

MICROCONTROLEURS MICROCONTROLEURS ET SYSTEMES NUMERIQUES TRAVAIL PRATIQUE NO 4

MT GROUPE A	MT Groupe B	EL		
No du Groupe	Premier Etudiant	Second Etudiant	Evaluation	Visa Correcteur
43	Joaquim Silveira	Edwin Berthchy		

4. UTILISATION DES SOUS-ROUTINES, INTRODUCTION À L'AFFICHAGE LCD

Ce travail pratique voit dans une première partie l'étude des sous-routines, et leur comparaison avec les macros. Le fonctionnement de la pile est également abordé.

Dans une deuxième partie, une introduction au fonctionnement du module LCD est présentée. L'accès à un contrôleur de LCD standard, ainsi que les opérations de base sont abordés.

4.1 LA PILE

p. 117

La pile (stack) est une zone mémoire utilisée pour la sauvegarde de variables temporaires et des adresses de retour lors des appels de sous-routines. Le SP signifiant stack pointer est un pointeur sur une zone en mémoire SRAM. Le ATmega128 possède deux registres spéciaux SPL et SPH qui stockent cette valeur.

- L'instruction push r0 place le contenu du registre r0 est bloqué à l'adresse courante pointée par SP puis, le SP est déréférencé.
- L'instruction pop r0 incrémente le SP, puis place le contenu (pointé) dans r0.

Simulez en pas-à-pas le code donné en Figure 4.1 et observez les registres et pointeurs (SP) par Debug→Window→Registers.. → Processor Status(mieux)

```

ldi r16,0xff
ldi r17,0x10
out SPL,r16
out SPH,r17
loop:
inc r16
push r16
rjmp loop

```

Figure 4.1: Traitement avec la pile (1).

Le SP pointe à l'adresse 0x10FF. Que se passe-t'il dans la boucle loop ?

~~R11~~ ~~0xFF~~ \rightarrow 0x00 \rightarrow SP \downarrow ~~0x01~~ 10fe
~~0x00~~ 10ff

[2] (qui s'incrément à chaque boucle)

Le contenu de r16 est placé sur la pile (0x00 \rightarrow 0x10FF, 0x01 \rightarrow 0x10FE,...). L'instruction push dure 2 cycle(s).

p. 338

Les instructions push et pop sont toujours utilisées en paire. L'instruction push est utilisée pour la sauvegarde temporaire du contenu de registres qui doivent être mis à disposition d'une macro ou routine par exemple. L'instruction pop restitue la valeur stockée.

Complétez, et simulez en pas-à-pas le code donné en Figure 4.2.

<pre> ldi r16,0xff out SPL,r16 ldi r17,0x10 out SPH,r17 ldi r16,0x0a ldi r17,0x0b loop:push r16 ; save r17, r16 push r17 clr r17 ; do other stuff with r16,r17 clr r16 pop r17 ; restore r16, r17 pop r16 rjmp loop </pre>	<p>SP \rightarrow [] 0x10FF \rightarrow [] 0x10FF \rightarrow [] 0x10FF</p>
--	---

Figure 4.2: Traitement avec la pile (2).

Quelle valeur est restituée par l'instruction pop dans le cas où l'instruction push a été précédemment utilisée plusieurs fois ? La dernière valeur qui a été mise sur la pile. Ce principe de mémoire s'appelle LIFO, signifiant Last In First Out, en opposition avec FIFO signifiant First In First Out.

\Leftarrow LIFO? p. 3

\hookrightarrow first in last out

Il convient toutefois de travailler avec la pile et les instructions push et pop de façon très rigoureuse. Voici trois cas dans lesquels des erreurs ont été commises et qui conduisent à différents comportements erronés. Identifiez les erreurs, leurs sources et conséquences. Pour vous aider dans cette tâche, affichez les fenêtres "Processor status," "Registers" et "Memory: dataRAM" à l'adresse de la pile, et simulez en pas-à-pas.

- Simulez le code donné en Figure 4.3, puis répondez aux questions.

```
; file failure01.asm target ATmega128L-4MHz-STK300
```

```
; purpose study incorrect code operation
```

```
.include "macros.asm"
```

```
.include "definitions.asm"
```

→ > dependencies → m128def.inc

```
reset:
```

LDSP RAMEND ; set up stack pointer (SP) 0x10ff

ldi r16, 0x11
ldi r17, 0xaa

= p. 335

main: ↗ r16

ldi xl, 0xff
ldi xh, 0x10

↘ r27

push r16

pointeur x / low
X y high



p. 338

st x, r17 → valeur r17 à l'adresse X ?

pop r16 → 0xaa
rjmp reset

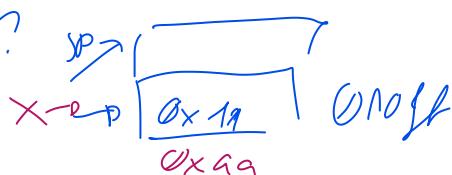


Figure 4.3: Erreur type No. 1, failure01.asm.

- après l'exécution du pop r16, le contenu de r16 n'est pas la valeur attribuée dans le reset. Quelle valeur est la valeur placée dans r16 ? ;
- la pile est-elle une zone mémoire protégée ? ;
- que s'est-il donc passé ?
- quel comportement erroné a ainsi été la source de l'erreur ?

- Simulez le code donné en Figure 4.4, puis répondez aux questions.

```

; file failure02.asm target ATmega128L-4MHz-STK300
; purpose study incorrect code operation
.include "macros.asm"
.include "definitions.asm"

;=====
; this stands for a big macro, over which a
; programer may have lost control
.macro DISTRACT ;void
    ;several instructions ...
    push r18
    push r17

    ;several instructions ...
    pop r17

    ;several instructions ...
.endmacro

reset:
    LDSP RAMEND      ;set up stack pointer (SP)
    ldi r16, 0x11
    ldi r17, 0xaa
    ldi r18, 0x55

main:
    push r16
    DISTRACT
    pop r16 → 0x55
    rjmp reset

```

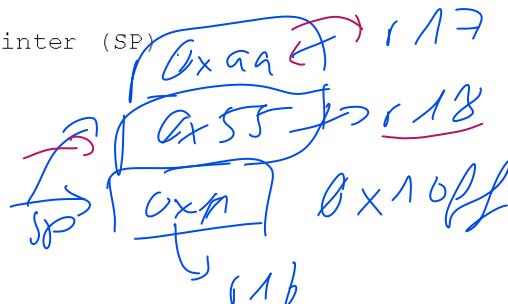


Figure 4.4: Erreur type No. 2, failure02.asm.

- après l'exécution du pop, r16 ne récupère pas la valeur attribuée dans le reset, quelle valeur récupère-t'il à la place ? 0x55
- quel comportement erronné a ainsi été la source de l'erreur ? il y a un pop qui manque dans la macro

- Simulez le code donné en Figure 4.5, puis répondez aux questions. Affichez le code désassemblé et suivez en pas-à-pas.

```

; file failure03.asm      target ATmega128L-4MHz-STK300
; purpose study incorrect code operation
.include "macros.asm"
.include "definitions.asm"

reset:
    LDSP RAMEND ;set up stack pointer (SP)

    ldi r16, 0x11
    ldi r17, 0xaa
    ldi r18, 0x55

main:
    push r16
    rcall distract → appel d'une sous-routine?
    pop r16
    → stockage sur la pile?

=====
; this stands for a big subroutine, where a
; programmer may have lost control
distract:
    ;several instructions ...
    push r17
    push r18

    ;several instructions ...
    pop r18 → pop qui manque

    ;several instructions ...
    ret → auquel → auquel? → auquel?
    → auquel?

```

Figure 4.5: Erreur type No. 3, failure03.asm.

- quelle est l'adresse à laquelle le PC saute à l'exécution de l'instruction ret ? ;
- qu'est-ce que cela signifie du point de vue de l'exécution ? ;
- d'où cette adresse a-t'elle été copiée vers le PC ? ;
- comment ces deux octets sont-ils parvenu (par "qui" ont-ils été placés) à cet endroit ?
;
- quel comportement erronné a ainsi été la source de l'erreur ? ;
- quelle est l'instruction exécutée immédiatement après l'exécution du pop r16 ? ;
- expliquez ce qui est anormal dans le comportement observé après l'exécution du pop r16, la raison et comment il faut y remédier.

4.2 COMPARAISON ENTRE MACRO ET SOUS-ROUTINE

4.2.1 SOUS-ROUTINE

Une sous-routine est un morceau de code dont le début est marqué par une étiquette, et qui se termine par l'instruction `ret`. Complétez la sous-routine `mul5` donnée en Figure 4.6, qui multiplie l'argument `r16` par cinq, assumant que l'instruction de multiplication ne soit pas disponible sur ATmega128.

X5

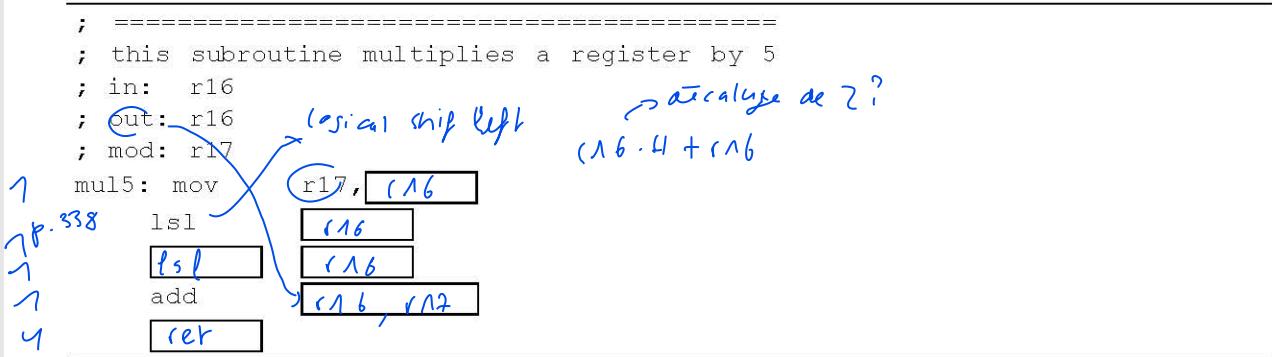


Figure 4.6: Sous-routine `mul5`.

A l'exécution de l'appel à une sous-routine, l'adresse de retour est stocké sur la pile, et l'adresse de la première instruction de la sous-routine à exécuter est placée dans le PC. Le SP est décrémenté afin de pointer vers la prochaine adresse libre. Simulez le code donné en Figure 4.7 basé sur la sous-routine `mul5` développée précédemment. La sous-routine `mul5` est appelée trois fois. Indiquez les éléments suivants: l'adresse courante du PC (`PC=0x...`), l'adresse de retour courante de la sous-routine (`ret-addr=0x...`), l'argument courant d'entrée (`in=0x...`), et l'argument courant de sortie (`out=0x...`).

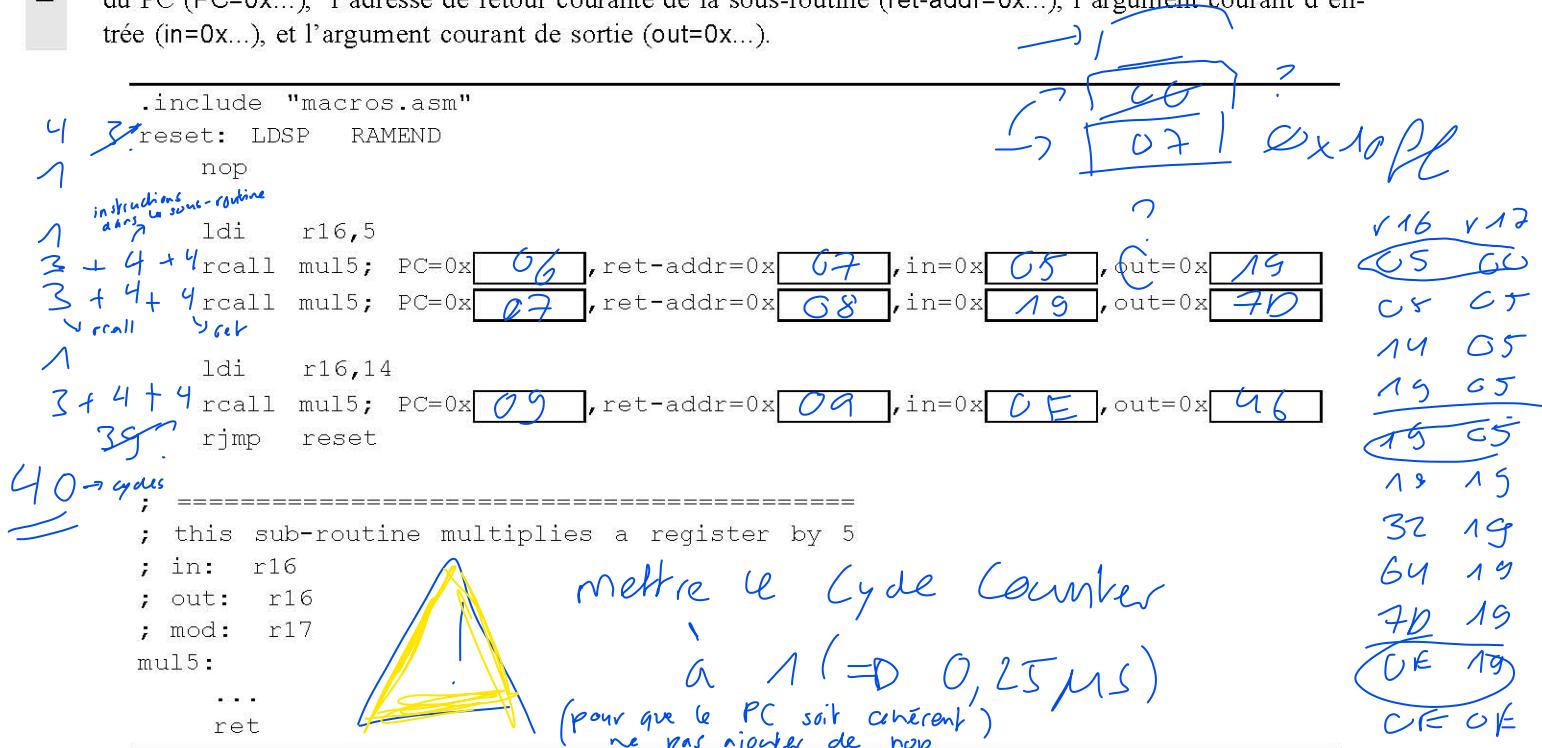


Figure 4.7: Utilisation d'appels aux sous-routines.

A quel endroit/adresse(s) peut-on observer l'adresse de retour de sous-routine ?

dernière adresse de la ram interne ? définie par

`RAMEND = 0x10ff`

4.2.2 MACRO

Ecrivez en Figure 4.8 la macro MUL5 qui multiplie le contenu d'un registre par cinq, en appliquant la même structure et les mêmes instructions que dans le cas de la sous-routine mul5.

```

macro MUL5; reg @0 ??
1 mov r17, @0
1 lsl @0
1 lsl @0
1 add @0, r17
.endmacro
→ 19args = 40 - 7.3
    
```

Figure 4.8: Macro MUL5.

La macro MUL5 peut être appliquée à tous les registres, sauf r17. Dans ce cas ce ne serait pas une multiplication par cinq qui serait effectuée, mais par 8.

4.2.3 COMPARAISON

Réécrivez le programme donné en Figure 4.7 en remplaçant les appels aux sous-routines par des invocations des macros.

Assemblez, puis comparez les codes désassemblés obtenus.

Comment se manifeste l'appel à une sous-routine dans le code désassemblé ?

→ moins rapide et fait intervenir la pile mais E mémoire

Comment se manifeste l'invocation d'une macro dans le code désassemblé ?

l'invocation de la macro implique la "réécriture" de chaque instruction présente dans la macro

→ plus rapide mais prend plus de mémoire

La macro MUL5 regxx est remplacée par 4 instruction(s) qui durent 4 cycles(s) ou 1 µs. L'appel de fonction mul5 introduit 7 cycles supplémentaires (3 cycles pour l'instruction rcall et 4 cycles pour l'instruction ret).

Placez un point d'arrêt sur l'instruction rjmp reset. Comparez les temps d'exécution.

Le code faisant appel à des sous-routines nécessite 10 us.

Le code faisant appel à des macros nécessite 4,75 us.

! → + 1 Cycle Courrier

4.3 INTRODUCTION AU LCD

4.3.1 INITIALISATION DU LCD

P. 125 à 127

Le LCD est accédé par un circuit de contrôle comprenant deux registres:

- IR signifie instruction register, et reçoit et stocke des informations de type "contrôle" (éffacement de l'écran, affichage...)
- DR signifiant data register et reçoit et stocke des informations de type "données" → code ASCII des caractères à afficher

p. M6

Ces deux registres sont adressés par le MCU comme la mémoire externe. Le registre IR est situé à l'adresse 0x **8000** et le registre DR à l'adresse 0x **C000**.

Pour accéder à la mémoire externe, il faut activer deux bits du registre de configuration MCUCR, signifiant **SRE (external SRAM enable)/SRW10 (external SRAM wait state)**. Complétez et commentez en Figure 4.9 la suite de trois instructions nécessaires à activer la mémoire externe.

```

in  r16, MCUCR ; charger le registre de configuration sur r16
sbr r16, (1<< SRE)+(1<< SRW10) ; set des bits SRE et SRW10
out MCUCR, r16 ; modifier le registre de configuration par la valeur de r16

```

Figure 4.9: Instructions d'activation de la mémoire externe.

Chargez le programme **lcd1.asm** donné en Figure 4.10, qui réalise une initialisation de LCD.

```

; file lcd1.asm      target ATmega128L-4MHz-STK300
; purpose LCD HD44780U initialization

.equ LCD_IR= 0x8000; address LCD instruction reg
.equ LCD_DR= 0xc000; address LCD data register

.macro LD_IR
a:   lds r16,LCD_IR    ; read the SRAM (LCD IR) into r16
      sbrc r16,7       ; check the busy flag (bit7)
      rjmp a           ; jump back if busy flag set
      ldi r16,@0        ; load value into r16
      sts LCD_IR,r16   ; store value to SRAM (LCD IR)
.endmacro

.reset:
      in  r16,MCUCR      ; enable ext. SRAM access
      sbr r16,(1<<SRE)+(1<<SRW10)
      out MCUCR,r16

.main:
      LD_IRb00000001    ; clear display
      LD_IRb00000010    ; return home
      LD_IRb00000110    ; entry mode set
      LD_IRb00001111    ; display on/off control
.loop:
      rjmp loop          ; infinite loop

```

Figure 4.10: lcd1.asm.

Ajoutez dans la partie main une ligne de code qui initialise le LCD pour un affichage de deux lignes.

LD_IR **0b 0010 1000** ?

Ajoutez une ligne qui initialise le LCD pour une ligne, mais utilise de grands caractères de 5x10 pixels.

LD_IR **0b0010 0100** o

0010'NF--

D → data length

N → # of lines

F → font

p. M7