Macro-opérations de calcul

Noyau exécutif SynDEx pour le microcontrôleur PIC 18F2680

Table des matières

1. Objet du document

Ce document décrit les macro-opérations disponibles dans le module calcul pour le microcontrôleur PIC18F2680 ainsi que la chaîne de compilation utilisée pour générer du code exécutable par ce microcontrôleur. Une dernière section traite des incompatibilités relevées dans le fichier *syndex.m4x.*

2. Définitions

On définit les termes suivants pour l'ensemble du document :

- *syndex5.m4x* désigne le fichier *syndex.m4x* livré avec les versions 5 de SynDEx,
- *syndex6.m4x* désigne le fichier *syndex.m4x* livré avec les versions 6 de SynDEx,
- *opération* signifie *macro-opération*.

3. Les macro-opérations

Cette section présente les opérations de calcul définies dans le fichier *pic18f2680.m4x*. L'encadré présente le nom de l'opération générique. Le texte de présentation traite de l'opération appelée par l'opération générique pour générer le code source spécifique au processeur cible. Le nom de l'opération spécifique est précisé entre parenthèses si nécessaire.

3.1. Les types

```
typedef_(typeName,typeSize)
```

Les types *bool*, *char*, *short*, *int* sont définis et utilisables.

3.2. L'allocation mémoire

```
alloc_(1type,2newLabel[,3size[,4memBank]])
```

L'opération *alloc_* (*basicAlloc_*) permet d'allouer des cellules mémoires pour des tableaux de dimension 1 de données de type *bool*, *char*, *short* et *int*.

Par la suite, lorsqu'on notera *cellule mémoire* ou plus simplement *mémoire*, on supposera celle-ci préalablement définie par l'opération *alloc_*.

3.3. Renommage d'une mémoire

```
alias_(newLabel,oldLabel[,offset=0[,size=oldLabel_size_-offset]])
```

L'opération *alias_* (*basicAlias_*) permet de renommer une cellule mémoire.

3.4. Initialisation d'une cellule mémoire

```
zero_(destLabel)
```

L'opération *zero_* permet d'initialiser (à zéro) une cellule mémoire.

3.5. Transfert d'une cellule mémoire

```
copy_(dest,srce[,size=srce_size_])
```

L'opération *copy_* (*basicCopy_*) transfère le contenu de la cellure mémoire *srce* dans la cellule mémoire *dest*.

3.6. Les structures de contrôle

```
if_(bool)
```

L'opération *if_* (*basicIf_*) réalise un saut conditionné par la valeur de la cellule mémoire *bool*.

ifnot_(bool)

L'opération *ifnot_* (*basicIfnot_*) réalise un saut (inversement) conditionné par la valeur de la cellule mémoire *bool*.

else_()

L'opération *else_* (*basicElse_*) est utilisée en conjonction avec les opérations *if_* et *ifnot_* pour exprimer une prise de décision.

endif_()

L'opération *endif_* (*basicEndif_*) termine une structure de contrôle débutée par une opération *if_* ou *ifnot_*.

switch_(buffer)

L'opération *switch_* (*basicSwitch_*) débute une structure de prise de décision à choix multiples.

case_(value)

Chacun des choix possibles est étiqueté par une constante entière précisée par une opération *case_ (basicCase_)*.

endcase_()

L'opération *endcase_ (basicEndcase_)* conclue une séquence introduite par une opération *case_*.

endswitch_()

L'opération *endswitch_* (*basicEndswitch_)* termine une structure de contrôle débutée par une opération *switch_*.

loop_()

L'opération *loop_* (*basicLoop_*) initie une boucle dont le nombre d'itérations dépend de la macro *NBITERATIONS*. Si cette dernière n'est pas définie, la boucle est infinie sinon le nombre d'itérations est fixé par la macro. Le nombre maximal d'itérations est fixé à 255.

endloop_()

L'opération *endloop_* (*basicEndloop_*) termine une structure de contrôle débutée par une opération *loop_*.

3.7. Les opérations de synchronisation

```
semaphores_({semFull,semEmpty,})
```

L'opérations *semaphores_* réserve l'espace mémoire nécessaire au stockage des semaphores utilisées.

```
Pre0_(s)
```

L'opération *PreO_* réalise une précédence communication de priorité 1->0.

```
Suc0_(s)
```

L'opération *SucO_* réalise une précédence calcul de priorité 0<-1.

```
Pre1_(s)
```

L'opération *Pre1_* réalise une précédence calcul de priorité 0->1 ou 1->1.

```
Suc1_(s)
```

L'opération *Suc1_* réalise une précédence communication de priorité 0<-1 ou 1<-1.

3.8. La séquence de communication

```
thread_(mediaType,mediaName {, procrName})
```

L'opération *thread_* (*basicthread_*) indentifie une séquence de communication à lancer en (pseudo)parallèle avec la séquence principale de calcul.

```
endthread_()
```

L'opération *endthread_* (*basicEncthread_*) termine une séquence de communication initiée par l'opération *thread_*.

3.9. La séquence de calcul

```
spawn_thread_(mediaName)
```

L'opération *spawn_thread_* est associée à une opération *thread_* pour lancer une séquence de communication.

```
wait_endthread_(mediaName)
```

L'opération *wait_endthread_* attend la fin d'exécution d'une séquence de communication.

```
main_()
```

L'opération *main_* identifie la séquence de calcul.

```
endmain_()
```

L'opération *endmain_* termine la séquence de calcul.

3.10. Les opérations unaires

```
gnot(logic[x]?1, logic[x]!2)
```

L'opération *gnot* réalise une négation logique d'un entier.

```
gneg(arith[x]?1, arith[x]!2)
```

L'opération *gneg* retourne l'opposé d'un entier.

3.11. Les opérations binaires

```
gand(logic[x]?1, logic[x]?2, logic[x]!3)
gor(logic[x]?1, logic[x]?2, logic[x]!3)
gxor(logic[x]?1, logic[x]?2, logic[x]!3)
```

Les opérations *gand, gor* et *gxor* réalisent respectivement les opérations et, *ou inclusif* et *ou exclusif* sur des entiers.

```
gadd(arith[x]?1, arith[x]?2, arith[x]!3)
gsub(arith[x]?1, arith[x]?2, arith[x]!3)
```

Les opérations *gadd* et *gsub* réalisent respectivement l'addition et la soustraction d'entiers.

```
gmul(arith[x]?1, arith[x]?2, arith[x]!3)
```

L'opération *gmul* réalise une multiplication d'entiers de type *char* et *short*. La fonction retourne un résultat entier de type *short*, ou *int,* lors de la multiplication de deux entiers de type *char*, ou respectivement *short*.

```
gdiv(char[x]?1, char[x]?2, char[x]!3, char[x]!4, bool !$5)
```

L'opération *gdiv* réalise une division de deux entiers de type *char*. Elle retourne deux entiers, le quotient et le reste de la division, ainsi qu'un booléen signalant une erreur.

3.12. Les opérations de comparaison

```
gEQ(types?1, types?2, bool!o)
gNE(types?1, types?2, bool!o)
```

Les opérations *gEQ* et *gNE* effectuent respectivement les comparaisons *égale* et *non-égale* sur deux entiers.

```
gLT(types?1, types?2, bool!o)
gNL(types?1, types?2, bool!o)
```

Les opérations *gLT* et *gNL* effectuent respectivement les comparaisons *inférieur-strict* et *supérieur-ou-égal* sur deux entiers.

```
gGT(types?1, types?2, bool!o)
gNG(types?1, types?2, bool!o)
```

Les opérations *gGT* et *gNG* effectuent respectivement les comparaisons *supérieur-strict* et inf*érieur-ou-égal* sur deux entiers.

```
gequal(types?1, types?2, bool!o)
gnotequal(types?1, types?2, bool!o)
gless(types?1, types?2, bool!o)
gnotless(types?1, types?2, bool!o)
```

Les opérations *gequal*, *gnotequal*, *gless* et *gnotless* sont respectivement équivalentes aux opérations *gEQ*, *gNE*, *gLT* et *gNL*.

4. La chaîne de compilation

Le fichier *pic18f2680.m4m* décrit la chaîne de compilation utilisée pour générer et assembler du code source pour le microcontrôleur PIC18F2680.

4.1. Outils logiciels

La chaîne de compilation pour le microcontrôleur PIC18F2680 est basée sur les outils fournis par son fabriquant, Microchip, associés à des outils disponibles librement : *make* et *m4*. La système d'exploitation supportant la chaîne de compilation est nécessairement Windows XP.

L'environnement de développement intégré *MPLAB* pour les microcontrôleurs PIC est disponible gratuitement sur le site de Microchip : www.microchip.com.

La version Windows de l'exécutable *make* utilisée est fournie librement sur le site www.mingw.org. L'environnement *mingw* est utilisé pour compiler une version Windows du programme *m4* à partir des sources disponibles sur le site www.gnu.org/software/m4

4.2. Description succincte

Le fichier de macro-code généré par SynDEx pour le microcontrôleur est, dans un premier temps, traduit en langage assembleur par le macro-assembleur m4 en utilisant les fichiers *syndex.m4x et pic18f2680.m4x*. Il est ensuite assemblé par l'exécutable *mpasmwin* qui génére un fichier au format *Intel Hex32*.

5. Imcompatibilités du fichier syndex.m4x

Cette section décrit des incompatibilités relevées dans le fichier *syndex6.m4x* qui rendent impossible la génération de code source compatible avec l'assembleur utilisé par la chaîne de compilation.

Ces incompatibilités semblent être apparues lors du passage de la version 5 à la version 6 de SynDEx. Le fichier *syndex5.m4x* proposant des versions correctes des opérations mentionnées.

5.1. La macro processor_

La macro *processor_* du fichier syndex6.m4x a fait l'objet d'une modification qui la rend incompatible avec tout autre langage ne disposant pas de la syntaxe de commentaire (/**/) définie par le langage C : la séquence d'appels actuelle rend impossible la redéfinition de la macro *comment_* (qui génére par défaut des commentaires à la C /* */) dans le fichier m4x du processeur cible.

5.2. La macro thread_

De même que pour la macro *processor_*, le commentaire /* media type = $1 media name = $2 */ ajouté dans la macro *thread_* du fichier syndex6.m4x utilise une syntaxe propre au langage C.