INF1900

Projet initial de système embarqué
Travail pratique No. 9
Trajectoire préprogrammée

Objectif: Faire effectuer automatiquement des déplacements au robot
Durée: Une semaine exactement
Travail préparatoire: travail pratique no. 5 sur le RS-232 et la mémoire externe
Documents à remettre: Le travail pratique est matière à une entrevue de laboratoire à partir du mercredi 20 mars durant les périodes de laboratoire. Un horaire est établi pour les entrevues. L'entrevue se fera par équipe de quatre étudiants avec un responsable de la partie technique. Durant l'entrevue, le robot devra effectuer une démonstration de son bon comportement devant l'évaluateur. Le code est aussi à placer sous Git avant l'entrevue.

Présentation en classe: fichier LibreOffice Impress

> «I am a very bottom-up thinker. If you give me the right kind of Tinker Toys, I can imagine the building. I can sit there and see primitives and recognize their power to build structures a half mile high, if only I had just one more to make it functionally complete. I can see those kinds of things»
>
> *-Ken Thompson, un des développeurs du système d'exploitation Unix et du langage C.*

Introduction

Depuis le début de la session, la très grande majorité des exercices ont porté sur des aspects de la carte mère et du microcontrôleur. Le robot lui-même a peu bougé. Il est temps de passer à l'action! On le fera se déplacer pour qu'il suive une trajectoire préprogrammée inscrite en mémoire externe. De plus, ce laboratoire permettra de mettre à profit les notions introduites lors des travaux pratiques précédents.

Une bonne façon de revoir les concepts vus dans les semaines précédentes est de les utiliser de nouveau, mais dans un contexte plus général. Ce qui est proposé comme travail ressemble un peu à un interpréteur de commandes. Les instructions d'un interpréteur de

commande sont directement exécutées. Des exemples connus d'interpréteurs de commandes sont Bash (Linux, Mac) et PowerShell (Windows).

Ici, nous procéderons un peu différemment. Les instructions seront plutôt compilées sur le PC pour produire un pseudo-code binaire (*bytecode* en anglais). Il s'agit d'octets qui ne peuvent pas être exécutés directement par un processeur, mais qui sont dans un format qui permet une interprétation plus facile puisque les analyses lexicale et syntaxique (*parsing*) du fichier source sont déjà réalisées. Généralement, ce format est indépendant du matériel et est sous une forme qui est difficile à lire pour l'être humain. Un programme en langage Java passe par un pseudo-code binaire lors de son exécution. Le résultat plus compact d'un pseudo-code binaire par rapport au code source d'origine a cependant des avantages en vitesse à l'exécution, pour le web par exemple avec le JavaScript.

Notre pseudo-code binaire sera inscrit dans la mémoire externe de la carte mère du robot. Le ATmega324PA devra le lire et comprendre la signification des octets pour arriver à faire bouger le robot selon ce qui y est inscrit. Le robot ne fait donc pas l'interprétation directe des instructions d'un fichier source ou d'un interpréteur de commandes, mais d'une liste d'octets encodés ayant une signification équivalente.

Passer par un langage spécialisé pour programmer un robot est assez courant. La société FANUC a opté pour cette approche avec le langage Karel un peu différent de l'original. Le département de génie mécanique de l'École Polytechnique de Montréal possède un robot industriel FANUC. Cette approche permet le contrôle du robot mais aussi la simulation et la vérification des opérations prévues pour le robot.

Lors du travail pratique 5, des données ont été écrites et lues en mémoire. La mémoire contenait alors des données. Elle devra maintenant contenir des instructions à exécuter. Les procédures d'accès à la mémoire seront réutilisées ici, mais dans un contexte un peu différent puisqu'elles auront un effet direct sur les mouvements du robot.

Les paragraphes suivants décrivent les instructions et leur équivalent en pseudo-code binaire. Quelques détails sur la façon de les compiler sur le PC sont aussi fournis.

Instructions

Les commandes forment un langage très simple qui ressemble un peu à un assembleur. Les différentes commandes que vous aurez à considérer sont présentées sous la forme d'instructions binaires au tableau 1.

Instruction binaire	mnémonique	Description
0000 0001	dbt	début

Instruction binaire	mnémonique	Description
0000 0010	att	attendre
0100 0100	dal	allumer la DEL
0100 0101	det	éteindre la DEL
0100 1000	sgo	jouer une sonorité
0000 1001	sar	arrêter de jouer la sonorité
0110 0000	mar	arrêter les moteurs
0110 0001	mar	arrêter les moteurs
0110 0010	mav	avancer
0110 0011	mre	reculer
0110 0100	trd	tourner à droite
0110 0101	trg	tourner à gauche
1100 0000	dbc	début de boucle
1100 0001	fbc	fin de boucle
1111 1111	fin	fin

Tableau 1: Jeu d'instructions du robot

On peut prendre un éditeur pour écrire un programme dans ce langage tout comme on le fait pour un programme en C/C++. Vous pouvez appeler le compilateur «progmem» sur votre machine Linux pour le compiler de la façon suivante pour produire un fichier de sortie contenant le pseudo-code binaire:

```
% progmem [-v] -o «fichierDeSortie» «fichierDentree»
```

Comme en C/C++, les instructions doivent être terminées par un point-virgule. Une instruction doit être écrite soit entièrement en minuscule, soit entièrement en majuscule, mais sans

combinaison des deux. Un opérande ayant une valeur entre 0 et 255 peut suivre le terme mnémonique si l'instruction le prévoit. Les symboles %, # et // permettent d'introduire un commentaire sur la ligne. Voici un fichier qui donne un exemple de programme. Ce fichier contient des erreurs. Quand il y a des erreurs, le fichier de sortie n'est pas produit. Un bon exercice pourrait être d'enlever les lignes provoquant les erreurs pour voir comment faire une première compilation et de comprendre les instructions dont les détails sont donnés à la section suivante. L'option -v du programme progmem vous donne plus de détails sur l'opération de production du pseudo-code binaire (très utile).

Pseudo-code binaire (bytecode)

Le pseudo-code binaire sera organisé selon un format uniforme afin de faciliter sa lecture en mémoire. Une instruction est sous forme binaire et comprend 8 bits. Une instruction est toujours suivie d'un opérande de 8 bits. Cet opérande n'est pas significatif si l'instruction n'en prévoit pas. Néanmoins, le fait de le placer dans un format binaire rend l'organisation des octets plus uniforme. Autrement dit, les instructions sont aux adresses paires en mémoire alors que les opérandes (significatifs ou non) sont aux adresses impaires.

Instruction (obligatoire) (8 bits)	Opérande (obligatoire) (8 bits)

Tableau 2: Format des instructions du pseudo-code binaire

La seule exception à cette organisation est pour les deux premiers octets qui forment une donnée de 16 bits. Cette valeur donne le nombre d'octets total du pseudo-code binaire. Par exemple, s'il y a 8 instructions de code source, le fichier binaire aura une taille de 18 octets au total et la lecture des deux premiers octets permet d'obtenir cette information.

Il est toujours possible de consulter les octets du fichier de sortie en pseudo-code binaire avec la commande Linux suivante:

```
% od -v -t x1 «fichierDeSortie»
```

Description des instructions

Voici une description complète des instructions illustrées à l'aide d'exemples.

Début

Mnémonique: dbt
Code binaire: 0x01
Description: Indique le début du code. Le programme commence ici (et pas avant).
Opérande: non
Exemple:

```
att 25 ;# ne sera pas exécuté
dal 2 ;% ne sera pas exécuté
dbt ;# début de l'exécution
att 10 ;# attendre 250 ms
```

Attendre

Mnémonique: att
Code binaire: 0x02
Description: attendre 25 ms fois la valeur de l'opérande
Opérande: oui
Exemple:

```
att 25 ;# ne sera pas exécuté
dal 2 ;% ne sera pas exécuté
dbt ;# début
att 10 ;// attendre 250 ms (instruction exécutée)
```

Allumer les DEL

Mnémonique: dal
Code binaire: 0x44
Description: allumer les DEL dont les bits correspondants de l'opérande sont à un. Si une seule DEL est utilisée, l'opérande est ignoré (mais présent).
Opérande: oui
Exemple:

```
dal 32 ;# allumer la DEL #6
dal 64 ;# allumer la DEL #7
dal 32 ;// ne fera rien puisque la DEL #6 est déjà allumée
dal 0 ;# ne fera rien
```

Éteindre les DEL

Mnémonique: det
Code binaire: 0x45
Description: éteindre les DEL dont les bits correspondants de l'opérande sont à un. Si une seule DEL est utilisée, l'opérande est ignoré (mais présent).
Opérande: oui
Exemple:

```
dal 127 ;# DEL 1, 2, 3, 4, 5, 6 et 7 sont allumées
det 64 ;// éteindre la DEL 7. Les autres restent allumées
det 64 ;# instruction sans effet sur les DEL
det 1 ;% éteindre la DEL 1. Les DEL 2, 3, 4, 5 et 6
allumées
det 0 ;# instruction sans effet sur les DEL
```

Jouer une sonorité

Mnémonique: sgo
Code binaire: 0x48
Description: jouer une sonorité du tableau 3. Il faut activer une sortie en mode PWM avec la fréquence donnée dans le tableau. Si la valeur n'est pas dans le tableau, la commande est ignorée.
Opérande: oui
Exemple:

```
sgo 69 ;# émettre un son à 440 Hz
sgo 200 ;# instruction ignorée
sgo 45 ;// émettre un son à 110 Hz
```

Arrêter de jouer la sonorité

Mnémonique: sar
Code binaire: 0x09
Description: arrêter de jouer la sonorité en cours (s'il y en a une qui joue).
Opérande: non
Exemple:

```
sgo 54 ;% jouer un son à 180 Hz
sar ;# arrêter de jouer un son
sar ;# instruction sans effet
```

Arrêter les moteurs

Mnémonique: mar
Code binaire: 0x60 ou 0x61
Description: arrêter les deux moteurs
Opérande: non
Exemple:

```
...
mar ;# Arrêter
mar ;// instruction sans effet
...
```

Avancer

Mnémonique: mav
Code binaire: 0x62
Description: avancer en ligne droite à une vitesse donnée par (opérande / 255 * 100%)
Opérande: oui
Exemple:

```
mar ;# Le robot est immobile
mav 0 ;% Le robot reste immobile
mav 255 ;# vitesse de 100%
mav 128 ;# vitesse de 50% (50% du PWM)
```

Reculer

Mnémonique: mre
Code binaire: 0x63
Description: reculer en ligne droite à la vitesse (opérande / 255 * 100%)
Opérande: oui
Exemple:

```
mar ;# moteurs ne tournent pas
mre 0 ;// robot reste immobile
mre 255 ;# vitesse maximale vers l'arrière
mre 64 ;# vitesse de 25% vers l'arrière (25% du PWM)
```

Tourner à droite

Mnémonique: trd
Code binaire: 0x64
Description: virage du robot de 90 degrés à droite
Opérande: non

Tourner à gauche

Mnémonique: trg
Code binaire: 0x65
Description: virage du robot de 90 degrés à gauche
Opérande: non

Début de boucle

Mnémonique: dbc
Code binaire: 0xC0
Description: emmagasiner l'adresse du présent point d'exécution du code dans une variable pour pouvoir y revenir. De plus, créer une variable qui conservera l'état du compteur de boucle. La boucle s'exécutera une fois de plus que la valeur précisée par l'opérande. Donc, si l'opérande est zéro, le code ne s'exécutera qu'une seule fois. Il ne peut y avoir deux boucles imbriquées (actives au même moment).
Opérande: oui
Exemple: voir l'instruction «fin de boucle» pour un exemple

Fin de boucle

Mnémonique: fbc
Code binaire: 0xC1
Description: si le compteur n'est pas égal à 0, retourner à l'adresse de début de boucle et décrémenter le compteur de boucle. Sinon, ne rien faire.
Opérande: non
Exemple:

```
...
dbc 100 ;// la boucle s'exécutera 101 fois
```

```
dal 1 ;# allumer la DEL en position 1
att 1 ;# attendre 25 ms
det 1 ;// éteindre la DEL en position 1
att 1 ;# attendre 25 ms
fbc ;# boucler à l'instruction dbc précédente
...
dbc 0 ;# faire le code suivant une fois
dal 1 ;% allumer la DEL en position 1
att 1 ;# attendre 25 ms
det 1 ;% éteindre la DEL en position 1
att 1 ;# attendre 25 ms
fbc ;// boucler à l'instruction dbc précédente
...
```

Fin

Mnémonique: fin
Code binaire: 0xFF
Description: fin de l'exécution du code. Le code qui suit cette instruction sera ignoré. Aucun son ne sera émis après cette instruction et le robot s'arrêtera également.
Opérande: non
Exemple:

```
att 25 ;# attendre 625 ms
fin ;% fin du programme et robot immobile
att 25 ;# instruction qui ne sera même pas lue
```

Fréquences des notes MIDI

Si vous êtes musicien, vous pouvez relier la fréquence d'un signal PWM et les notes d'une chanson pour rendre le comportement de votre robot plus agréable. Si vos connaissances en musique sont plus limitées, votre robot devra se contenter d'émettre des sons quelconques. Cette remarque n'a aucune conséquence sur quelque forme d'évaluation que ce soit. Par contre, il est toujours bon de connaître la relation entre la fréquence d'un signal et les notes de musique. Plus d'informations sont disponibles sur Wikipédia pour l'interface MIDI, et le cycle des notes. Voici un tableau des notes MIDI et des fréquences correspondantes.

Note (donnée)	Fréquence (en Hz)	Période (en ms)	Temps actif (en ms)
45	110	9.09090909090909	4.54545454545455
46	116.5409404	8.58067556983523	4.29033778491762

Note (donnée)	Fréquence (en Hz)	Période (en ms)	Temps actif (en ms)
47	123.4708253	8.0990792558233	4.04953962791165
48	130.8127827	7.64451286594282	3.82225643297141
49	138.5913155	7.21545932713006	3.60772966356503
50	146,832384	6.810486713076	3.405243356538
51	155.5634919	6.42824346533392	3.21412173266696
52	164.8137785	6.06745388259235	3.03372694129618
53	174.6141157	5.72691386315858	2.86345693157929
54	184.9972114	5.40548688637651	2.70274344318825
55	195.997718	5.1021002195874	2.5510501097937
56	207.6523488	4.81574133799616	2.40787066899808
57	220	4.54545454545455	2.27272727272727
58	233.0818808	4.29033778491762	2.14516889245881
59	246.9416506	4.04953962791001	2.02476981395501
60	261.6255653	3.82225643297141	1.91112821648571
61	277.182631	3.60772966356373	1.80386483178186
62	293.6647679	3.405243356538	1.702621678269
63	311.1269837	3.21412173266593	1.60706086633296
64	329.6275569	3.03372694129526	1.51686347064763
65	349.2282314	2.86345693157929	1.43172846578964
66	369.9944227	2.70274344318825	1.35137172159413
67	391.995436	2.55105010979435	1.27552505489718

Note (donnée)	Fréquence (en Hz)	Période (en ms)	Temps actif (en ms)
68	415.3046976	2.40787066899866	1.20393533449933
69	440	2.27272727272727	1.13636363636364
70	466.1637615	2.14516889245835	1.07258444622917
71	493.8833013	2.02476981395542	1.01238490697771
72	523.2511306	1.91112821648571	0.955564108242853
73	554.365262	1.80386483178219	0.901932415891094
74	587.3295358	1.702621678269	0.8513108391345
75	622.2539674	1.60706086633296	0.803530433166482
76	659.2551138	1.51686347064786	0.758431735323929
77	698.4564629	1.43172846578964	0.715864232894822
78	739.9888454	1.35137172159394	0.675685860796972
79	783.990872	1.27552505489701	0.637762527448506
80	830.6093952	1.20393533449918	0.601967667249592
81	880	1.13636363636364	0.568181818181818

(source: http://www.tonalsoft.com/pub/news/pitch-bend.aspx)

Tableau 3: fréquences des notes MIDI

Travail à effectuer

Le travail à faire est en deux parties. Il faut d'abord écrire le code qui permet au robot de lire le *bytecode* en mémoire externe et de l'interpréter correctement. Le problème se découpe bien et le travail peut être réparti de façon à ce que chacun des membres de l'équipe puisse contribuer à l'architecture du code autant qu'à la programmation.

Nous vous suggérons quelques trucs pour la mise au point des systèmes logiciels et matériels. Ces conseils vous seront fort utiles puisque la quantité de code à écrire augmentera d'ici la fin de la session.

Avant d'effectuer l'interprétation proprement dite, il serait souhaitable que le robot effectue une petite séquence de démarrage. Ce peut être, par exemple, de faire clignoter la DEL 2 fois rouge et 2 fois vert au départ. De cette façon, on sait que le robot a démarré et qu'il est dans une phase connue très rapidement. Quand les piles sont faibles ou qu'il y a un problème sérieux, on peut le réaliser tout de suite si le robot ne traverse même pas sa séquence de démarrage.

Une deuxième partie consiste à écrire des instructions dans le langage présenté plus haut, de les compiler avec progmem sur le PC, et de les charger en mémoire externe sur la carte mère. Vous êtes libre d'écrire les instructions que vous désirez. Par contre, chaque instruction doit être utilisée au moins une fois. Essayez d'écrire un ensemble d'instructions qui donne une belle chorégraphie à votre robot!

Il vous faudra utiliser l'interface RS-232 pour écrire votre *bytecode* en mémoire en écrivant un petit programme utilitaire pour effectuer cette tâche. Ce programme d'écriture est donc différent de celui qui ira lire les octets en mémoire. Il faudra peut-être ajuster quelque peu le code inscrit pour le problème 3 de la semaine 5 pour supporter le transfert des octets à charger en mémoire du PC vers la carte. Normalement, ce travail devrait se faire facilement en rajoutant une routine de transfert. Vous n'aurez qu'à ajuster la configuration du périphérique interne de l'AVR pour permettre la réception. Le programme serieViaUSB peut être appelé pour transmettre le *bytecode* vers la carte par RS-232 via le câble USB avec la commande suivante:

```
% serieViaUSB -e -f «fichierDeSortie»
```

Il faudra charger le fichier de sortie complet en mémoire externe (les instructions utilisées autant que celles inutilisées et les deux premiers octets qui donnent la dimension du programme). Une fois le fichier de sortie inscrit en mémoire externe sur la carte, il faudra charger le programme de lecture et d'interprétation des octets en mémoire. Il faudra peut-être osciller quelques fois entre ces deux programmes à charger dans le ATmega324PA lors de la mise au point. Il s'agit d'un inconvénient qui peut être un peu gênant. Par contre, les combiner en un seul programme n'est pas évident (bien que possible) et n'est pas exigé, ni même recommandé.

Les derniers détails concernent les sonorités à jouer. Il faudra brancher une sortie PWM au fil rouge du piézoélectrique de votre robot. Le fil noir devra être connecté à une broche adjacente qu'il faudra configurer en sortie à la valeur constante de zéro volt. C'est une façon de simuler une mise à la masse sur ce fil. Il est à noter que les fils rouge et noir pourraient

être inversés. Le piézoélectrique n'a pas réellement de polarité. Cependant, puisque la carte mère sera bientôt passablement garnie de fils, il est souhaitable de conserver un peu d'ordre dans les couleurs. Autre petit détail important. Il est vrai qu'il faut un signal ayant une forme PWM pour générer un son en sortie. Par contre, le mode CTC d'une minuterie peut être utilisé pour générer cette onde. Ce mode d'opération est différent de celui utilisé pour faire fonctionner les moteurs. On notera aussi qu'il devrait être possible de jouer une sonorité pendant que le robot se déplace.

Suivi

Pour la semaine prochaine, avant l'entrevue de laboratoire:

- Avoir un robot fonctionnel pour l'entrevue de laboratoire.
- Avoir rempli le formulaire d'évaluation par les pairs si ce n'est pas déjà fait. Normalement, le formulaire devrait être rendu à la deuxième séance de laboratoire de la semaine 9.
- Avoir un nom pour votre robot inscrit sur la première page de votre évaluation par les pairs quelque part.
- Votre code placé sous Git dans le répertoire `https://githost.gi.polymtl.ca/git/inf1900-WXYZ/tp/tp9/`
- Par principe d'équité, il ne sera plus possible de déposer une nouvelle version du code dans l'entrepôt Git après le début des entrevues. L'heure de l'entrevue de la première équipe (toutes sections confondues) dans l'horaire fixe l'heure limite du dernier dépôt dans l'entrepôt pour toutes les équipes du cours. Donc, le mercredi 20 mars 8h15 AM maximum.