

MICROCONTRÔLEURS MICROCONTRÔLEURS ET SYSTÈMES NUMÉRIQUES TRAVAIL PRATIQUE NO 1

MT GROUPE A	MT Groupe B X	EL		
No du Groupe	Premier Etudiant	Second Etudiant	Evaluation	Visa Correcteur
<u>093</u>	Nathann Morand	Felipe Ramirez		

1. INTRODUCTION À L'ASSEMBLEUR/SIMULATEUR

Ce travail pratique est conçu comme un tutoriel, guidant le lecteur dans l'utilisation de AVR Studio. Remplissez les cases **comme celle-ci** par vos réponses !

Dans le cours de ce travail pratique, un assistant vous attribuera le numéro du kit microcontrôleur (MCU) que vous utiliserez au long du semestre pour les laboratoires. Ce kit sera rangé dans les armoires au terme du travail pratique, et sera partagé avec d'autres étudiants/cours. Veuillez en prendre soin, et annoncer à un assistant tout matériel manquant.

1.1 UTILISATION DES PCS

Veuillez vous connecter sur les PCs avec votre compte EPFL habituel, puis créer un répertoire tp01 dans un dossier sur laquelle vous bénéficiez des droits d'écriture (répertoire itinérant sur un serveur, clé USB, c:\\temp), dans lequel vous créerez les projets et placerez vos fichiers. Ce répertoire est local à votre PC afin de garantir une rapidité d'accès aux fichiers maximale. A la fin de la session, veuillez copier ce répertoire dans votre partition EPFL personnelle pour sauvegarde (my documents), et enlever tous les fichiers présents sur la partition locale de l'ordinateur. L'utilisation de laptops est possible; les logiciels et drivers (pilotes de périphériques) doivent alors être installés correctement.

Veuillez respecter scrupuleusement les directives suivantes, qui sont données par l'administrateur de la salle:

- ne jamais placer de fichiers sur le “desktop,” car ces fichiers font partie de votre configuration personnelle qui est chargée lors de chaque login à travers le réseau, sous peine de ralentir considérablement votre temps d'accès, et celui des autres;
- ne **jamais** rien déconnecter du PC; ni le clavier ou la souris, ni l'écran, ni le réseau, ...
- veuillez utiliser les salles de PCs en connaissance et en accord avec les “Directive pour l'utilisation de l'infrastructure électronique de l'EPFL”, données à l'URL suivante: https://polylex.epfl.ch/wp-content/uploads/2019/01/6.1.4_Dir_infrastructure_electronique_20141113_en.pdf;
- veuillez garder les salles dans un état de rangement et propreté décents, et ne pas y manger, boire ou fumer.

1.2 CRÉER UN NOUVEAU PROJET

Suivez les instructions suivantes:

- Créez un répertoire sur votre disc dur local, par exemple dans “my documents,” c:\Users\your-name\tp01 ; n’oubliez pas de le recopier sur votre compte EPFL en fin de session si vous désirez garder vos fichiers, sous peine de les perdre entièrement.
- Lancez le programme Atmel Studio 7.0:
Start→Programs→Atmel Studio 7.0→Atmel Studio 7.0;

La version utilisée est la version 7.0.1188, qui peut être vérifiée par le menu: help→About AtmelStudio

- Du menu File, choisissez New→Project
une fenêtre de dialogue avec le titre “New project” apparaîtra (Figure 1.1);
- dans le champ situé sur la gauche, sélectionnez Assembler, et dans le champ situé au milieu, sélectionnez AVR Assembler Project
- dans le champ situé au fond et nommé Name gardez le défaut nommé AssemblerApplication1
- dans le champ situé au fond et nommé Location entrez le chemin vers le répertoire créé précédemment, par exemple c:\Users\yourname\tp01;
- dans le champ situé au fond et nommé Solution name, gardez le défaut nommé AssemblerApplication1;

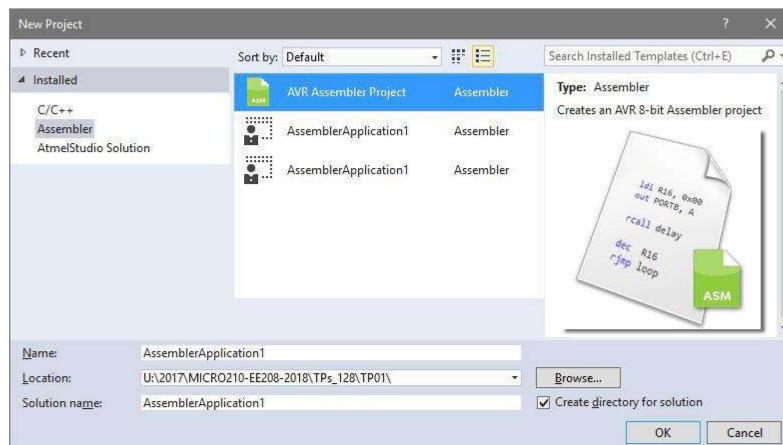


Figure 1.1: Fenêtres “New project.”

Après avoir cliqué sur le bouton OK, la fenêtre de sélection du microcontrôleur apparaît (Figure 1.2). Il faut choisir le device nommé ATmega128. Les caractéristiques du MCU peuvent être consultées dans le champ situé sur la droite de la fenêtre.

Un premier environnement de travail est créé, comprenant également un premier fichier assembleur pré-rempli (Figure 1.3).

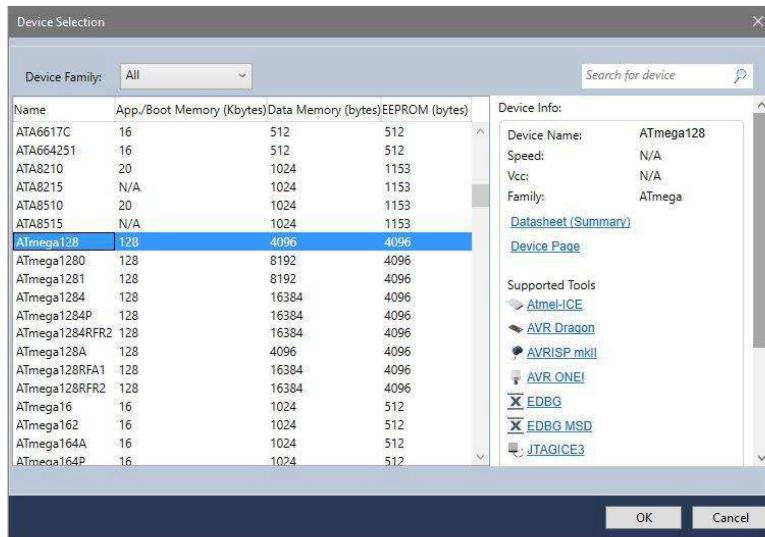


Figure 1.2: Fenêtre de sélection du device (MCU).

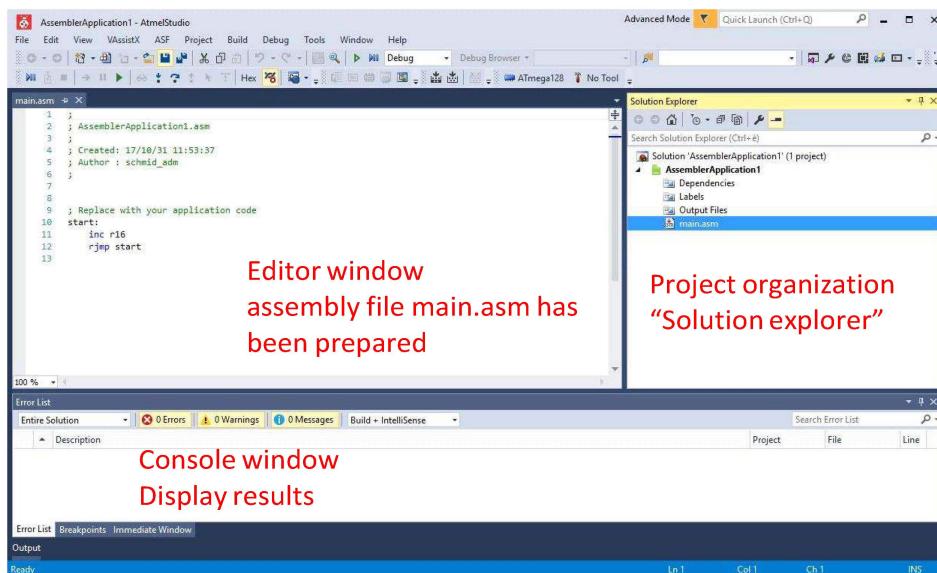


Figure 1.3: Fenêtre présentant l'environnement de développement.

1.3 CRÉER UN NOUVEAU FICHIER ASSEMBLEUR (.asm)

- Dans le Solution explorer, effectuez un clic avec le bouton de droite sur AssemblerApplication1, puis dans le menu déroulant Add→New Item.... Cette procédure ajoute un nouveau fichier dans le projet, qui bénéficie des propriétés du projet (Figure 1.4).
- Dans la fenêtre qui apparaît, sélectionnez sur la partie de gauche Assembler, puis Assembler file sur la partie de droite; finalement, dans la partie du fond, nommez le nouveau fichier ex1.asm et gardez l'extension .asm. (Figure 1.5).

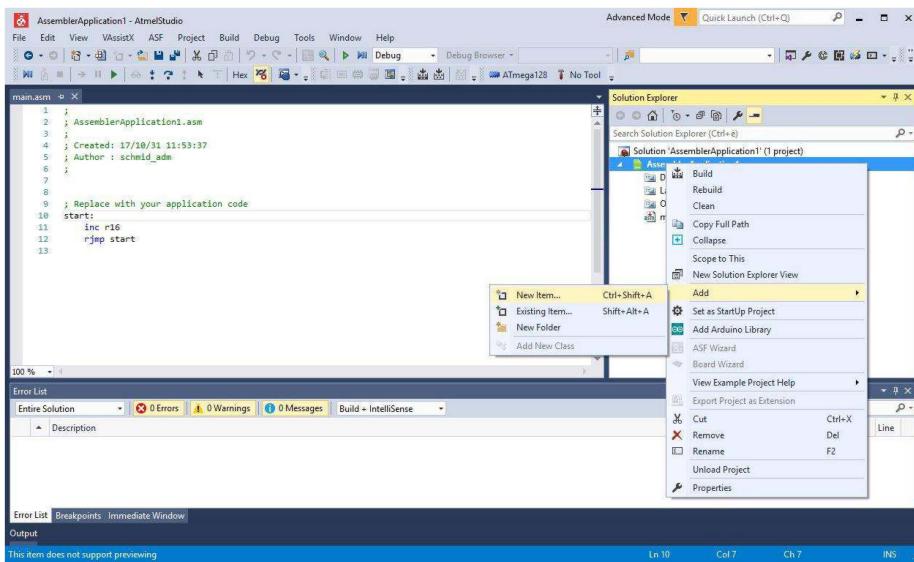


Figure 1.4: Ajout d'un nouveau fichier au projet.

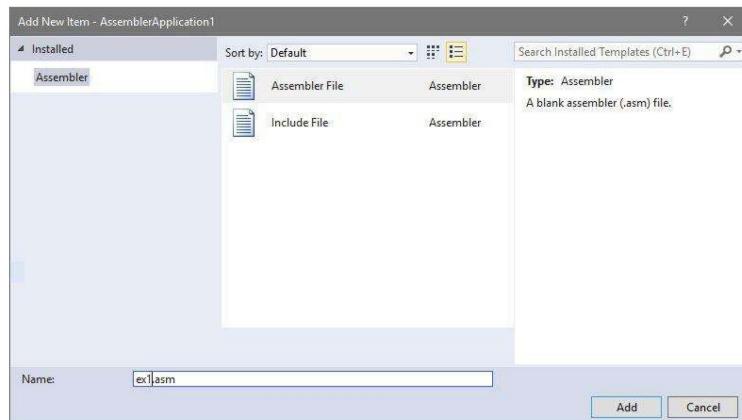


Figure 1.5: Ajout d'un nouveau fichier au projet.

Cliquez sur ADD. Le fichier ex1.asm fait partie maintenant du projet AssemblerApplication1.

Effectuez un clic au moyen du bouton de droite de la souris sur le fichier ex1.asm qui apparaît dans la fenêtre du Solution explorer, puis dans le menu déroulant sélectionnez Set as entry file, afin de marquer ce fichier comme celui qui devra être assemblé. La flèche rouge vertical dans l'icône du fichier indique que ce fichier-ci est le fichier d'entrée actif pour l'assembleur (Figure 1.6).

Ouvrez le fichier en cliquant sur le l'icône. Entrez le code proposé en Figure 1.7 dans ce fichier, au moyen de l'éditeur qui est apparu:

La fenêtre de l'éditeur comporte ainsi le code reporté en Figure 1.8.

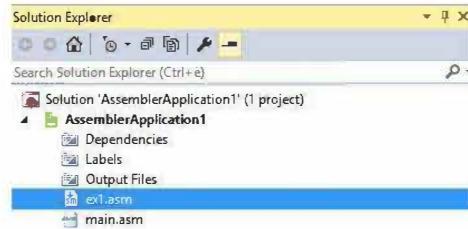


Figure 1.6: Sélection du fichier à assembler.

```

ldi      r16, 0x03
reset:
inc      r16
add      r0, r16
rjmp    reset

```

Figure 1.7: ex1.asm

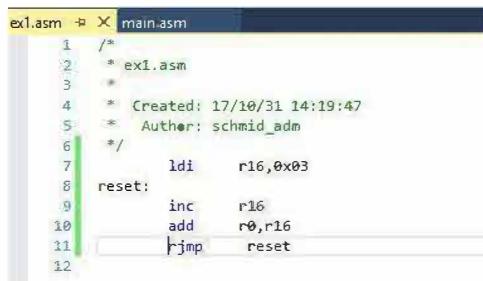


Figure 1.8: Fenêtre de l'éditeur comprenant le code ex1.asm

Assemblez ce programme avec la commande Build→Build Solution. La touche de fonction F7 est un raccourci pour l'assemblage.

Expliquez le rôle de l'assembleur:

L'assembleur permet de donner un nom plus tangible au instruction sans avoir besoin d'écrire en hexamdecimal notre code. le tout en gardant une relation casi 1-1 entre l'assembleur et le code binaire derrière.

La fenêtre située au fond de l'environnement de développement (Console) Output indique la version de l'assembleur, le nombre de mots de code générés et les erreurs éventuelles (Figure 1.9), et ainsi le succès ou non de l'assemblage.

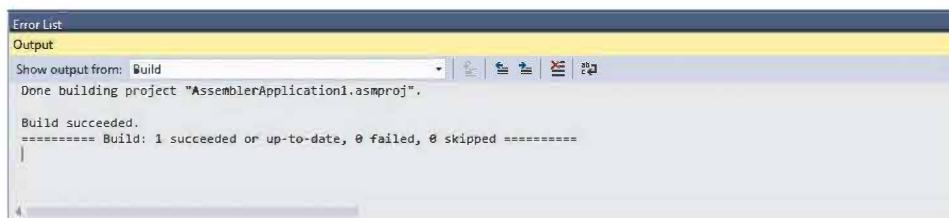


Figure 1.9: Fenêtre Output.

1.4 LE SIMULATEUR (DEBUGGER)

Simulez le programme par les commandes Build→Build Solution, puis Debug→Step into.

La première fois que vous lancez le simulateur dans un nouveau projet, il est nécessaire d'indiquer quel outil de simulation est utilisé. Une boîte de dialogue le demande, puis la fenêtre de configuration apparaît. Dans la partie de la fenêtre de configuration “Select debugger/programmer,” (Figure 1.10) choisissez:

- Simulator.

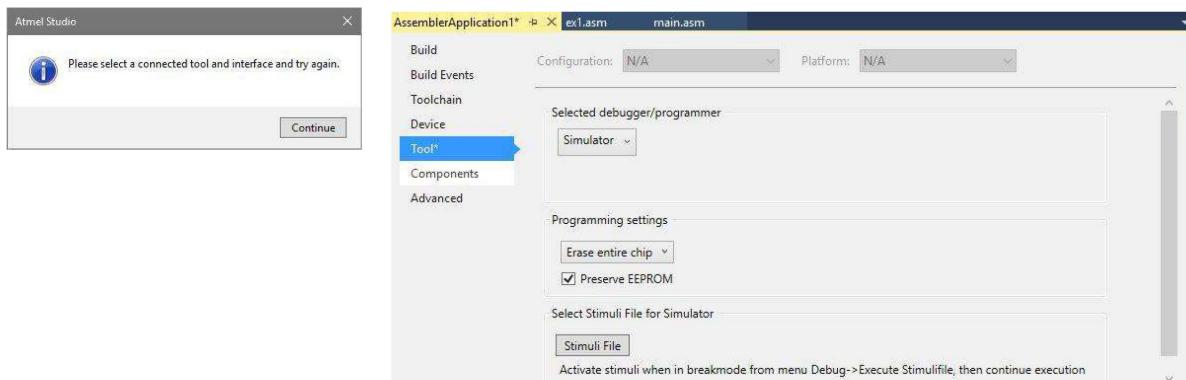


Figure 1.10: Fenêtre de configuration de l'outil, à configurer pour chaque nouveau projet.

Il est maintenant possible de simuler le programme. Parcourez le programme avec la commande Debug→Step into, qui peut être activé par la touche de fonction **F11** comme raccourci-clavier (mode pas-à-pas). Chaque instruction est alors simulée, une après l'autre, et l'état du MCU peut être vérifié à chaque instruction.

Ouvrez la fenêtre d'observation des registres: Debug→Window→Registers (r0 à r31). Redimensionnez cette fenêtre pour arranger les registres afin de faciliter la visualisation des données (Figure 1.11(a)).

Registers

R00 = 0x00 R01 = 0x00 R02 = 0x00
R03 = 0x00 R04 = 0x00 R05 = 0x00
R06 = 0x00 R07 = 0x00 R08 = 0x00
R09 = 0x00 R10 = 0x00 R11 = 0x00
R12 = 0x00 R13 = 0x00 R14 = 0x00
R15 = 0x00 R16 = 0x03 R17 = 0x00
R18 = 0x00 R19 = 0x00 R20 = 0x00
R21 = 0x00 R22 = 0x00 R23 = 0x00
R24 = 0x00 R25 = 0x00 R26 = 0x00
R27 = 0x00 R28 = 0x00 R29 = 0x00
R30 = 0x00 R31 = 0x00

(a)

Processor Status

Name	Value
Program Counter	0x00000001
Stack Pointer	0x0000
X Register	0x0000
Y Register	0x0000
Z Register	0x0000
Status Register	0111000000000000
Cycle Counter	0
Frequency	4.000 MHz
Stop Watch	0.00 µs

Registers

R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00

(b)

Figure 1.11: Fenêtre des registres (a) et de status du MCU (b).

Ouvrez la fenêtre de visualisation de l'état du processeur: Debug→Window→Processor Status (Figure 1.11(b)). Cette fenêtre vous permet de visualiser l'état actuel de plusieurs registres, comme par exemple le compteur ordinal, en anglais **Program Counter** ou en abréviation **PC**. Quel est la fonction du compteur ordinal ?

Le compteur ordinal mémorise l'adresse de l'instruction suivante à executer.

Le programme commence son exécution à l'adresse **0x00**.

Le compteur de cycles (Cycle Counter) et le chronomètre (Stop Watch) ne font pas partie du microcontrôleur réel. Ce sont des outils d'aide au déverminage (debug).

Avancez de plusieurs pas et observez le Program Counter (PC). Quelles valeurs prend-il successivement ? **0x0**, **0x1**, et **0x2**. **puis la boucle nous fait faire 0x1, 0x2,0x3,0x1,0x2, ...**

A quoi sert l'instruction inc r16 ?

elle incrémente le registre r16 de 1

Dans quel document trouvez-vous l'information détaillée sur les instructions du microcontrôleur ?

Atmel-0856-AVR-Instruction-Set-Manual.pdf

A quoi sert l'instruction add r0, r16 ?

Elle additionne le contenu de **r0 et r16** et place le résultat dans **r0**.

Quelle différence observez-vous entre les instructions inc et add au niveau de la modification des fanions ?

Les flag C et H ne sont pas affecté par l'insctuction inc. alors que add vas modifier ces flag.

Le symbole **reset** est une étiquette (label) qui prend la valeur d'une adresse. Quelle instruction se trouve à l'adresse reset ? **inc r16**.

A quoi sert l'instruction rjmp reset ?

elle modifie le PC pour qu'on retourne a l'insctruction inc

Avancez dans le programme en pas-à-pas (F11), et observez la fenêtre Processor:

L'instruction inc dure **1** cylce(s) ou **1us@1Mhz** microsecondes.

L'instruction rjmp dure **2** cycle(s) ou **2us@1Mhz** microsecondes.

Dans la fenêtre Processor Status, vous pouvez également observer les registres à usage général r0 à r31, placer le curseur dans un registre et en changer la valeur instantanée.

Dans quelle base sont affichées les valeurs des registres ? Hexadécimal.

Observez l'avancement de la flèche jaune dans la fenêtre ex1 lorsque vous simulez en mode pas-à-pas. Celle-ci indique la prochaine instruction qui va être exécutée. Simulez en pas-à-pas, et immédiatement avant que add r0, r16 ne soit exécutée, chargez 0xbb dans le registre r0, et 0x26 dans le registre r16. Le résultat de l'addition est 0xe1, et met les fanions (flags) suivants à 1: S et N.

Les différentes mémoires du microcontrôleur peuvent être visualisées dans la fenêtre Debug→Window→Memory→Memory1 (Figure 1.12) (4.19: View→Memory x).

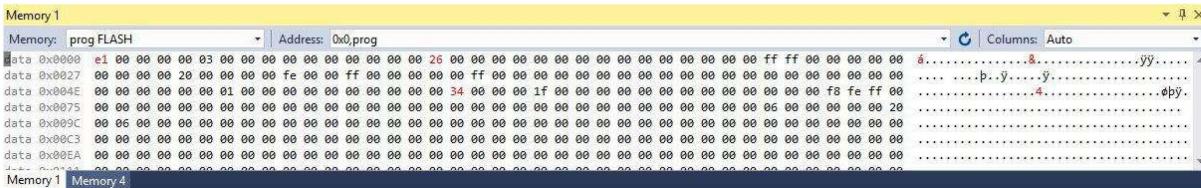


Figure 1.12: Fenêtre “Memory.” Visualisation de la mémoire programme.

Trouvez les quatre principaux types de mémoires ou registres, et leur dénomination dans le software:

<u>prog (mémoire programme)</u>	nommée <u>FLASH</u> ,
<u>data (espace mémoire pour donné)</u>	nommée <u>REGISTERS</u>
<u>data (registre de commandes des I/O)</u>	nommée <u>MAPPED IO</u>
<u>eeprom (donné persistante)</u>	nommée <u>EEPROM</u>

(les parties liées au boot du programme, BOOT_* et à l'oscillateur OSCCAL ne sont pas relevantes).

1.5 LE CODE MACHINE

Lorsque vous assemblez le fichier source, l'assembleur traduit ces instructions en code machine. Le programme ex1.asm contient quatre instructions assembleur qui produisent quatre codes machines. En cliquant sur l'icône (Debug→Window→Disassembly), il est possible de basculer entre le fichier assembleur et le fichier contenant le code machine. Au moyen d'un clic du bouton de droite de la souris dans la fenêtre Disassembly, sélectionnez les options que vous souhaitez voir affichées, par exemple Show Line Number, Show Code Bytes, etc.

Attention cependant à l'interprétation du code observé; il s'agit du code *désassemblé*, il est donc normal qu'il y ait certaines différences entre ces deux représentations d'un exécutable identique.

Trouvez le code machine en hexadécimal qui est généré lors de l'assemblage de ex1.asm, et complétez en Figure 1.13.

code machine remis dans le bon ordre

00000000 :	<u>e003</u>	LDI R16, 0x03
00000001 :	<u>9503</u>	INC R16
00000002 :	<u>0e00</u>	ADD R0, R16
00000003 :	<u>cffd</u>	RJMP PC-0x0002

Figure 1.13: Test code machine 1.

Notez que le MCU inverse l'ordre des huit bits de poids fort (placés avant le point) et des huit bits de poids faibles (placés après le point). Donc le code correct marqué par xx.yy est yyxx, par exemple fd.cf est lu cffd. Stoppez la simulation par Debug→Stop debugging.

Assemblez le code donné en Figure 1.14, observez le code désassemblé.

```

ldi      r16, 0x05
nop
nop
reset:
dec      r16
add      r0, r16
jmp      reset

```

Figure 1.14: Test code machine 2.

L'instruction de saut non-conditionnel jmp effectue un branchement à l'adresse **0x3**. Son code machine est le suivant **0c.94 03.00**; il est composé de **32** bits, alors que la majorité des instructions de l'AVR sont codées sur **16** bits. Ainsi le code machine du jmp remis dans le bon ordre est le suivant **940c 0003**.

1.6 INCRÉMENTER ET DÉCRÉMENTER DES REGISTRES

Les instructions inc et dec permettent d'incrémenter, respectivement décrémenter un registre. Créez le programme ex2.asm en Figure 1.15.

```

loop :
inc      r0
dec      r2
rjmp   loop

```

Figure 1.15: ex2.asm.

Avant d'assembler, il est nécessaire d'indiquer à l'assembleur quel est le fichier d'entrée (ex1.asm, ou ex2.asm) par le menu Set as Entry File (voir plus haut).

Assemblez (Build→Build Solution), puis exécutez en mode pas à pas.

Forcez la valeur stockée dans le registre r0 à 0xfe, la valeur stockée dans le registre r2 à 0x03.

Complétez la séquence de valeurs pour r0 à l'approche de 0: ... , 0xfe, 0xff, **0x00**, **0x01**,

Complétez la séquence de valeurs pour r2 à l'approche de 0xff: ... , 0x02, 0x01, **0x00**, **0xff**,

Expliquez ce qui se passe lors d'un dépassement de capacité. Le microcontrôleur peut-il continuer à calculer après un dépassement de capacité ?

lors d'un dépassement de capacité, le micro contrôleur utiliser des fanions pour nous indiquer le dépassement mais peut tout à fait continuer de calculer après le dépassement.

1.7 LES MESSAGES D'ERREURS

Créez le fichier ex3.asm donné en Figure 1.16.

```
reset:  
    ldi  
    ldi      r16  
    ldi      r16,10  
    ldi      r0, 10  
    clr      r16,10  
  
    rjmp    reset
```

Figure 1.16: ex3.asm.

Il y a plusieurs erreurs dans cette suite d'instructions. Essayez de l'assembler, et trouvez les messages d'erreurs, et expliquez la cause de l'erreur ! En double-cliquant sur ces messages d'erreurs, la ligne du fichier contenant le code assembleur incriminée est mise en évidence. Corrigez les erreurs.

File: ex3.asm, line 2, error:

wrong number of operand

quelle en est la cause ?

il manque le registre et la valeur à y mettre

File: ex3.asm, line 3, error:

wrong number of operand

quelle en est la cause ?

il manque la valeur à mettre dans le registre r16

File: ex3.asm, line 5, error:

invalid register

quelle en est la cause ?

r0 n'est pas modifiable via la fonction ldi

File: ex3.asm, line 6, error:

wrong number of operand

quelle en est la cause ?

clr ne nécessite pas d'argument numérique. seul le registre est nécessaire.

1.8 NOMBRES EN PRÉSENTATIONS BINAIRE, DÉCIMALE, HEXADÉCIMALE

Tous les registres sont composés de 8 bits. Chaque bit peut prendre une valeur binaire 0 ou 1. Donc, au moyen de 8 bits, 256 combinaisons différentes peuvent être choisies pour représenter de l'information. Une valeur numérique peut être représentée suivant différents formats, par exemple, au formats binaire, décimal, hexadécimal, décimal signé, ASCII. Les quatre expressions données en Figure 1.17 représentent la même valeur.

```

reset:
    ldi r16, 0b11111110 ; binary
    ldi r16, 0xfe          ; hexadecimal
    ldi r16, 254           ; decimal
    ldi r16, -2            ; decimal signé

```

Figure 1.17: Représentations du même nombre.

Ainsi ces quatre instructions produisent exactement le même code machine reproduit en Figure 1.18.

00000000 0e.ef	LDI R16, 0xFE	Load immediate
00000001 0e.ef	LDI R16, 0xFE	Load immediate
00000002 0e.ef	LDI R16, 0xFE	Load immediate
00000003 0e.ef	LDI R16, 0xFE	Load immediate

Figure 1.18: Code machine produit pour le code en Figure 1.17.

Exprimez les valeurs données dans la Table 1.1 en représentations différentes. Remplissez les cases vides.

binaire	hexadécimal	décimal non-signé	décimal signé
0b0000'0001	0x01	1	1
0b0000'1111	0x 0F	15	15
0b1111'0001	0x F0	240	- 16
0b0010'0001	0x 21	33	33

Table 1.1: Valeurs à convertir.

1.9 CALCUL D'UNE SUITE DE CARRÉS

Nous utilisons le fait que la différence entre deux carrés consécutifs est une suite de nombres impairs.

Carrés: 1, 4, 9, 16, 25, 36,... ; différence: (4-1=)3, (9-4=)5, 7, 9,...

Ecrivez un programme qui calcule une suite de carrés sur ce principe, en complétant les instructions manquantes en Figure 1.19; l'utilisation de multiplication n'est pas permise dans ce cas.

```

reset :
    clr r0
    inc r0
    mov r1, r0
loop :
    [inc r1]
    [inc r1]
    [add r0,r1]
    rjmp loop

```

Figure 1.19: Suite de carrés.

Quelle valeur trouve-t'on dans r0 et r1 au terme des trois instructions d'initialisation ?

r0 0x01	r1 0x01	.
---------	---------	---

Parcourez la boucle et vérifiez que la série produite est bien une suite de carrés. La suite des carrés produite en notation hexadécimal est: 0x00, 0x01, 0x04, [0x 09], 0x10, [0x19], [0x 24], [0x 31], etc.

1.10 CALCUL DE LA SUITE DE FIBONACCI

Dans la suite de Fibonacci chaque terme est la somme des deux termes précédents.

Par exemple: 0, 1, 1, 2, 3, 5, 8, 13, 21.

Ecrivez un programme qui utilise les registres r0, r1 et r2 calculant la suite de Fibonacci, en complétant les instructions manquantes en Figure 1.20, et en utilisant les instructions clr, inc, mov, et add.

```
reset:
    clr r0
    [clr r1 ; assure que les registre sont vide et ne contienne pas de résultat d'opération précédente]
    clr r2
    [inc r1 ; mets dans r1 pour innitialiser la suite]

loop:
    [mov r2,r0 ; les mov sont la pour qu'on conserve l'hystorique des valeurs]
    [add r2,r1 ; ajoute le resultat des 2 nombre précédent dans r2]
    [mov r0,r1]
    [mov r1,r2]
    rjmp loop
```

Figure 1.20: Suite de Fibonacci.

La suite des valeurs produite en hexadécimal est 0x00, 0x01, 0x01, 0x02, 0x03, 0x05, 0x08, [0x 0d], [0x 15], [0x 22], [0x37].

Pour interrompre la simulation sur une ligne spécifique afin d'observer un résultat particulier valide en un instant unique, il est possible d'insérer un point d'arrêt, nommé breakpoint en anglais, et qui est visible par un signe carré et rouge, situé sur la gauche du code, au point d'arrêt choisi. Comment place-t'on un point d'arrêt ?

on appuie sur F9 apres avoir cliquer sur la ligne ou on veut mettre le breakpoint

Ajoutez des commentaires sur chaque ligne qui expliquent ce que fait le programme. Un commentaire commence par un [;].

1.11 CARTE STK-300

Le kit microcontrôleurs est basé sur une carte STK-300. Il sagit d'un kit éducatif qui a été conçu dans le but d'accéder à beaucoup de points de mesures et de permettre un interfaçage facile avec des modules externes divers, et n'a pas été conçu avec un but de réalisation la plus compacte possible. Reportez sur la Figure 1.21 les principaux modules la composant, expliquez leur fonctionnalité et connections.

1	microcontrôleur Atmega 103L
	effectue les opérations
2	horloges
	donne la fréquence au microcontrôleur pour que les instruction soit executé a un rythme régulier
3	IO
	permet d'intéragir électriquement avec les périphérique
4	régulateur de tension
	assure une tension d'allimentation exacte sur tout le circuit
5	ports B relié a un afficher led
	permet d'allumé les petites lumières
6	port D relié a une série de bouton
	permet a l'utilisateur d'intéragir avec le microcontrôleur
7	DC barrel jack qui fournit la tension d'alim et bouton ON/OFF
8	port série RS232
	permet une communication via un terminal
9	port ISP
	permet de brancher le programmeur AVR et charger des programme
10	SRAM
	permet de stocker des information en mémoire
11	connecteur de LCD et réglage de la luminosité
	permet de connecter la carte a un écrans LCD externe

Figure 1.21: Carte STK-300.

Expliquez l'utilité des câbles plats souples en 12 et 13, et ce qui se passe s'ils sont déconnectés.

les câbles permettent de déconnecter les port B et D de la barre de led réspéctivement du bouton et de pouvoir les branché à d'autre périphérique. (donc si ils sont déconneteter on ne pourra plus utiliser la barre de led et les boutons.

