TP1 - Architecture des systèmes à processeurs - assembleur M32C/87

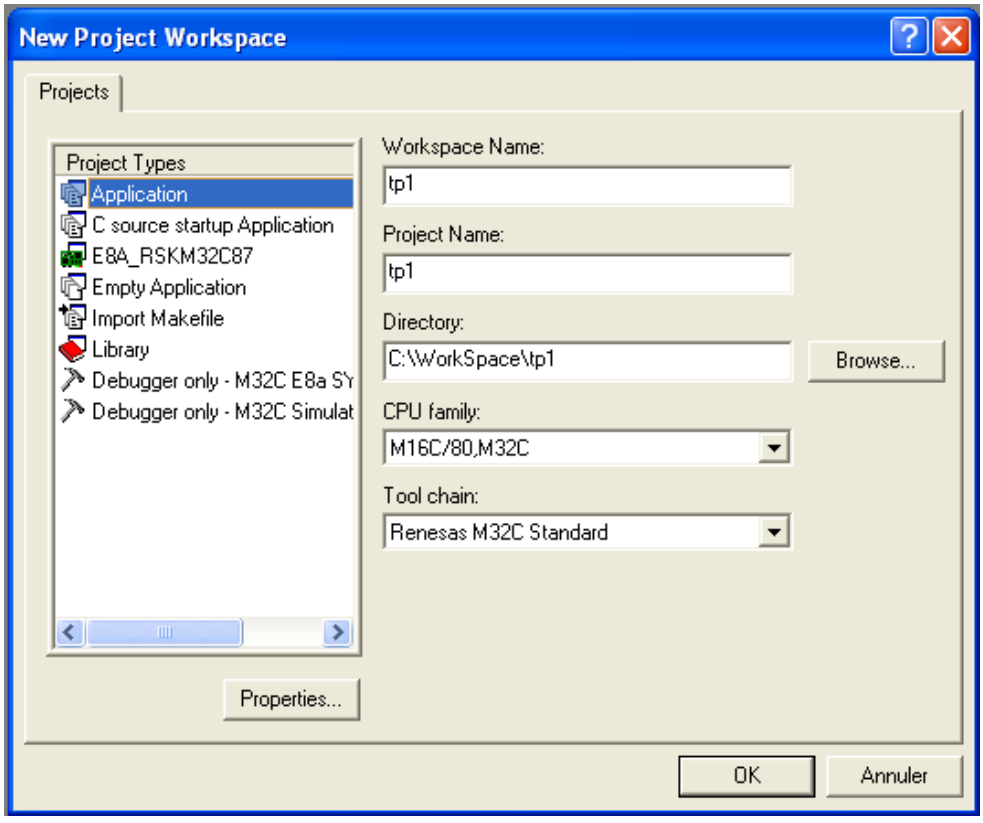
Christophe BLANC
Université Blaise Pascal
IUT GEII - S2
Email: christophe.blanc@lasmea.univ-bpclermont.fr
Site : www.christophe-blanc.info

18 mars 2009

1 Préparation

Lire le guide d'utilisation et faire fonctionner le RSK[1]. Dans ce tp, pour ne pas avoir à utiliser le RSK, nous pourrons utiliser un simulateur.

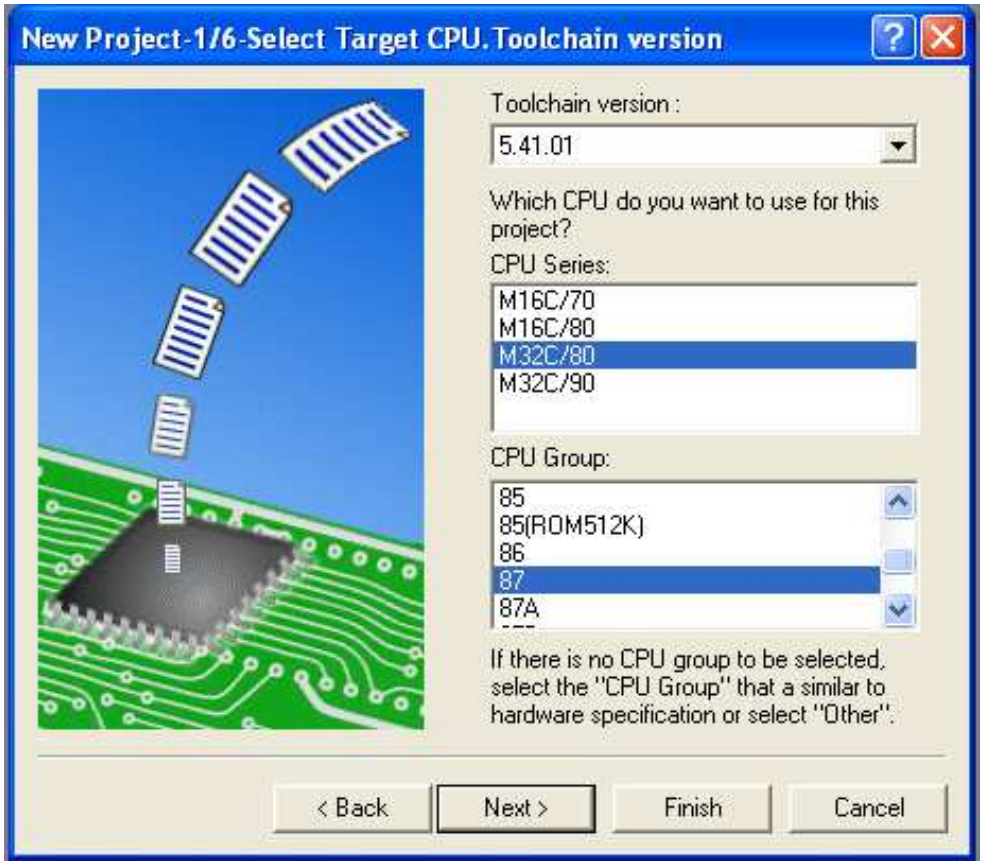
- Créer un nouveau projet en respectant les options de la figure ci-dessous. Attention, vous n'avez pas le droit d'écrire sur le disque C :. Utilisez plutôt votre répertoire de travail sur le serveur (chemin sous l'option 'Directory').



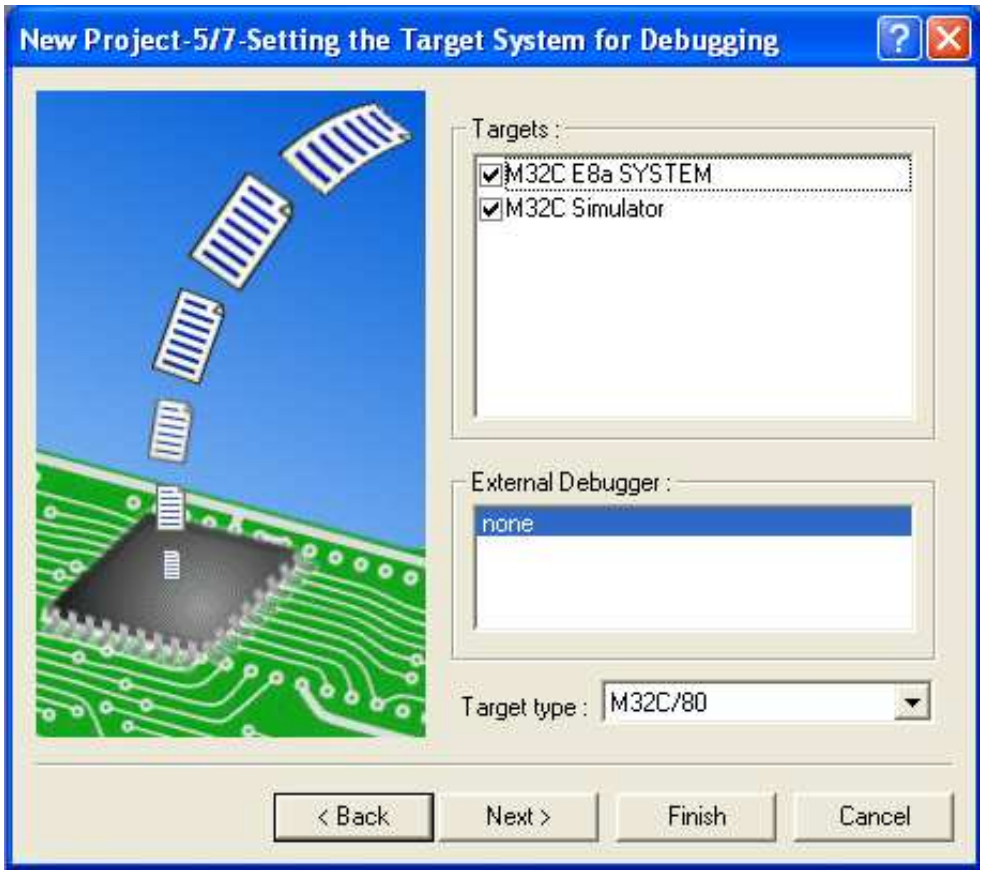
- Cliquez sur le bouton <OK> une fois les options définies

Une fenêtre pour la création du projet apparaît. Remplissez les options comme indiquées sur la figure ci-dessous et cliquez sur <Next>.

[1] Renesas Starter Kit

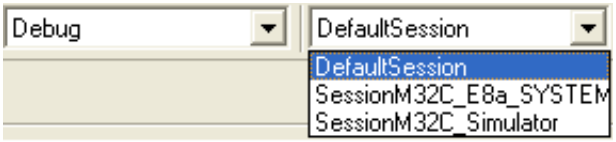


– Cliquez sur <Next> jusqu'à ce que la fenêtre "New Project - 5/6 - Setting The Target System for Debugging" apparaisse.
– Sélectionnez les deux cibles comme indiqué sur la figure ci-dessous (on pourra travailler soit avec le RSK soit avec le simulateur).

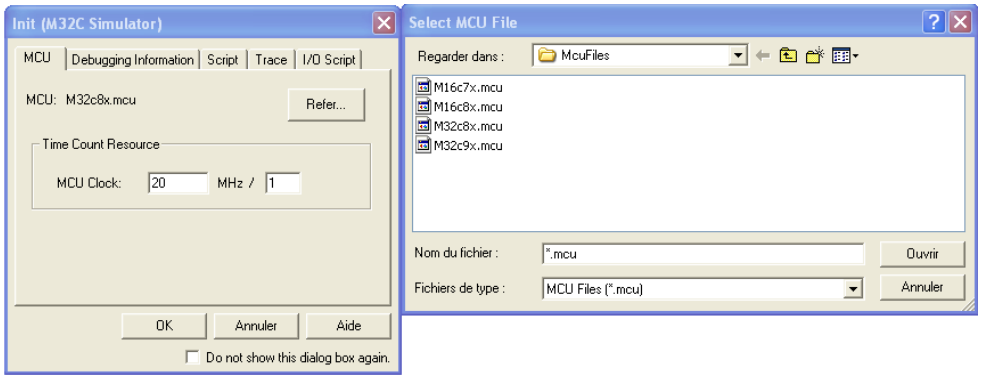


– Cliquez sur <Next> trois fois et ensuite sur <Finish> pour créer votre projet.

Vous pouvez changer de cible en la sélectionnant comme indiqué sur la figure ci-dessous.



- RSK SessionM32C_E8a_SYSTEM : reporter vous au guide d'utilisation rapide pour remplir les options
- Simulateur SessionM32C_Simulator : cliquez sur <refer> et sélectionnez le fichier "M32c8x.mcu"



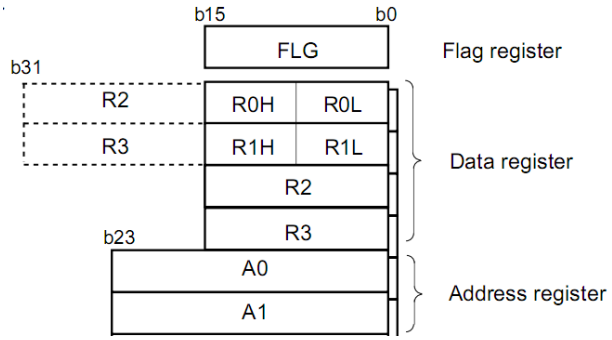
- Cliquez sur <OK>

Nous pouvons maintenant écrire notre premier programme en assembleur M32C/87 dans la fonction "void main(void)" du fichier "tp1.c". Pour écrire des instructions en assembleur, il faut les placer entre les deux directives #pragma comme indiqué ci-dessous.
Exemple : instruction mov

```
void main(void)
{
    #pragma ASM
        mov.l:g #08h,R2R0
    #pragma ENDASM
}
```

2 But

Les microprocesseurs de la famille Renesas M32C/80 ont une architecture comportant 4 registres de données (R0 à R3), 2 registres d'adresses (A0 et A1) et un registre d'état (FLG).



Les calculs utilisent une unité arithmétique et logique (U.A.L) fonctionnant suivant le principe décrit en cours. La taille des opérandes peut être de 8 bits (.B Byte), 16 bits (.W Word) ou 32 bits (.L Long Word).
L'opération arithmétique étant réalisée, le registre d'état (FLG - Flag Register) est mis à jour en fonction du résultat. Il comporte entre autres 4 indicateurs. La fonction assurée par chaque indicateur est :

- Bit 0 : Carry Flag (C Flag) indique une retenue en arithmétique non signée,
- Bit 2 : Zero Flag (Z Flag) signale que le résultat est nul,

– Bit 3 : Sign Flag (S Flag) révèle que le résultat est négatif,
– Bit 5 : Overflow Flag (O Flag) indique en débordement en arithmétique signée.

Le programme source assembleur suivant permet :

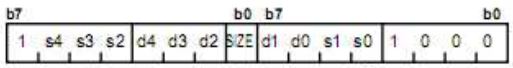
– d'utiliser les commandes élémentaires nécessaires pour faire la mise au point de programme,
– d'observer le résultat de l'exécution d'instructions,
– d'approfondir l'arithmétique signée en code complément à deux.

Etiquette	Opération	Opérande	Commentaires
_FPTJ	ADD.B :G	R0l,R1l	//R1l< −R1l+R0l
	JMP	_FPTJ	//Branchement à l'adresse symbolique _FPTJ

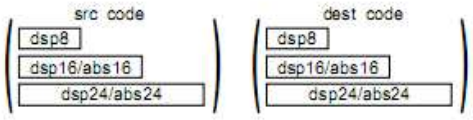
Le codage de l'instruction ADD.size :G Src,Dest est :

ADD

(6) ADD.size:G src, dest



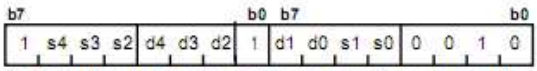
*1 For indirect instruction addressing, the following number is added at the beginning of code:
01000001 when src is indirectly addressed
00001001 when dest is indirectly addressed
01001001 when src and dest are indirectly addressed



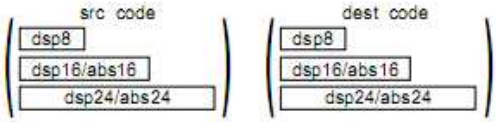
.size	SIZE
.B	0
.W	1

src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	R0L/R0/---	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	R1L/R1/---	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	R0H/R2/-	1 0 0 0 0	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	R1H/R3/-	1 0 0 0 1		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

(7) ADD.L:G src, dest



*1 For indirect instruction addressing, the following number is added at the beginning of code:
01000001 when src is indirectly addressed
00001001 when dest is indirectly addressed
01001001 when src and dest are indirectly addressed



src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0	src/dest		s4 s3 s2 s1 s0 d4 d3 d2 d1 d0
Rn	---/---/R2R0	1 0 0 1 0	dsp:8[SB/FB]	dsp:8[SB]	0 0 1 1 0
	---/---/R3R1	1 0 0 1 1		dsp:8[FB]	0 0 1 1 1
	---/---/-	- - - - - - - - - -	dsp:16[An]	dsp:16[A0]	0 1 0 0 0
	---/---/-	- - - - - - - - - -		dsp:16[A1]	0 1 0 0 1
An	A0	0 0 0 1 0	dsp:16[SB/FB]	dsp:16[SB]	0 1 0 1 0
	A1	0 0 0 1 1		dsp:16[FB]	0 1 0 1 1
[An]	[A0]	0 0 0 0 0	dsp:24[An]	dsp:24[A0]	0 1 1 0 0
	[A1]	0 0 0 0 1		dsp:24[A1]	0 1 1 0 1
dsp:8[An]	dsp:8[A0]	0 0 1 0 0	abs16	abs16	0 1 1 1 1
	dsp:8[A1]	0 0 1 0 1	abs24	abs24	0 1 1 1 0

3 Travail à réaliser

1. Etablir le programme objet correspondant codé en binaire puis en hexadécimal. Le code objet de l'instruction JMP.B _FPTJ est : $BBFD

2. Sous HEW, copier les instructions assembleur dans la fonction "void main(void)" du fichier "tp1.c" entre les deux directives "#PRAGMA ASM" et "#PRAGMA ENDASM".
3. Compilez le programme (cible : SessionM32C_Simulator) : 0 Errors, 0 Warnings
4. Connectez vous à la cible et downloader le programme sur la cible.
5. Placez un point d'arrêt sur la ligne "_FPTJ : add.b :G R0l,R1l"
6. Retrouver le code objet de la question 1 en utilisant la commande "Disassembly" du menu "View"
7. Retrouver le code objet de la question 1 en utilisant la commande View− >CPU− >Memory
8. Cliquez sur <Reset Go>
9. En utilisant, le bouton <Step Over> expliquer le fonctionnement du programme
10. Pour afficher le contenu des registres du microprocesseur, cliquez sur View− >CPU− >Registers
11. Initialiser (en cliquant 2x sur le nom du registre) le contenu des registres avec les valeurs définies dans le tableau ci-dessous. Puis faire exécuter le programme en pas à pas en notant précisément :
    – le contenu des registres R0 et R1
    – le contenu du registre d'état
12. Interpréter les résultats en particulier le contenu du registre d'état.
    Valeurs initiales des registres R0 et R1 pour l'addition sur 8 bits :

R0	$00	$20	$A0	$50	$90	$20A0
R1	$00	$10	$10	$40	$80	$1010

13. Reprendre les mêmes manipulations en utilisant une addition sur 16 bits (ADD.W :G) puis sur 32 bits (ADD.L :G)
    Valeurs initiales des registres R0 et R1 pour l'addition sur 16 bits :

R0	$2000	$00A0	$7050	$9090
R1	$1000	$0010	$2040	$8080

    Valeurs initiales des registres R2R0 et R3R1 pour l'addition sur 32 bits :

R2R0	$F0000000	$20000000	$000000A0	$00007050	$00009090
R3R1	$10000000	$10000000	$00000010	$00002040	$00008080

14. Conclusions