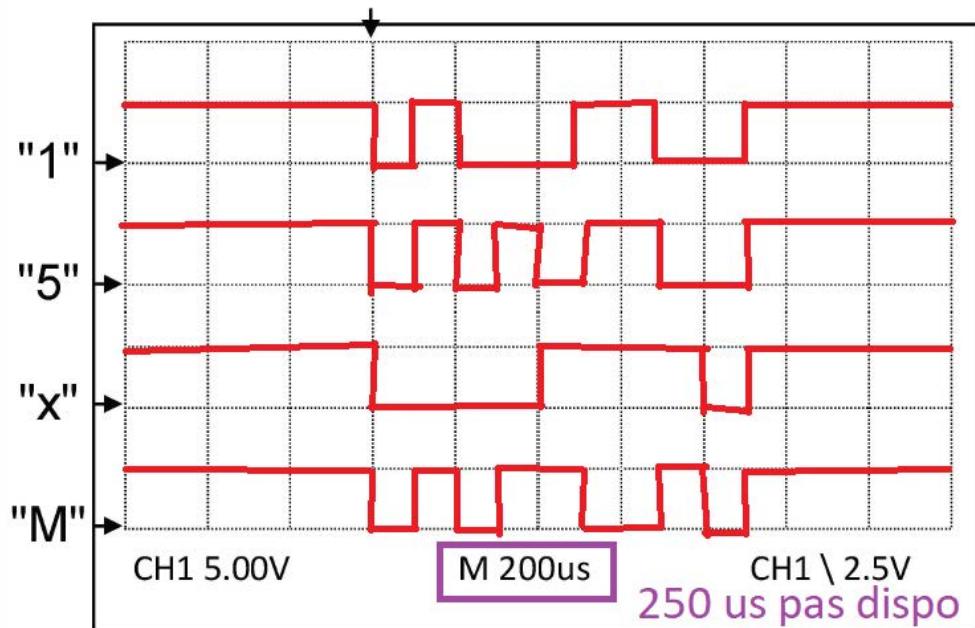


Figure 8.4: Transmission RS232.

Sur le terminal, tapez les caractère 1 puis 5, x, et finalement M. Reportez les quatre signaux observés. Indiquez clairement les start bit (S), stop bit (P), et les huit data bits. Si vous utilisez PuTTY, alors il est nécessaire d'appuyer la touche enter du PC afin de déclencher l'envoi des caractères. Donnez la suite des valeurs logiques obtenues en Figure 8.5.

$1 \equiv$	s 1 0 0 0 1 1 0 0 P
$5 \equiv$	s 1 0 1 0 1 1 0 0 P
$x \equiv$	s 0 0 0 1 1 1 1 0 P
$M \equiv$	s 1 0 1 1 0 0 1 0 P

Figure 8.5: Niveaux logiques lors de la transmission.

Les bits sont transmis avec le bit de poids faible en tête. Inversez l'ordre des bits mesurés, et donnez le code ASCII en binaire et en hexadécimal en Figure 8.6.

$1 \equiv$	0b 0 0 1 1 0 0 0 1	\equiv	0x 3 1
$5 \equiv$	0b 0 0 1 1 0 1 0 1	\equiv	0x 3 5
$x \equiv$	0b 0 1 1 1 1 0 0 0	\equiv	0x 7 8
$M \equiv$	0b 0 1 0 0 1 1 0 1	\equiv	0x 4 D

Figure 8.6: Codes hexadécimaux transmis.

8.2 LECTURE/ÉCRITURE

Chargez le programme `uart1.asm`. Ce programme lit une valeur reçue sur RX et la retransmet vers le PC par TX. L'état d'une LED permet bascule (toggle) à chaque fois qu'un nouveau caractère est traité.

Reportez en Figure 8.7 les signaux observés en plaçant le probe relié au canal 1 sur PE0 (Rx, receive from PC), et le probe relié au canal 2 sur PE1 (Tx, transmit to PC), lorsque vous appuyez sur la touche Y (shift-y).

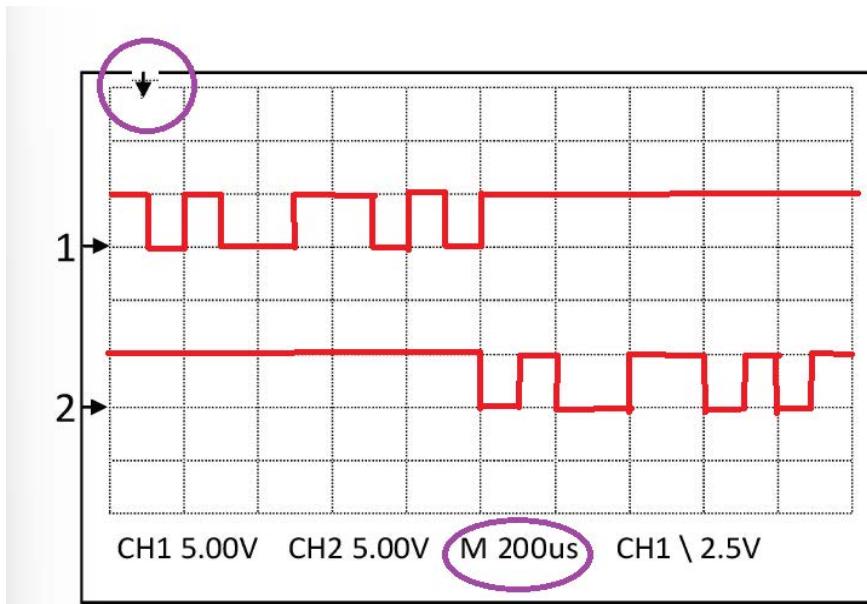


Figure 8.7: Signaux observés lors de la transmission par `uart1.asm`.

Les routines `getc` et `putc` servent à effectuer les opérations de décalages en entré et sortie (shift-in, shift-out) nécessaires aux transferts entre la ligne série et le registre d'entrée/sortie contenant la donnée sur 8-bit. Cette opération aurait pu être réalisée au moyen de compteurs initialisés à la valeur huit. Une autre méthode est cependant appliquée, et un autre test est réalisé:

- Quelle condition indique dans `putc` que tous les huit bits ont été émis (shift-out) ?

`r20 = 0x00`

- Quelle condition indique dans `getc` que huit bits ont été reçus (shift-in) ?

`carry = 1`

Modifiez le programme principal (main) pour que toutes les 100 ms une lettre de l'alphabet soit envoyée vers le terminal; complétez le code du nouveau programme nommé `uart1-alphabet.asm`, donné en Figure 8.8.

```

main:
    ldi [r23], $61 ; initialize char to 'a'
next:
    mov [r20], r23
    rcall putc      ; put character c
    WAIT_MS 100    ; wait 100 ms
    inc [r23]       ; increment c
    cpi [r23], 'z' ; compare c with 'z'
    brne [next]    ; branch to next if not equal
    rjmp main

```

Figure 8.8: main de `uart1-alphabet.asm`

Etudiez et complétez uart1-alphabet.asm. L'écriture débute immédiatement après avoir terminé le téléchargement du programme sur le système-cible, et les lettres apparaissent sur le terminal, sans fin.

Pourquoi a-t-on besoin d'utiliser un registre intermédiaire r23, au lieu de travailler directement avec le registre ctx (=r20) ?

la sous-routine modifie r20 et donc la valeur est perdu.

On constate que le caractère z n'est pas affiché. Comment peut-on changer la condition de comparaison afin que la lettre z soit aussi affichée ?

cpi r23, 0x7b ; compare c with ..

8.3 UTILISATION DU MODULE UART

Chargez le programme uart2.asm. Ce programme utilise le module UART de l'AVR avec ses registres à décalage qui s'occupent de l'envoi et de la réception des caractères.

Quel est le Baud rate maximum (acceptable) avec un quartz à 4MHz ? Dans quel document trouve-t'on cette information ? 250'000 Bauds. Datasheet ATmega128, p194

Est-ce que le registre UDR signifiant UART I/O Data Register est identique pour la transmission et la réception ?

non, le registre UDR est en réalité 2 registre qui ont la même adresse l'un en lecture et l'autre en écriture

A quelle Baud l'UART est-elle configurée ? Dans quel fichier se trouve cette information ?

9600 dans le fichier uart.asm.

Reconfiguez le terminal RealTerm au Baud correct, et redémarrez l'application de communication qui réalise un écho du caractère émis.

8.4 OPÉRATIONS AVEC DES CHAÎNES DE CARACTÈRES

Chargez le programme string1.asm (ne téléchargez pas). Simulez ce programme et observez comment sont utilisés les pointeurs (x, y, z) pour passer les constantes de texte comme arguments à des fonctions. Une chaîne de caractères est terminée par un NUL, dont le code ASCII est 0x00.

Que fait la fonction strstrldi ?

charge la chaîne de caractères pointé par z dans l'emplacement pointé par x

Les deux registre pointeurs x et z sont les argument de la fonction strstrldi. Quelle est leur fonction ?

- z pointe sur l'adresse de la source dans la flash
- x pointe sur l'adresse de la destination dans la pile

Quelle est l'adresse de la chaîne constante sbf1 (exprimé en bytes) ? 0x260. C'est la valeur écrite en x pour pointer sur le début de la chaîne constante.

Mettez un breakpoint sur chacune des lignes suivantes:

```
CXZ strstrldi,sbf1,2*cs1    ; load buffers with string constants
CXZ strstrldi,sbf2,2*cs2
CXZ strstrldi,sbf3,2*cs3
```

Simulez en continu jusqu'au breakpoint; dès le breakpoint atteint, simulez en pas à pas et observez les modifications sur les pointeurs et la SRAM.

8.5 EXTRACTION DE SOUS-CHAÎNES

En utilisant comme point de départ la fonction `strstrncpy` à modifier, écrivez les trois routines décrites en Table 8.9. Il faut à chaque fois ajouter un offset au pointeur, décrémenter un compteur et tester.

fonction	paramètres	explication	
<code>strstr_left</code>	<code>x, y, a</code>	copie les premiers <code>a</code> caractères de <code>y</code> vers <code>x</code>	$x \leftarrow left(y, a)$
<code>strstr_right</code>	<code>x, y, a</code>	copie les derniers <code>a</code> caractères de <code>y</code> vers <code>x</code>	$x \leftarrow right(y, a)$
<code>strstr_mid</code>	<code>x, y, a, b</code>	copie <code>b</code> caractères à partir de la position <code>a</code> de <code>y</code> vers <code>x</code>	$x \leftarrow mid(y, a, b)$

Table 8.9: Paramètres des fonctions à écrire.

- Pour la fonction `strstr_left` il faut décompter le nombre de caractères dans `a0` jusqu'à zéro, ou jusqu'à ce que la fin de la chaîne soit atteinte, au cas où elle serait plus courte que `a0`. Complétez le code en Figure 8.10.

```
strstr_left:
    ld      w, y+
    dec    [a0]
    breq   PC+4
    st      x+, [w]
    tst    w
    brne  strstr_left
    ret
```

Figure 8.10: `str_left`.

- Pour la fonction `strstr_right` il faut d'abord trouver la fin de la chaîne (`y`), et ensuite soustraire `a0` au pointeur `y` (2-byte). Complétez le code en Figure 8.11.

```
strstr_right:
    ld      w, y+ ; find the end of string (y)
    tst    w
    brne  PC-2
    sub    yl, a0 ; decrement y by a
    brcc  PC+2 ; check for carry
    dec    yh ; adjust in case of carry

    ld      w, y+
    st      x+, w
    tst    w
    brne  PC-3
    ret
```

Figure 8.11: `str_right`.

- Pour la fonction `strstr_mid` il faut combiner les deux méthodes précédentes: d'abord trouver la fin de la chaîne (`y`), et ensuite soustraire `a0` du pointeur `y` (2-byte), et copier `b0` caractères vers (`x`). Complétez le code en Figure 8.12.

```

strstr_mid:
    ld      w, y+    ; find the end of string (y)
    tst    w
    brne  PC-2
    sub   yl, a0    ; decrement y by a
    brcc  PC+2    ; check for carry
    dec    yh        ; adjust in case of carry

    ld      w, y+
    dec   b0
    breq PC+4
    st    x+, w
    tst    w
    brne PC-5
    ret

```

Figure 8.12: str_mid.

Insérez les réponses que vous proposez pour les trois sous-routines précédentes dans le programme `string2.asm` donné en Figure 8.13 afin de tester vos résultats. Observez le résultat en mémoire SRAM.

```

; file string2_s.asm target ATmega128L-4MHz-STK300
; purpose lab extraction de sous-chaines

.include "macros.asm"           ; include macro definitions
.include "definitions.asm"       ; include register/constant definitions

; === interrupt table ===
.org 0
jmp reset

.org 0x46
.include "string.asm"           ; include string manipulation routines
.include "uart.asm"              ; include UART routines
.include "printf.asm"            ; include formatted printing routines

reset:
LDSP RAMEND                   ; Load Stack Pointer (SP)
rcall UART0_init
rjmp main

; === string constants in program memory ===
cs1: .db "hello world.",0
cs2: .db "how are you today?",0
cs3: .db 0

; === string buffers in SRAM ===
.dsseg
sbfl: .byte 32
sbf2: .byte 32
sbf3: .byte 32
.cseg

main:
CXZ strstrldi,sbf1,2*cs1      ; load string constants into buffer SRAM
CXZ strstrldi,sbf2,2*cs2
CXZ strstrldi,sbf3,2*cs3

ldi a0,                      ; COMPLETE HERE
ldi b0,                      ; COMPLETE HERE

CXY strstr_left,sbf1,sbf2    ; test strstr_left routine; comment others
;CXZ strstr_right,sbf1,sbf2  ; uncomment to test
;CXZ strstr_mid,sbf1,sbf2   ; uncomment to test

rjmp main

strstr_left:
ld w,y+                       ; COMPLETE HERE (3 lines)
tst w
brne strstr_left
ret

strstr_right:
dec yh                         ; COMPLETE HERE (5 lines)
; adjust in case of carry

ld w,y+                         ; COMPLETE HERE (2 lines)

brne PC-3
ret

strstr_mid:
; COMPLETE HERE (6 lines)

ld w,y+                         ; COMPLETE HERE (2 lines)
st x+,w
tst w
; COMPLETE HERE (1 line)
ret

```

Figure 8.13: string2.asm.

