

MICROCONTRÔLEURS MICROCONTRÔLEURS ET SYSTÈMES NUMÉRIQUES TRAVAIL PRATIQUE NO 4

No du Groupe	Premier Etudiant	Second Etudiant	Evaluation	Visa Correcteur
093	Nathann Morand	Felipe Ramirez		

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

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 la valeur stocké dans r0 à l’endroit pointé par SP puis, le SP est decrementé.
- L’instruction `pop r0` incrémente le SP, puis place la valeurs pointé par SP dans R0.

Simulez en pas-à-pas le code donné en Figure 4.1 et observez les registres et pointeurs (SP) par De-
bug→Window→Registers. .

```
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 . Que se passe-t'il dans la boucle loop ?

L'instruction push dure cycle(s).

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.

```
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  ; restore r16, r17
pop 
rjmp loop
```

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 ? . Ce principe de mémoire s'appelle LIFO, signifiant , en opposition avec signifiant .

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: dataIRAM” 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"

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

    ldi r16, 0x11
    ldi r17, 0xaa

main:
    ldi x1, 0xff
    ldi xh, 0x10

    push r16

    st x, r17

    pop r16
    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 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
; programmer 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
    rjmp reset

```

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

- après l'exécution du pop, r16 ne retrouve pas la valeur attribuée dans le reset, quelle valeur retrouve-t'il à la place ? **0x55** qui est le contenu de r18
- quel comportement erroné a ainsi été la source de l'erreur ? **il y a une "fuite" de mémoire dans DISTRACT. en effet on push 2 valeurs mais on ne récupère que r17 ce qui fait que quand on pop r16, on obtient r18**.

- 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

    pop r16

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

    ;several instructions ...
    pop r18

    ;several instructions ...
    ret

```

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 ?

si autre chose
- d'où cette adresse a-t-elle été copiée vers le PC ? que 0xFF en mémoire)
- comment ces deux octets sont-ils parvenus (par "qui" ont-ils été placés) à cet endroit ?

;
- quel comportement erroné a ainsi été la source de l'erreur ?
- quelle est/serait 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.

```
; =====  
; this subroutine multiplies a register by 5  
; in:  r16  
; out: r16  
; mod: r17  
mul5: mov    r17, r16  
        lsl    r16  
        lsl    r16  
        add    r16, r17  
        ret
```

(bit shift 2 fois -> multiplie par 4
ajoute encore une fois pour avec
5 fois

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...`).

```
.include "macros.asm"  
reset: LDSP  RAMEND  
        nop  
  
        ldi    r16, 5  
        rcall  mul5; PC=0x00000005, ret-addr=0x06, in=0x05, out=0x19  
        rcall  mul5; PC=0x00000006, ret-addr=0x07, in=0x19, out=0x7D  
  
        ldi    r16, 14  
        rcall  mul5; PC=0x00000008, ret-addr=0x09, in=0x06, out=0x46  
        rjmp   reset  
  
; =====  
; this sub-routine multiplies a register by 5  
; in:  r16  
; out: r16  
; mod: r17  
mul5:  
        ...  
        ret
```

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

A quel endroit/adresse(s) peut-on observer l'adresse de retour de sous-routine ? dans le PC quand on a fait l'instruction `ret`

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
    mov r17, @0
    ldl @0
    ldl @0
    add @0, r17
.endmacro
```

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 .

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 ?

le programme utilise rcall pour modifier le PC et accéder à la fonction

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

le compilateur remplace chaque instance de la macro par sa série d'instruction.

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

Placez un point d'arrêt sur l'instruction rjmp reset. Comparez les temps d'exécution. Attention, le simulateur ne calcule pas correctement si le stopwatch ou le cycle counter sont à zéro; il est préférable (mais peu pratique) de s'assurer que ces deux systèmes ne soient pas à zéro et d'effectuer la soustraction. Vérifiez votre résultat en le confrontant au résultat théorique obtenu par comptage du nombre de cycles.

Le code faisant appel à des sous-routines nécessite μ s.

Le code faisant appel à des macros nécessite μ s.

4.3 INTRODUCTION AU LCD

4.3.1 INITIALISATION DU LCD

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

- IR signifie , et reçoit et stocke des informations de type .

- DR signifiant data register et reçoit et stocke des informations de type donné a afficher telle que les caractères à afficher.

Ces deux registres sont adressés par le MCU comme la mémoire externe. Le registre IR est situé à l'adresse 0x8000 et le registre DR à l'adresse 0xc000.

Pour accéder à la mémoire externe, il faut activer deux bits du registre de configuration MCUCR, signifiant MCU general control register. Complétez et commentez en Figure 4.9 la suite de trois instructions nécessaires à activer la mémoire externe.

```
in  r16, MCUCR ; copy MCUCR dans r16
sbr r16, (1<<SRE)+(1<<SRW10) MCUCR, r16 ; set des bits SRE et SRW10
out MCUCR, r16 ; mets a jour MCUCR avec les valeurs changé
```

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_IR 0b00000001     ; clear display
    LD_IR 0b00000010     ; return home
    LD_IR 0b00000110     ; entry mode set
    LD_IR 0b00001111     ; 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 0b00111000

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

LD_IR 0b00110100