TRAVAUX DIRIGES

ASSEMBLEUR

Une image contenant texte, Police, Graphique, capture d’écran

Description générée automatiquement

Table des matières

[Objectifs & Consignes 3](#_Toc194958710)

[Assembleurs généralités 3](#_Toc194958711)

[Langages de différents niveaux 3](#_Toc194958712)

[Rappels du cours 4](#_Toc194958713)

[1. Quelques principes de base du Motorola 68K 5](#_Toc194958714)

[2. Représentation des nombres entiers 5](#_Toc194958715)

[3. Les registres du Motorola 68000 6](#_Toc194958716)

[3.1. Les registres des données 6](#_Toc194958717)

[3.2. Les registres d’adressage 6](#_Toc194958718)

[3.3. Le registre d’adressage A7 6](#_Toc194958719)

[3.4. Le compteur de programme et le registre d’état 7](#_Toc194958720)

[4. Principales instructions du Motorola 68000 7](#_Toc194958721)

[4.1. Instructions arithmétiques et logiques 7](#_Toc194958722)

[4.2. Instructions arithmétiques 7](#_Toc194958723)

[4.3. Instructions logiques 10](#_Toc194958724)

[4.4. Instructions de décalage et de branchement 12](#_Toc194958725)

[4.5. Instructions de décalage 12](#_Toc194958726)

[4.6. Instructions de branchement 14](#_Toc194958727)

[4.7. Branchements et conditions CCR 16](#_Toc194958728)

[4.8. Les différentes conditions des branchements 33](#_Toc194958729)

[4.9. Mode d’adressage 40](#_Toc194958730)

[4.10. Instructions de transfert 40](#_Toc194958731)

[4.11. Opérations sur Opérandes, Comparaisons, Sauvegardes 41](#_Toc194958732)

[5. LE LOGICIEL EASY68K 47](#_Toc194958733)

[5.1. L’environnement Easy68K 47](#_Toc194958734)

[5.2. Les fichiers manipulés 49](#_Toc194958735)

[5.3. Description d’une ligne de code 49](#_Toc194958736)

[5.4. Edition, compilation et simulation 50](#_Toc194958737)

[5.5. Simulation du programme 51](#_Toc194958738)

[6. Travail demandé 53](#_Toc194958739)

# Objectifs & Consignes

Le but de ces travaux dirigés est de manipuler l’assembleur de Motorola, le modèle 68000.

Après une première phase de rappels des principes du langage assembleur, ces travaux dirigés présentent l’environnement de programmation 68000 de Motorola.

Concernant l’application Easy68K, il faudra télécharger le logiciel, l’installer pour réaliser les exercices demandés par sur cet environnement.

Pour la notation, il sera demandé de mettre des captures d’écran de vos simulations. La note maximale de 20/20 ne sera pas attribuée car vous effectuerez certainement ce travail avec l’aide d’intelligence extérieure ou en groupe. Cependant, certaines questions issues de ce travail vous seront posées lors de l’évaluation finale qui sera un devoir sur table au format papier. Votre intérêt est donc de travailler avec intelligence 😊 et de connaitre les grandes lignes de ce travail!

Enfin, ce travail est nominatif et devra être rendu sur MOODLE en fin de séance. Tous dépassements de limite compteront comme un zéro sur la moyenne générale.

# Assembleurs généralités

L'assembleur est un langage de programmation constitué d'instructions mnémoniques qui représentent des actions, permettant de manipuler directement les valeurs et d'interagir avec le matériel, offrant ainsi une rapidité supérieure aux langages de haut niveau.

Ces travaux/cours présente le langage utilisé par l’assembleur de Motorola 68000.

Les humains programment les machines dans des langages plus haut niveau.

# Langages de différents niveaux

1. langage machine : **01111111 11001010 01001000**
2. langage assembleur : **ADD, R0, R0, #3** : ajoute le contenu de R0 au nombre 3, stocke le dans R0.
3. langage haut niveau : C, Rust etc. : on peut contrôler directement la mémoire (virtuelle) et piloter du matériel. La majorité des pilotes matériels sont écrits en C.
4. langages haut niveau (python, javascript) : pas d’accès direct au matériel.
5. donc, pour contrôler du matériel directement, un humain écrit en assembleur. Un programme (‘l’assembleur’) le compile en langage machine.
6. Un compilateur est un programme qui traduit du langage haut niveau en langage machine.

**Ex : gcc (Linux) compile du C en langage machine.**Exemples d’instructions en assembleur

* **SUB R1, R0, #30**

réalise la soustraction R0 - 30 et stocke le résultat dans R1.

* **LDR R0, 70**

Place le contenu de la mémoire 70 dans le registre R0

* **STR R0, 123**

Stocke le contenu de R0 à l’adresse mémoire 123.

* **CMP R0, #24**

Compare le registre R0 et le nombre 24

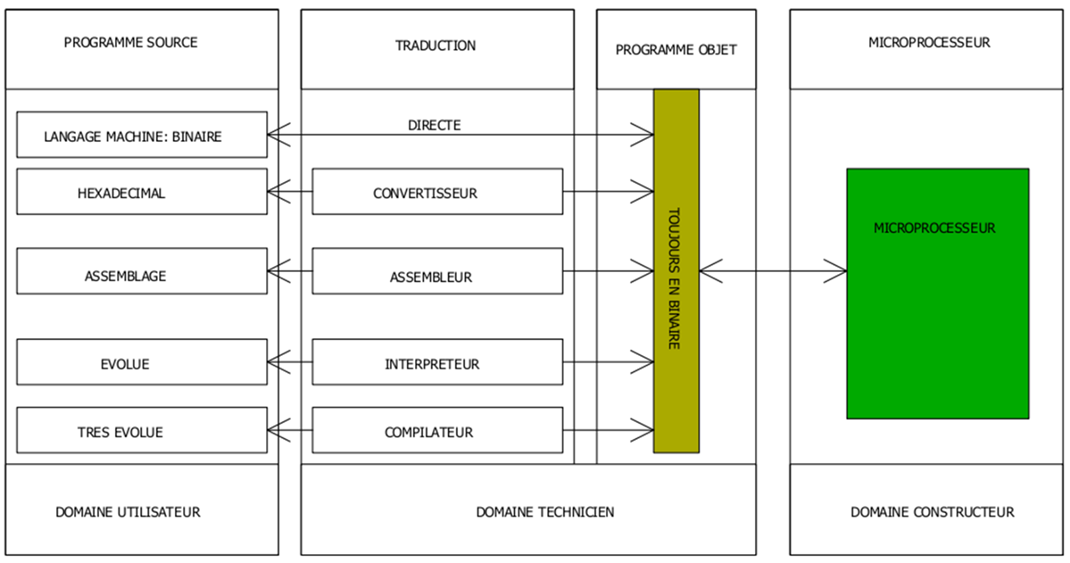
* **BEQ label**

Deux parties :

* BEQ : Branch Equal (BNE, BGT, BLT etc.) :
* label : un label = une adresse spécifique de la mémoire.

Si la dernière ‘comparaison’ est égale, continue au label Sans “branchement”, la machine exécute les instructions en allant d’une adresse à la suivante.

# Rappels du cours



# Quelques principes de base du Motorola 68K

Le Motorola 68000 est un microprocesseur de 16/32 bits introduit en 1979, caractérisé par un ensemble d'instructions de 32 bits, un bus de données externe de 16 bits et un bus d'adresses de 24 bits, facilitant ainsi la programmation sans segmentation de la mémoire.

Les instructions machines sont propres à chaque processeur (heureusement, il existe des principes communs, de grandes familles).

# Représentation des nombres entiers

Le 68000 ne traite que des nombres entiers. Pour les nombres réels, on a recours à :

* un coprocesseur mathématique (FPU), par exemple le 68881
* une bibliothèque d ’émulation logicielle par Développement Limité

Le format d ’un nombre réel est normalisé : IEEE 754 (simple ou double précision)

|  |  |  |
| --- | --- | --- |
| **Nombres entiers Non Signés (NS) :** | | |
| **bits** |  |  |
| 8 | $00 à $FF (0 à 255) | .B Byte |
| 16 | $0000 à $FFFF (0 à 65535) | .W Word |
| 32 | $00000000 à $FFFFFFFF (0 à 4294967295) | .L Long |

Nombres entiers Signés (S) :

On utilise la complémentation à 2.

* Si MSB=1, nombre <0
* Si MSB=0, nombre >=0

|  |  |  |
| --- | --- | --- |
| **Nombres entiers Signés (S) :** | | |
| **bits** |  |  |
| -8 | MSB=0 $00 à $7F (0 à 127)  MSB=1 $80 à $FF (-128 à -1) | .B Byte |
| -16 | MSB=0 $0000 à $7FFF (0 à 32767)  MSB=1 $8000 à $FFFF (-32768 à -1) | .W Word |
| -32 | MSB=0 $00000000 à $7FFFFFFF  MSB=1 $80000000 à $FFFFFFFF | .L Long |

# Les registres du Motorola 68000

Le Motorola 68000 possède 8 registres pour les adresses et 8 registres pour les données, permettant de stocker des nombres jusqu'à 32 bits par 2 sections de mots de 16 bits.

# Les registres des données

8 registres pour les données D0, D1…D7

Une image contenant ligne, capture d’écran, reçu, texte

Le contenu généré par l’IA peut être incorrect.

# Les registres d’adressage

9 registres pour les données A0, A1…A7

Une image contenant ligne, capture d’écran, reçu, texte

Le contenu généré par l’IA peut être incorrect.

# Le registre d’adressage A7

Le registre A7 est un registre d'adresse, et il possède toutes les caractéristiques décrites ci-après. Il peut être utilisé exactement comme les registres A0 à A6. Cependant ce registre présente la particularité d'être directement utilisé par le 68000, comme pointeur de pile. Car le 68000 a besoin de stocker de manière dynamique des informations liées à son fonctionnement, variables locales , adresse de retour de sous-programme, passage de paramètres, interruption…

Une image contenant texte, capture d’écran, ligne, Police

Le contenu généré par l’IA peut être incorrect.

Dans l'architecture du Motorola 68000, deux registres importants sont utilisés pour gérer les modes de fonctionnement du processeur : le User Stack Pointer (USP) et le Supervisor Stack Pointer (SSP).

# Le compteur de programme et le registre d’état

Une image contenant ligne, capture d’écran, reçu, Parallèle

Le contenu généré par l’IA peut être incorrect.

# Principales instructions du Motorola 68000

Le MC68000 dispose d'un jeu d'instructions riche et varié, permettant une grande flexibilité dans la programmation.

Pour la description de ces instructions, Dx et Dy représentent un des huit registres D0 à D7. De même pour Ax et Ay qui représentent les registres d'adresse.

# Instructions arithmétiques et logiques

|  |  |  |  |
| --- | --- | --- | --- |
| **Additions** | * **ABCD** * **ADD** * **ADDA** * **ADDI** * **ADDQ** * **ADDX** * **…** | **Logiques** | * **AND** * **ANDI** * **ANDI to SR** * **ANDI to CCR** |

# Instructions arithmétiques

**ABCD Addition décimale avec le bit d'extension X du CCR**

**Syntaxe:** ABCD Dy, Dx ou ABCD -(Ay),-(Ax)

**Taille opérande:** Byte

Addition BCD entre deux registres de données Dy, Dx ou entre deux cases mémoires adressées en -(Ay),-(Ax).

**Modifications du registre CCR:**

X: Idem à C V: Indéfini

N: Indéfini C: 1 si retenue décimale, sinon 0

Z: 0 si résultat<>0,

sinon inchangé

Exemple assembleur:

**ABCD D2, D1**

**ABCD -(A0), -(A1)**

**ADD Addition binaire**

**Syntaxe:** ADD <AE>, Dn ou ADD Dn, <AE>

**Taille opérande:** Byte, Word, Long word

Addition binaire entre un opérande et le contenu d'un registre de données D0-D7.

**Modifications du registre CCR:**

X: Idem à C V: 1 si débordement

N: 1 si résultat<0 C: 1 si retenue

Z: 1 si résultat=0

Exemple assembleur:

**ADD.W #$1234, D0**

**ADD.L D0, -(A1)**

**ADDA Addition avec registre d'adresse**

**Syntaxe:** ADDA <AE>, An

**Taille opérande:** Word, Long word

Addition binaire entre un opérande et le contenu d'un registre d'adresse.

**Modifications du registre CCR:**

Non affecté

Exemple assembleur:

**ADDA.W #$1234, A0**

**ADDA.L -(A1), A0**

**ADDI Addition Immédiate**

**Syntaxe**: ADDI #<donnée>, <AE>

**Taille opérande:** Byte, Word, Long word

Addition binaire entre la donnée Immédiate et l'opérande de destination.

**Modifications du registre CCR:**

X: Idem à C V: 1 si débordement

N: 1 si résultat<0 C: 1 si retenue

Z: 1 si résultat=0

Exemple assembleur:

**ADDI.W #$1234, D5**

**ADDI.B #$03, (A4)+**

**ADDQ Addition rapide**

**Syntaxe:** ADDQ #<donnée>, <AE>

**Taille opérande:** Byte, Word, Long word

Addition binaire entre la donnée Immédiate et l'opérande de destination. La donnée est comprise entre $01 et $08.

**Modifications du registre CCR:**

X: Idem à C V: 1 si débordement

N: 1 si résultat<0 C: 1 si retenue

Z: 1 si résultat=0

Exemple assembleur:

**ADDQ.B #$01, D0**

**ADDQ.L #$07, $02(A1)**

**ADDX Addition binaire avec le bit d'extension**

**Syntaxe:** ADDX Dy, Dx ou ADDX -(Ay), -(Ax)

**Taille opérande:** Byte, Word, Long word

Addition de la source et de la destination avec le bit d'extension X.

**Modifications du registre CCR:**

X: Idem à C V: 1 si débordement

N: 1 si résultat<0 C: 1 si retenue

Z: 0 si résultat<>0

sinon inchangé

Exemple assembleur:

**ADDX.B D0, D1**

**ADDX.L -(A0), -(A1)**

# Instructions logiques

**AND ET logique**

**Syntaxe:** AND <AE>, Dn ou AND Dn, <AE>

**Taille opérande:** Byte, Word, Long word

ET logique entre source et destination.

**Modifications du registre CCR:**

X: Non affecté V = 0

N: 1 si le MSB du résultat=1 C = 0

Z: 1 si résultat=0

Exemple assembleur:

**AND.W #$1234, D0**

**AND.L -(A2), D3**

**ANDI ET logique Immédiat**

**Syntaxe:** ANDI #<donnée>, <AE>

**Taille opérande:** Byte, Word, Long word

ET logique entre la donnée Immédiate et la destination.

**Modifications du registre CCR:**

X: Non affecté V = 0

N: 1 si le MSB du résultat=1 C = 0

Z: 1 si résultat=0

Exemple assembleur:

**ANDI.W #$1234, D5**

**ANDI.B #$67, -(A2)**

**ANDI to CCR**

**Syntaxe:** ANDI #<donnée 8 bits>, CCR

**Taille opérande:** Byte

ET logique entre une donnée 8 bits et le registre de Codes Conditions CCR

**Modifications du registre CCR:**

X, N, Z, V, C: résultat de l'opération CCR And #<donnée>

Exemple assembleur:

**ANDI.B #$12, CCR**

**EOR OU Exclusif XOR**

**Syntaxe**: EOR Dn, <AE>

**Taille opérande:** Byte, Word, Long word

OU Exclusif XOR entre source et destination, le résultat est mis dans la destination.

**Modifications du registre CCR:**

X: Non affecté V: =0

N: 1 si MSB résultat=1 C: =0

Z: 1 si résultat=0

Exemple assembleur:

**EOR.W D0, D1**

**EOR.B D1, $12345678**

**EOR.L D2, (A1)+**

**EORI OU Exclusif XOR Immédiat**

**Syntaxe**: EORI #<donnée>, <AE>

**Taille opérande:** Byte, Word, Long word

OU Exclusif XOR entre donnée Immédiate et destination, le résultat est mis dans la destination.

**Modifications du registre CCR:**

X: Non affecté V: =0

N: 1 si MSB résultat=1 C: =0

Z: 1 si résultat=0

Exemple assembleur:

**EORI.W #$1234, (A1)+**

**EORI.B #$12, $12345678**

**EORI to CCR**

**Syntaxe:** EORI #<donnée 8 bits>, CCR

**Taille opérande**: Byte

OU Exclusif XOR entre les 8 bits de la donnée Immédiate et le registre de Code Conditions CCR.

**Modifications du registre CCR:**

X, N, Z, V, C: résultat de l'opération CCR XOR #<donnée>

Exemple assembleur:

**EORI.B #$12, CCR**

**EORI to SR**

**Syntaxe:** EORI #<donnée 16 bits>, SR

**Taille opérande:** Word

OU Exclusif XOR entre les 16 bits de la donnée Immédiate et le registre d'état SR si le processeur est en mode Superviseur.

**Modifications du registre CCR:**

X, N, Z, V, C: (Recopie dans CCR du résultat de l'opération SR XOR #<donnée>)

Exemple assembleur:

**EORI.W #$1234, SR**

# Instructions de décalage et de branchement

|  |  |  |  |
| --- | --- | --- | --- |
| **Décalages arithmétiques** | * **ASL (Arithmetic Shift Left)** * **ASR (Arithmetic Shift Right)** | **Branchements (BRA)\*** | * **BCC (Branch if Carry Clear)** * **BCHG (Bit Change)** * **BCLR (Bit Clear)** * **BRA (Branch Always)** * **BSET (Bit Set)** * **BSR (Branch to Subroutine)** |

\*Le Motorola 68000 utilise l'instruction BRA (Branchement inconditionnel) pour gérer le flux d'exécution des programmes.

# Instructions de décalage

**ASL Décalage arithmétique gauche**

**Syntaxe:** ASL Dx, Dy ou ASL #<donnée>, Dy ou ASL <AE>

**Taille opérande:** Byte, Word, Long word

Décalage arithmétique gauche des bits de l'opérande.

Registre: Nombre de Décalages -> donnée Immédiate (0-7) Stocké dans Dx

Case mémoire: Un seul Décalage, exclusivement sur un mot.

**Modifications du registre CCR:**

X: Idem C V: 1 si le MSB a été modifié

N: 1 si le MSB du résultat=1 C: Dernier bit sorti de l'opérande

Z: 1 si résultat=0

Exemple assembleur:

**ASL.B D0, D1**

**ASL.W #$1234, D0**

**ASL (A1)+**

**ASR Décalage arithmétique droit**

**Syntaxe:** ASR Dx, Dy ou ASR #<donnée>, Dy ou ASR <AE>

**Taille opérande:** Byte, Word, Long word

Décalage arithmétique droit des bits de l'opérande.

**Registre:** Nombre de Décalages -> donnée Immédiate (0-7)

Stocké dans Dx

Case mémoire: Un seul Décalage, exclusivement sur un mot.

**Modifications du registre CCR:**

X: Idem C V: 1 si le MSB a été modifié

N: 1 si le MSB du résultat=1 C: Dernier bit sorti de l'opérande

Z: 1 si résultat=0

Exemple assembleur:

**ASR.B D0, D1**

**ASR.W #$1234, D0**

**ASR (A1)+**

# Instructions de branchement

Voici les détails concernant cette instruction BRA :

|  |  |
| --- | --- |
| **Branchement inconditionnel** | **Branchement conditionnel** |
|  |  |

|  |  |
| --- | --- |
| **Branchement inconditionnel** | **Branchement conditionnel** |
| **Instructions :**  **BRA** étiquette Déroutement relatif.  **PC← PC + étiquette**  **↑**  Déplacement : nombre d'instructions entre l'endroit du branchement et l'endroit où le programme doit se brancher.  **JMP** étiquette Déroutement absolu.  **PC← étiquette**  **↑**  **Adresse de branchement** | **Structure :** si <condition> alors <traitement> |

**Syntaxe :** BRA label

**Paramètres :** label =Indique l'étiquette où l'exécution du programme doit se poursuivre.

**Description :** l'instruction BRA permet de modifier le compteur de programme (PC) pour que l'exécution se poursuive à l'emplacement spécifié par le label, en ajoutant un déplacement à la valeur actuelle du PC.

**Remarque :** les appels longs ne sont supportés qu'à partir des microprocesseurs 68020 ou supérieurs.

**Algorithme :** si une condition est vraie, alors le PC est mis à jour avec la nouvelle valeur :

**PC ← PC + d.**

**Impact sur les drapeaux :** cette instruction n'affecte pas les bits de drapeau du registre de codes de condition.

**Cette instruction BRA est essentielle pour le contrôle de l'exécution dans les programmes écrits pour le processeur Motorola 68000.**

**Les instructions de test et de branchement**

Elles permettent d'effectuer un branchement en fonction du registre de conditions en intégrant un comptage.

**Syntaxe :** Bcc étiquette (cc : code condition)

**BCC- BVS Branchements conditionnels**

Les branchements conditionnels, ou sauts, sont effectués en fonction du contenu du Registre CCR.

**Syntaxe:** BEQ <étiquette>

BCS <étiquette>

Si la condition est vraie, l'exécution du programme se poursuit à l'adresse (PC) + Déplacement mentionné par l' étiquette.

Exemple assembleur:

**BEQ $06**

**BNE @LaBas** // Le @ est spécifique à l'assembleur Ed

**Modifications du registre CCR:**

Non affecté

**Remarque :** il s'agit d'un branchement relatif.

# Branchements et conditions CCR

|  |  |
| --- | --- |
| **Description des différents branchements** | **Conditions CCR pour branchement** |
| BCC: Branchement si retenue à zéro | C=0 |
| BCS: Branchement si retenue à un | C=1 |
| BEQ: Branchement si égal | Z=1 |
| BNE: Branchement si différent | Z=0 |
| BGE: Branchement si supérieur ou égal | N xor V = 0 |
| BGT: Branchement si supérieur | Z + [N xor V] = 0 |
| BHI: Branchement si supérieur | C + Z = 0 |
| BLE: Branchement si inférieur ou égal | Z + [N xor V] = 1 |
| BLS: Branchement si inférieur ou égal | C + Z = 1 |
| BLT: Branchement si inférieur | N xor V = 1 |
| BMI: Branchement si négatif | N=1 |
| BPL: Branchement si positif | N=0 |
| BVC: Branchement si pas de dépassement | V=0 |
| BVS: Branchement si dépassement | V=1 |
| BT: Branchement si vrai | tout le temps |
| BF: Branchement si faux | jamais |

**BCHG Test d'un bit et modification**

**Syntaxe:** BCHG Dn, <AE> ou BCHG #<donnée>, <AE>

**Taille opérande:** Byte, Long word

Un bit de la destination est testé pour permettre le positionnement du flag Z, puis ce bit est inversé. Dans le cas d'une opération sur une case mémoire, les bits 0 à 7 peuvent être testés (octet).

**Modifications du registre CCR:**

X: Non affecté V: Non affecté

N: Non affecté C: Non affecté

Z: 1 si résultat=0

Exemple assembleur:

**BCHG.L D0, D1**

**BCHG.B #$02, -(A1)**

**BCLR Test d'un bit et remise à zéro**

**Syntaxe:** BCLR Dn, <AE> ou BCLR #<donnée>, <AE>

**Taille opérande:** Byte, Long word

Un bit de la destination est testé pour permettre le positionnement du flag Z, puis ce bit est mis à zéro.

Dans le cas d'une opération sur une case mémoire, les bits 0 à 7 peuvent être testés (octet).

**Modifications du registre CCR:**

X: Non affecté V: Non affecté

N: Non affecté C: Non affecté

Z: 1 si résultat=0

Exemple assembleur:

**BCLR.L D0, D1**

**BCLR.B #$02, -(A1)**

**BRA Branchement inconditionnel**

**Syntaxe:** BRA <étiquette>

**Taille opérande:** Byte, Word

L'instruction suivante qui sera exécutée est située à l'adresse

(PC) + Etiquette

**Modifications du registre CCR:**

Non affecté

Exemple assembleur:

**BRA $02**

**BRA @LaBas // Le @ est spécifique à Ed**

**BSET Test d'un bit et mise à un**

**Syntaxe:** BSET Dn, <AE> ou BSET #<donnée>, <AE>

Taille opérande: Byte, Long word

Un bit de la destination est testé pour permettre le positionnement du flag Z, puis ce bit est mis à un.

Dans le cas d'une opération sur une case mémoire, les bits 0 à 7 peuvent être testés (octet).

**Modifications du registre CCR:**

X: Non affecté V: Non affecté

N: Non affecté C: Non affecté

Z: 1 si résultat=0

Exemple assembleur:

**BSET.L D0, D1**

**BSET.B #$02, -(A1)**

**BSR Branchement sous-programme**

**Syntaxe:** BSR <étiquette>

**Taille opérande:** Byte, Word

L'adresse 32 bit de l'instruction suivante est sauvegardée dans la pile puis l'exécution du programme se poursuit à l'adresse (PC) + Etiquette

**Modifications du registre CCR:**

Non affecté

**Exemple assembleur:**

**BSR $02**

**BSR @LàBas**

**BTST Test d'un bit**

**Syntaxe:** BTST Dn, <AE> ou BTST #<donnée>, <AE>

**Taille opérande:** Byte, Long word

Un bit de la destination est testé pour permettre le positionnement du flag Z du CCR. Le bit testé n'est pas modifié.

Dans le cas d'une opération sur une case mémoire, les bits 0 à 7 peuvent être testés (octet).

**Modifications du registre CCR:**

X: Non affecté V: Non affecté

N: Non affecté C: Non affecté

Z: 1 si résultat=0

Exemple assembleur:

**BTST.L D0, D1**

**BTST.B #$02, -(A1)**

**CHK Test des limites d'un registre**

Syntaxe: CHK <AE>, Dn

Taille opérande: Word

Les 16 bits de poids faible du registre de données Dn sont comparées ainsi: Si (Dn) < 0 ou (Dn) > (<AE>), le processeur génère l'exception Chk.

**Modifications du registre CCR:**

X: Non affecté V: Indéfini

N: 1 si (Dn)<0, 0 si (Dn)>(<AE>) C: Indéfini

Z: Indéfini

Exemple assembleur:

**CHK.W D0, D1**

**CHK.W -(A1), D2**

**CHK.W #$1234, D2**

**LSL Décalage logique gauche**

**Syntaxe:** LSL Dx, Dy ou LSL #<donnée>, Dy ou LSL <AE>

**Taille opérande:** Byte, Word, Long word

Décalage logique gauche des bits de l'opérande.

Registre: Nombre de Décalages -> donnée Immédiate (0-7)

Stocké dans Dx

Case mémoire: Un seul Décalage, exclusivement sur un mot.

**Modifications du registre CCR:**

X: Idem C V: =0

N: 1 si résultat<0 C: Dernier bit sorti de l'opérande

Z: 1 si résultat=0

Exemple assembleur:

**LSL.B D0, D1**

**LSL.W #$1234, D0**

**LSL (A1)+**

**LSR Décalage logique droit**

**Syntaxe:** LSR Dx, Dy ou LSR #<donnée>, Dy ou LSR <AE>

**Taille opérande:** Byte, Word, Long word

Décalage logique droit des bits de l'opérande.

Registre: Nombre de Décalages -> donnée Immédiate (0-7) - Stocké dans Dx

Case mémoire: Un seul Décalage, exclusivement sur un mot.

**Modifications du registre CCR:**

X: Idem C V: =0

N: 1 si résultat<0 C: Dernier bit sorti de l'opérande

Z: 1 si résultat=0

Exemple assembleur:

**LSR.B D0, D1**

**LSR.W #$1234, D0**

**LSR (A1)+**

**NBCD Complémentation décimale avec le bit d'extension**

**Syntaxe:** NBCD <AE>

**Taille opérande:** Byte

L'opérande et le bit X du CCR sont soustraits à zéro, le résultat est mis dans l'opérande. Si X=0, on a un complément à 10, sinon il s'agit d'une complémentation à 9.

**Modifications du registre CCR:**

X: =1 si retenue décimale V: Indéfini

N: Indéfini C: =1 si retenue décimale

Z: 0 si résultat <> 0

Exemple assembleur:

**NBCD D0**

**NBCD $1234**

**NEG Négation**

**Syntaxe:** NEG <AE>

Taille opérande: Byte, Word, Long word

L'opérande est soustrait de zéro. Le résultat est remis dans l'opérande.

**Modifications du registre CCR:**

X: =1 si retenue décimale V: =1 si débordement

N: =1 si résultat < 0 C: =1 si retenue décimale

Z: =1 si résultat = 0

Exemple assembleur:

**NEG.W D0**

**NEG.L $02(A0)**

**NEGX Négation avec le bit d'extension**

**Syntaxe:** NEGX <AE>

**Taille opérande:** Byte, Word, Long word

L'opérande et le bit X sont soustraits de zéro.

Le résultat est remis dans l'opérande.

**Modifications du registre CCR:**

X: =1 si retenue décimale V: Indéfini

N: Indéfini C: =1 si retenue décimale

Z: =0 si résultat <> 0

Exemple assembleur:

**NEGX.W D0**

**NEGX.L $02(A0)**

**NOP Pas d'opération**

**Syntaxe:** NOP

**Taille opérande**: Aucun

Aucune opération ne se produit, l'état du processeur reste inchangé sauf pour le PC qui pointe sur l'instruction suivante.

**Modifications du registre CCR:**

X: Non affecté V: Non affecté

N: Non affecté C: Non affecté

Z: Non affecté

Exemple assembleur:

**NOP**

**NOT Complément à un**

**Syntaxe:** NOT <AE>

**Taille opérande:** Byte, Word, Long word

Complément à un de l'opérande.

**Modifications du registre CCR:**

X: Non affecté V: =0

N: =1 si résultat < 0 C: =0

Z: =1 si résultat = 0

Exemple assembleur:

**NOT.W D0**

**NOT.L $02(A0)**

**OR OU logique inclusive**

**Syntaxe:** OR <AE>, Dn ou OR Dn, <AE>

**Taille opérande:** Byte, Word, Long word

OU logique entre la source et la destination.

**Modifications du registre CCR:**

X: Non affecté V: =0

N: =1 si MSB résultat=1 C: =0

Z: =1 si résultat = 0

Exemple assembleur:

**OR.W D1, D0**

**OR.L D1, $02(A3)**

**ORI OU logique inclusif Immédiat**

**Syntaxe:** ORI #<donnée>, <AE>

**Taille opérande:** Byte, Word, Long word

OU logique entre la donnée Immédiate et la destination.

**Modifications du registre CCR:**

X: Non affecté V: =0

N: =1 si MSB résultat=1 C: =0

Z: =1 si résultat = 0

Exemple assembleur:

**ORI.B #$03, (A1)+**

**ORI.W #$1234, D1**

**ORI to CCR OU logique inclusif avec le CCR**

**Syntaxe:** ORI #<donnée>, CCR

**Taille opérande:** Byte

OU logique entre la donnée Immédiate 8 bits et le CCR.

**Modifications du registre CCR:**

X: =1 si bit 4 donnée =1 V: =1 si bit 1 donnée =1

N: =1 si bit 3 donnée =1 C: =1 si bit 0 donnée =1

Z: =1 si bit 2 donnée =1

Exemple assembleur:

**ORI #$02, CCR**

**ORI to SR OU logique inclusif avec le SR**

**Syntaxe:** ORI #<donnée 16 bits>, SR

**Taille opérande:** Byte

OU logique entre la donnée Immédiate 16 bits et le SR. Le processeur doit être en mode Superviseur, sinon "violation de privilège".

**Modifications du registre CCR:**

X: =1 si bit 4 donnée =1 V: =1 si bit 1 donnée =1

N: =1 si bit 3 donnée =1 C: =1 si bit 0 donnée =1

Z: =1 si bit 2 donnée =1

Exemple assembleur:

**ORI #$0002, SR**

**PEA Sauvegarde adresse effective**

**Syntaxe:** PEA <AE>

**Taille opérande:** Long word

L'adresse effective est mise sur la pile.

**Modifications du registre CCR:**

X: Non affecté V: Non affecté

N: Non affecté C: Non affecté

Z: Non affecté

Exemple assembleur:

**PEA (A0)**

**PEA $1234**

**RESET Remise à zéro des circuits externes**

**Syntaxe:** RESET

**Taille opérande:** Aucun

La sortie Hardware Reset est validée si on est en mode superviseur.

Sinon, violation de privilège.

**Modifications du registre CCR:**

X: Non affecté V: Non affecté

N: Non affecté C: Non affecté

Z: Non affecté

Exemple assembleur:

**RESET**

**ROL Rotation à gauche**

**Syntaxe:** ROL Dx, Dy ou ROL #<donnée>, Dy ou ROL <AE>

**Taille opérande:** Byte, Word, Long word

Rotation à gauche de la destination. Le nombre de rotations est indiqué par la source.

Si l'opérande est une case mémoire, il n'y a qu'une rotation.

**Modifications du registre CCR:**

X: Non affecté V: =0

N: =1 si MSB résultat =1 C: dernier bit hors de l'opérande

Z: =1 si résultat =0 ou 0 pour une rotation nulle.

Exemple assembleur:

**ROL (A0)+**

**ROL #$01, D0**

**ROR Rotation à droite**

**Syntaxe:** ROR Dx, Dy ou ROR #<donnée>, Dy ou ROR <AE>

**Taille opérande:** Byte, Word, Long word

Rotation à droite de la destination. Le nombre de rotations est indiqué par la source.

Si l'opérande est une case mémoire, il n'y a qu'une rotation.

**Modifications du registre CCR:**

X: Non affecté V: =0

N: =1 si MSB résultat =1 C: dernier bit hors de l'opérande

Z: =1 si résultat =0 ou 0 pour une rotation nulle.

Exemple assembleur:

**ROR (A0)+**

**ROR #$01, D0**

**ROLX Rotation à gauche avec le bit d'extension**

**Syntaxe:** ROLX Dx, Dy ou ROLX #<donnée>, Dy ou ROLX <AE>

**Taille opérande:** Byte, Word, Long word

Rotation à gauche de la destination. Le nombre de rotations est indiqué par la source. La rotation se fait par le bit X du CCR.

Si l'opérande est une case mémoire, il n'y a qu'une rotation.

**Modifications du registre CCR:**

X: Idem C V: =0

N: =1 si MSB résultat =1 C: dernier bit hors de l'opérande

Z: =1 si résultat =0 ou 0 pour une rotation nulle.

Exemple assembleur:

**ROLX (A0)+**

**ROLX #$01, D0**

**RORX Rotation à droite avec le bit d'extension**

Syntaxe: RORX Dx, Dy ou RORX #<donnée>, Dy ou RORX <AE>

**Taille opérande:** Byte, Word, Long word

Rotation à droite de la destination. Le nombre de rotations est indiqué par la source. La rotation se fait par le bit X du CCR.

Si l'opérande est une case mémoire, il n'y a qu'une rotation.

**Modifications du registre CCR:**

X: Idem C V: =0

N: =1 si MSB résultat =1 C: dernier bit hors de l'opérande

Z: =1 si résultat =0 ou 0 pour une rotation nulle.

Exemple assembleur:

**RORX (A0)+**

**RORX #$01, D0**

**RTE Retour d'exception**

**Syntaxe:** RTE

**Taille opérande**: Aucun

Le registre SR et le PC sont restitués par la pile si le processeur est en mode superviseur, sinon il y a "violation de privilège".

**Modifications du registre CCR:**

X: Suivant mot dépilé V: Suivant mot dépilé

N: Suivant mot dépilé C: Suivant mot dépilé

Z: Suivant mot dépilé

Exemple assembleur:

**RTE**

**RTR Retour de sous-programme et restitution des Codes Condition**

**Syntaxe:** RTR

**Taille opérande:** Aucun

Le registre CCR et le PC sont restitués par la pile.

Le registre SR n'est pas affecté.

**Modifications du registre CCR:**

X: Suivant mot dépilé V: Suivant mot dépilé

N: Suivant mot dépilé C: Suivant mot dépilé

Z: Suivant mot dépilé

Exemple assembleur:

**RTR**

**RTS Retour de sous-programme**

**Syntaxe:** RTS

**Taille opérande:** Aucun

Le registre PC est restitué par la pile.

**Modifications du registre CCR:**

X: Non affecté V: Non affecté

N: Non affecté C: Non affecté

Z: Non affecté

Exemple assembleur:

**RTS**

**SBCD Soustraction décimale avec le bit d'extension**

**Syntaxe:** SBCD Dy, Dx ou SBCD -(Ay), -(Ax)

**Taille opérande:** Byte

Soustraction de la Source et du bit X à la Destination. Le résultat en décimal codé binaire est mis dans la Destination.

**Modifications du registre CCR:**

X: =1 si retenue décimale V: Indéfini

N: Indéfini C: =1 si retenue décimale

Z: =0 si résultat <> 0

**Exemple assembleur:**

**SBCD D0, D1**

**SBCD -(A1), -(A0)**

**Scc Positionnement selon condition**

**Syntaxe**: Scc <AE>

**Taille opérande:** Byte

La condition spécifiée est testée. Si la condition est vraie, l'octet adressé par <AE> est mis à $FF. Sinon, il est mis à $00.

**Modifications du registre CCR:**

X: Non affecté V: Non affecté

N: Non affecté C: Non affecté

Z: Non affecté

Exemple assembleur:

**SEQ @Etiquette**

**SNE @LaBas**

**STOP Chargement du SR et arrêt**

**Syntaxe:** STOP #<donnée 16 bits>

**Taille opérande**: Aucun

La donnée Immédiate 16 bits est mise dans le SR.

Le PC est positionné sur la prochaine instruction et le processeur s'arrête. L'exécution reprend après une exception Trace, une interruption ou un Reset.

Le processeur doit être en mode Superviseur.

**Modifications du registre CCR:**

X: Suivant donnée Immédiate V: Suivant donnée Immédiate

N: Suivant donnée Immédiate C: Suivant donnée Immédiate

Z: Suivant donnée Immédiate

Exemple assembleur

**STOP #$1234**

**SUB Soustraction binaire**

**Syntaxe:** SUB <AE>, Dn ou SUB Dn, <AE>

Taille opérande: Byte, Word, Long word

Soustraction de la Source et de la Destination.

Le résultat est mis dans la Destination.

**Modifications du registre CCR:**

X: =1 si retenue V: =1 si débordement

N: =1 si résultat < 0 C: =1 si retenue

Z: =1 si résultat = 0

Exemple assembleur

**SUB.W D0, D1**

**SUB.W $02(A1), D0**

**SUBA Soustraction à un registre d'adresse**

**Syntaxe:** SUBA <AE>, An

**Taille opérande:** Word, Long word

Soustraction de la Source et de la Destination.

Le résultat est mis dans la Destination et l'opérande source est étendu à 32 bits avant l'opération.

Modifications du registre CCR:

X: Non affecté V: Non affecté

N: Non affecté C: Non affecté

Z: Non affecté

Exemple assembleur

**SUBA.W D0, A1**

**SUBA.L $02(A1), A0**

**SUBI Soustraction Immédiate**

**Syntaxe**: SUBI #<donnée>, <AE>

**Taille opérande**: Byte, Word, Long word

Soustraction de la donnée Immédiate et de la Destination. Le résultat est mis dans la Destination.

**Modifications du registre CCR:**

X: =1 si retenue V: =1 si débordement

N: =1 si résultat < 0 C: =1 si retenue

Z: =1 si résultat = 0

Exemple assembleur:

**SUBI.W #$1234, D1**

**SUBI.B #$12, (A1)+**

**SUBQ Soustraction rapide**

**Syntaxe:** SUBQ #<donnée 8 bits>, <AE>

**Taille opérande**: Byte, Word, Long word

Soustraction de la donnée Immédiate et de la Destination.

La donnée peut prendre les valeurs 1 à 8. Le résultat est mis dans la Destination.

Le CCR n'est pas affecté dans le cas de la soustraction à un registre d'adresse.

**Modifications du registre CCR:**

X: =1 si retenue V: =1 si débordement

N: =1 si résultat < 0 C: =1 si retenue

Z: =1 si résultat = 0

Exemple assembleur:

**SUBQ.B #$01, D0**

**SUBQ.L #$01, $02(A1)**

**SUBX Soustraction avec bit d'extension**

**Syntaxe:** SUBX Dy, Dx ou SUBX -(Ay), -(Ax)

**Taille opérande:** Byte, Word, Long word

Soustraction de la Source et du bit X à la Destination. Le résultat est mis dans la Destination.

**Modifications du registre CCR:**

X: =1 si retenue V: =1 si débordement

N: =1 si résultat < 0 C: =1 si retenue

Z: =0 si résultat <> 0

Exemple assembleur:

**SUBX.B D0, D1**

**SUBX.L -(A1), -(A0)**

**SWAP Echange interne**

**Syntaxe:** SWAP Dn

**Taille opérande:** Word

Echange entre les 16 bits de poids faible et les 16 bits de poids fort d'un registre de données.

**Modifications du registre CCR:**

X: Non affecté V: =0

N: =1 MSB résultat = 1 C: =0

Z: =1 si résultat = 0

Exemple assembleur:

**SWAP D3**

**TAS Test d'un octet et mise à 1 du bit 7**

**Syntaxe:** TAS <AE>

**Taille opérande:** Byte

Test d'un octet par l'adresse effective dont le bit 7 est mis à 1.

**Modifications du registre CCR:**

X: Non affecté V: =0

N: =1 MSB résultat = 1 C: =0

Z: =1 si résultat = 0

Exemple assembleur:

**TAS D0**

**TAS $02(A1)**

**UNLK Déconnexion (UnLink)**

**Syntaxe:** UNLK An

**Taille opérande:** Aucun

Le registre d'adresse An est transféré dans le pointeur de pile. Le contenu de la pile est ensuite transféré dans An. Il s'agit du processus complémentaire de LINK.

**Modifications du registre CCR:**

X: Non affecté V: Non affecté

N: Non affecté C: Non affecté

Z: Non affecté

Exemple assembleur:

**UNLK A2**

**Bibliographie:**

MC68000 databook Motorola

# Les différentes conditions des branchements

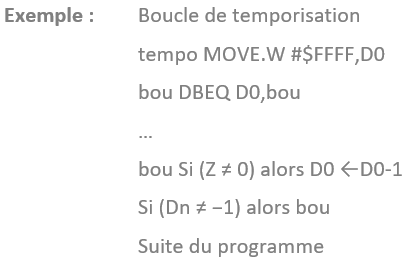
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Conditions** | **Signification** | **Définition** | **Détail** | **Remarques** |
| **CC** | Carry Clear VRAI SI C=0 | Retenue à 0 |  |  |
| **CS** | Carry Set VRAI SI C=1 | Retenue à 1 |  |  |
| **EQ** | Equal to VRAI SI Z=1 | Egal |  |  |
| **NE** | Not Equal VRAI SI Z=0 | Non égal |  |  |
| **GE** | Great than or Equal | Supérieur ou égal |  | Arithmétique signée |
| **GT** | Great Than | Supérieur |  |
| **HI** | Higher than | Plus grand |  | Arithmétique non signée |
| **LE** | Less than or Equal | Inférieur ou égal |  | Arithmétique signée |
| **LT** | Less Than | Inférieur |  |
| **LS** | Low than or Same as | Plus petit ou égal |  | Arithmétique non signée |
| **MI** | Minus | Négatif |  |  |
| **PL** | Plus | Positif |  |  |
| **VC** | oVerflow Clear VRAI SI V=0 | Pas de dépassement |  |  |
| **VS** | oVerflow Set VRAI SI V=1 | Dépassement |  |  |
| **T** | Always True VRAI SI - (= BRA) |  |  |  |
| **F** | Always False |  |  |  |

**Rappels**: deux manières d’affecter un branchement

* **Inconditionnel JMP, BRA**
* **Conditionnel Bcc, (DBcc)**

**Une image contenant texte, reçu, Police, blanc

Le contenu généré par l’IA peut être incorrect.**



DBcc et Scc utilisent les 16 conditions

Bcc n’utilise pas T et F (14 conditions).

On distingue le test :

* de conditions simples : un seul flag est testé.
* de conditions multiples : plusieurs flags sont testés.

|  |  |  |
| --- | --- | --- |
| **Symbole** | **Nombre Non Signés** | **Nombre Signés** |
| **>** | HI | GT |
| **>=** | CC | GE |
| **<** | CS | LT |
| **<=** | LS | LE |
| **Status** | **C,Z** | **V,Z,N** |

**Remarques :**

Le flag N intervient toujours dans les équations logiques de ces instructions de branchement lorsque l ’on manipule des nombres Signés.

C’est à nous de savoir le type des entiers manipulés et donc utiliser la bonne instruction de branchement conditionnel en conséquence (ET NON LE PROCESSEUR !).

**Points clés**

Dans l'architecture du Motorola 68000, **le Compteur de Programme (PC) joue un rôle crucial dans l'exécution des instructions.**

Voici les principales caractéristiques et fonctions du PC :

1. **Fonction principale** : Le PC contient l'adresse de la prochaine instruction à exécuter. À chaque cycle d'horloge, le PC est automatiquement incrémenté pour pointer vers l'instruction suivante dans la mémoire.
2. **Adresses de 32 bits** : Le PC utilise une largeur d'adresse de 32 bits, ce qui permet d'adresser jusqu'à 4 Go de mémoire, bien que le 68000 ait un espace d'adressage effectif limité à 16 Mo en raison de la largeur de son bus d'adresses de 24 bits.
3. **Mode d'adressage** : Le PC peut être utilisé dans des instructions d'adressage relatif. Par exemple, les instructions de branchement conditionnel utilisent le PC pour déterminer la nouvelle adresse de saut en ajoutant un déplacement à la valeur actuelle du PC.
4. **Interaction avec le registre d'état (SR)** : Lorsqu'une interruption ou une exception se produit, l'adresse du PC est sauvegardée sur la pile, permettant au processeur de revenir à l'état précédent après le traitement de l'interruption.

**En résumé, le PC dans le Motorola 68000 est essentiel pour le contrôle du flux d'exécution des programmes, permettant à la fois l'incrémentation automatique et la gestion des sauts conditionnels.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Instructions de test et de comparaison** | | | |
| **Tests de bit** | * **BTST (Bit Test)** * **TAS (Test And Set)** | **Comparaisons** | * **CMP (Compare)** * **CMPA (Compare Address)** * **CMPI (Compare Immediate)** * **CMPM (Compare Memory)** |

**Registre CCR (Condition Code Register)**

**Le CCR contient des bits d'état qui reflètent les résultats des opérations effectuées par le processeur.** Il est utilisé pour déterminer les conditions dans lesquelles certaines instructions peuvent être exécutées, comme les branchements conditionnels.

**Modes d'accès :**

**En mode utilisateur, il est possible d'écrire uniquement dans l'octet utilisateur du CCR.** En mode superviseur, l'accès est élargi et il est possible d'écrire dans le mot entier du CCR.

**Registre SR (Status Register)**

**Le SR est un registre qui contient des informations sur l'état du processeur, y compris les bits de contrôle et les bits d'exception.** Il permet de gérer les interruptions et les exceptions.

**Modes d'accès :**

En mode utilisateur, l'instruction MOVE from SR permet de lire la partie superviseur du registre SR, ce qui pose des problèmes de sécurité et de virtualisation.

En mode superviseur, le SR est accessible dans son intégralité, permettant une gestion complète des exceptions et des interruptions.

**Points clés**

Le registre A7 est utilisé comme pointeur de pile pour le mode superviseur et utilisateur, ce qui est essentiel pour la gestion des sous-programmes.

**La distinction entre les modes utilisateur et superviseur est cruciale pour la sécurité et la gestion des ressources dans les systèmes d'exploitation modernes.**

**Ces registres jouent un rôle fondamental dans le fonctionnement du M68000, en permettant une gestion efficace des états et des exceptions.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Instructions de contrôle de flux** | | | |
| **Sauts** | * **JMP (Jump)** * **JSR (Jump to Subroutine)** | **Trappes** | * **TRAP (Trap)** * **TRAPV (Trap on Overflow)** |

**JMP Saut de type Jump**

**Syntaxe:** JMP <AE>

**Taille opérande:** Aucun

L'exécution se poursuit à l'adresse spécifiée dans l'instruction, qui peut être une Etiquette

**Modifications du registre CCR:**

Non affecté

Exemple assembleur:

**JMP.L (A1)**

**JMP @LaBas**

**JSR Saut vers un sous-programme**

**Syntaxe:** JSR <AE>

**Taille opérande:** Aucun

L'exécution se poursuit à l'adresse spécifiée dans l'instruction, qui peut être une Etiquette.

L'adresse de l'instruction suivante est d'abord mise sur la pile.

**Modifications du registre CCR:**

Non affecté

Exemple assembleur:

**JSR.L (A1)**

**JSR @LaBasAussi**

**TRAP Demande d'exception Trap**

**Syntaxe:** TRAP #<numéro>

**Taille opérande:** Aucun

Le processeur entame une procédure d'exception. Le numéro permet de disposer de 16 vecteurs propres à l'instruction TRAP.

**Modifications du registre CCR:**

X: Non affecté V: Non affecté

N: Non affecté C: Non affecté

Z: Non affecté

Exemple assembleur:

**TRAP #1**

**TRAPV Exception si débordement**

**Syntaxe: TRAPV**

**Taille opérande:** Aucun

Si le bit V du CCR est positionné, il y a génération de l'exception de vecteur nø 7. Sinon, l'exécution du programme se poursuit normalement.

**Modifications du registre CCR:**

X: Non affecté V: Non affecté

N: Non affecté C: Non affecté

Z: Non affecté

Exemple assembleur:

**TRAPV**

**TST Test d'un opérande**

**Syntaxe:** TST <AE>

**Taille opérande:** Byte, Word, Long word

L'opérande est comparé à zéro. Les codes condition sont alors positionnés.

**Modifications du registre CCR:**

X: Non affecté V: =0

N: =1 si résultat < 0 C: =0

Z: =1 si résultat = 0

Exemple assembleur:

**TST.W D0**

**TST.L $02(A1)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Instructions diverses** | | | |
| **Multiplications** | * **MULS (Multiplication Signed)** * **MULU (Multiplication Unsigned)** | **Division** | * **DIVS (Division Signed)** * **DIVU (Division Unsigned)** |

**MULS Multiplication signée**

**Syntaxe:** MULS <AE>, Dn

**Taille opérande:** Word

Multiplication 2 x 16 bits, résultat signé sur 32 bits dans Dn

**Modifications du registre CCR:**

X: Non affecté V: =0

N: 1 si résultat < 0 C: =0

Z: 1 si résultat = 0

Exemple assembleur:

**MULS D0, D1**

**MULS #$1234, D2**

**MULS $1234, D3**

**MULU Multiplication non signée**

**Syntaxe:** MULU <AE>, Dn

**Taille opérande:** Word

Multiplication 2 x 16 bits, résultat non signé sur 32 bits dans Dn

**Modifications du registre CCR:**

X: Non affecté V: =0

N: 1 si MSB résultat=1 C: =0

Z: 1 si résultat = 0

Exemple assembleur:

**MULU D0, D1**

**MULU #$1234, D2**

**MULU $1234, D3**

**DIVS Division signée**

**Syntaxe**: DIVS <AE>, Dn

**Taille opérande:** Word

Division de l'opérande destination 32 bits par l'opérande source 16 bits.

**Le résultat 32 bits signé est organisé comme suit:**

Quotient : Mot de poids faible

Reste : Mot de poids fort.

Le signe du reste est le même que celui du dividende, le débordement peut être positionné.

**Modifications du registre CCR:**

X: Non affecté V: 1 si débordement

N: 1 si quotient<0, Indéfini si V=1 C: =0

Z: 1 si quotient=0, Indéfini si V=1

Exemple assembleur:

**DIVS.W #$1234, D0**

**DIVS.W -(A1), D2**

**DIVU Division non signée**

**Syntaxe:** DIVU <AE>, Dn

**Taille opérande:** Word

Division de l'opérande destination 32 bits par l'opérande source 16 bits.

Le résultat 32 bits non signé est organisé comme suit:

Quotient : Mot de poids faible

Reste : Mot de poids fort.

**Modifications du registre CCR:**

X: Non affecté V: 1 si débordement

N: 1 si MSB quotient=1, Indéfini si V=1 C: =0

Z: 1 si quotient=0, Indéfini si V=1

Exemple assembleur:

**DIVU.W #$1234, D0**

**DIVU.W -(A1), D2**

# Mode d’adressage

**Le Motorola 68000 dispose de 14 modes d'adressage distincts**, permettant une grande flexibilité dans l'accès aux données.

Instruction MOVE

* 75 % d’un programme
* MOVE .B/.W/.L source, destination
* Déplace une donnée 8/16/32 bits d’un registre ou d’une position mémoire vers un autre registre ou une autre position mémoire.

|  |  |  |  |
| --- | --- | --- | --- |
| **Transferts** | * **MOVE (Transfert de données)** * **MOVEA (Transfert d'adresse)** * **MOVE to CCR (Transfert vers le registre CCR\*)** * **MOVE from SR (Transfert depuis le registre SR\*)** * **MOVEM (Transfert multiple)** * **MOVEQ (Transfert immédiat)** | **Manipulations** | * **CLR (Clear)** * **NEG (Négatif)** * **NOT (Non logique)** * **SWAP (Permutation)** |

\*Le MC68000 dispose de deux registres importants liés au contrôle des opérations : le CCR (Condition Code Register) et le SR (Status Register).

# Instructions de transfert

Ces modes peuvent être regroupés en cinq catégories principales :

1. **Adressage absolu**

L'adresse du mot à manipuler est indiquée directement dans l'instruction.

Exemple : **MOVE.W $1000, D0 copie la valeur à l'adresse mémoire $1000 dans D0.**

1. **Adressage direct de registres**

Les données à manipuler sont stockées dans un registre (adresse ou données).

Exemple : **MOVE.W D1, D0 copie le contenu de D1 dans D0.**

1. **Adressage immédiat**

Utilisé pour assigner une valeur constante dans un programme, précédée du symbole dièse #.

Exemple : **MOVE.W #$1234, D0 copie la valeur $1234 directement dans D0.**

1. **Adressage indirect de registres**

Les données sont accessibles via un registre d'adresse, permettant de pointer vers des emplacements mémoire.

Exemple : **MOVE.W (A0), D1 copie la valeur pointée par A0 dans D1.**

1. **Adressage relatif au compteur programme (PC)**

Permet d'accéder à des données en fonction de la position actuelle du compteur de programme (PC).

Exemple :

**MOVE.B $10(PC), D0**

**l'opérande est récupéré à l'adresse effective qui est calculée comme suit :**

* **Adresse effective (EA) = PC**
* **déplacement (10 dans ce cas).**

# Opérations sur Opérandes, Comparaisons, Sauvegardes

**CLR Remise à zéro d'un opérande**

**Syntaxe:** CLR <AE>

**Taille opérande:** Byte, Word, Long word

Tous les bits de la destination sont mis à zéro.

**Modifications du registre CCR:**

X: Non affecté V: =0

N: =0 C: =0

Z: =1

Exemple assembleur:

**CLR.L D1**

**CLR.B -(A1)**

**CMP Comparaison Registre Dn**

**Syntaxe:** CMP <AE>, Dn

**Taille opérande:** Byte, Word, Long word

L'opérande source est soustrait de l'opérande destination afin de positionner les flags Registre CCR:

**Modifications du registre CCR:**

X: Non affecté V: 1 si débordement

N: 1 si résultat<0 C: 1 si retenue

Z: 1 si résultat=0

Exemple assembleur:

**CMP.L D0, D1**

**CMP.W #$1234, D2**

**CMP.L -(A1), D0**

**CMPA Comparaison Adresse An**

**Syntaxe:** CMPA <AE>, An

**Taille opérande:** Word, Long word

L'opérande source est soustrait de l'opérande destination afin de positionner les flags Registre CCR.

**Modifications du registre CCR:**

X: Non affecté V: 1 si débordement

N: 1 si résultat<0 C: 1 si retenue

Z: 1 si résultat=0

Exemple assembleur:

**CMP.L D0, A1**

**CMP.W #$1234, A5**

**CMP.L -(A1), A4**

**CMPI Comparaison Immédiate**

**Syntaxe:** CMPI #<donnée>, <AE>

**Taille opérande:** Byte, Word, Long word

La donnée Immédiate est soustraite de l'opérande destination afin de positionner les flags Registre CCR.

**Modifications du registre CCR:**

X: Non affecté V: 1 si débordement

N: 1 si résultat<0 C: 1 si retenue

Z: 1 si résultat=0

Exemple assembleur:

**CMPI.B #$02, D1**

**CMPI.W #$1234, (A1)+**

**CMPI.L #$12345678, -(A3)**

**CMPM Comparaison mémoire**

**Syntaxe:** CMPM (Ay)+, (Ax)+

**Taille opérande:** Byte, Word, Long word

L'opérande source est soustrait de l'opérande destination afin de positionner les flags Registre CCR:

**Modifications du registre CCR:**

X: Non affecté V: 1 si débordement

N: 1 si résultat<0 C: 1 si retenue

Z: 1 si résultat=0

Exemple assembleur:

**CMPA.L (A1)+, (A0)+**

**CMPA.B (A2)+, (A1)+**

**DBCC . DBVS**

Test condition, Décrémentation et branchement.

Syntaxe: DBcc Dn, <étiquette>

**Taille opérande:** Word

**Modifications du registre CCR:**

Non affecté

**Si la condition est vérifiée (Dn=-1):**

Pas d'opération.

**Si la condition n'est pas vérifiée (Dn<>-1):**

Dn=Dn-1

Branchement à l'adresse (PC) + Déplacement mentionné par l' Etiquette.

Les conditions sont identiques à celles des instructions BCC . BVS.

Exemple assembleur:

**DBNE D0, $02**

**DBEQ D1, @LaBas**

**EXG Echange de registres**

**Syntaxe:** EXG Rx, Ry

**Taille opérande:** Long word

Echange entre deux registres 32 bits.

Ces échanges peuvent être:

Dn, An, Dn-An, An-Dn.

**Modifications du registre CCR:**

Non affecté

Exemple assembleur:

**EXG D1, A2**

**EXT Extension du signe d'un registre de données**

**Syntaxe:** EXT Dn

**Taille opérande:** Word, Long word

Sur un Mot: Recopie du bit 7 sur les bits 8 à 15. Sur un Mot long: Recopie du bit 15 sur les bits 16 à 31.

**Modifications du registre CCR:**

X: Non affecté V: =0

N: 1 si résultat<0 C: =0

Z: 1 si résultat=0

Exemple assembleur:

**EXT.W D1**

**ILLEGAL**

Cette instruction codée $4AFC génère l'exception "Instruction Illégale" de vecteur nø4.

**Modifications du registre CCR:**

Non affecté

**LEA Chargement adresse effective**

**Syntaxe:** LEA <AE>, An

**Taille opérande:** Long word

Les 32 bits de l'adresse effective sont chargés dans le registre d'adresse An.

**Modifications du registre CCR:**

Non affecté

Exemple assembleur:

**LEA.L $12345678, A1**

**LEA.L $04(A0), A2**

**LINK Réalisation de liens Link**

**Syntaxe:** LINK An, #<Déplacement 16 bits>

**Taille opérande:** Aucun

Le contenu de An est mis sur la pile, puis An est chargé avec la valeur du SP. Enfin, le Déplacement Immédiat est ajouté au SP avec extension sur 32 bits.

**Modifications du registre CCR:**

Non affecté

Exemple assembleur:

**LINK A1, #$1234**

# Le logiciel EASY68K

Le logiciel EASy68K est un environnement de développement intégré gratuit pour programmer et simuler le microprocesseur Motorola 68000, offrant des outils d'édition, d'assemblage et de débogage, compatible avec Windows XP à 11.

Le but de cette partie est d'introduire à l’édition, la compilation et la simulation avec EASy68K.

# L’environnement Easy68K

EASy68K est un environnement de développement intégré (IDE) conçu spécifiquement pour la programmation et la simulation du microprocesseur Motorola 68000.

**Ce simulateur est gratuit, open source et sous Licence (General Public Use Licence).**

Une image contenant texte, logiciel, nombre, Logiciel multimédia

Description générée automatiquement

Pour pouvoir l’utiliser sous Windows, il suffit de télécharger puis lancer l’installateur SetupEASy68K.exe :

**Télécharger SetupEASy68K.exe depuis** [**http://www.easy68k.com/**](http://www.easy68k.com/)

[**https://fr.freedownloadmanager.org/Windows-PC/EASy68K-GRATUIT.html**](https://fr.freedownloadmanager.org/Windows-PC/EASy68K-GRATUIT.html)

**Ou encore :**

[**https://easy68k.software.informer.com/download/#downloading**](https://easy68k.software.informer.com/download/#downloading)

Sachant que les utilisateurs sous Linux ou Mac peuvent l’installer sous l’émulateur Windows, en tant que super utilisateur, avec la commande :

**wine ./SetupEASy68K.exe**

Une fois cet environnement est installé, vous disposeriez d’un menu similaire à celui-ci :

Une image contenant texte, logiciel, ordinateur, Icône d’ordinateur

Description générée automatiquement

L’environnement EASy68K comporte un éditeur EASy68K, un Help, un simulateur Sim68K et un éditeur binaire EASyBIN qui permet de manipuler des zones mémoires ou des fichiers S-Record.

Ces fichiers peuvent être manipulés en hexadécimal ou en mode texte.

# Les fichiers manipulés

Afin de tester un programme assembleur, vous devez suivre plusieurs étapes. Le diagramme de la figure suivante illustre les étapes par lesquelles passe un programme écrit en assembleur 68000.

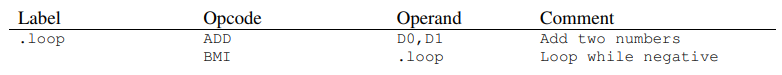
Une image contenant diagramme, Police, texte, ligne

Description générée automatiquement

1. La première étape est d’éditer un fichier (EASy68K) contenant le code du programme assembleur dont l’extension doit être ‘.X68’
2. Une fois le fichier ‘.X68’ est enregistré, vous pouvez le compiler. Cette étape génère deux fichiers : un pour la simulation (exécutable) d’extension ‘.S68’ et un autre pour debugger d’extension ‘.L68’
3. La simulation de ce programme est assurée par le programme Sim68K.

# Description d’une ligne de code

Un programme en assembleur 68000 est constitué par des lignes de la forme :

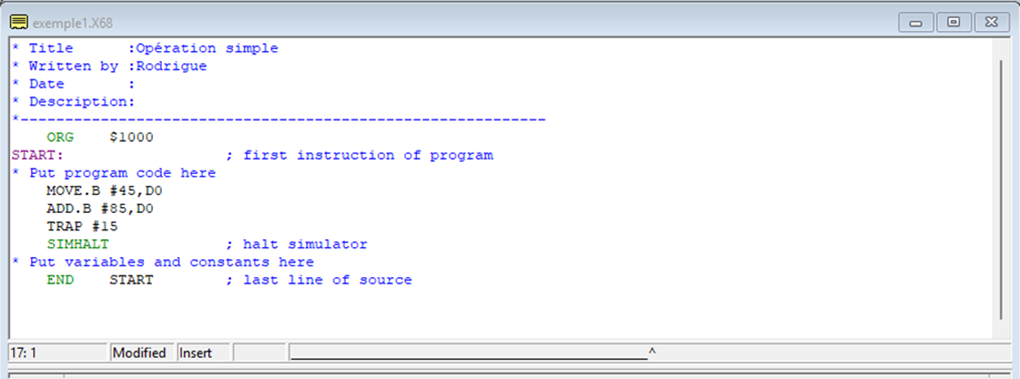


* **Label :** une étiquette définie par le programme pour marquer un endroit du programme
* **Opcode :** une instruction que le processeur peut exécuter
* **Operand :** les données nécessaires à l’exécution des instructions
* **Comment :** des explications pour documenter le programme.

# Edition, compilation et simulation

1. Création et édition d’un programme

Lancer l’éditeur EASy68K et créer un nouveau fichier et saisir le code suivant :



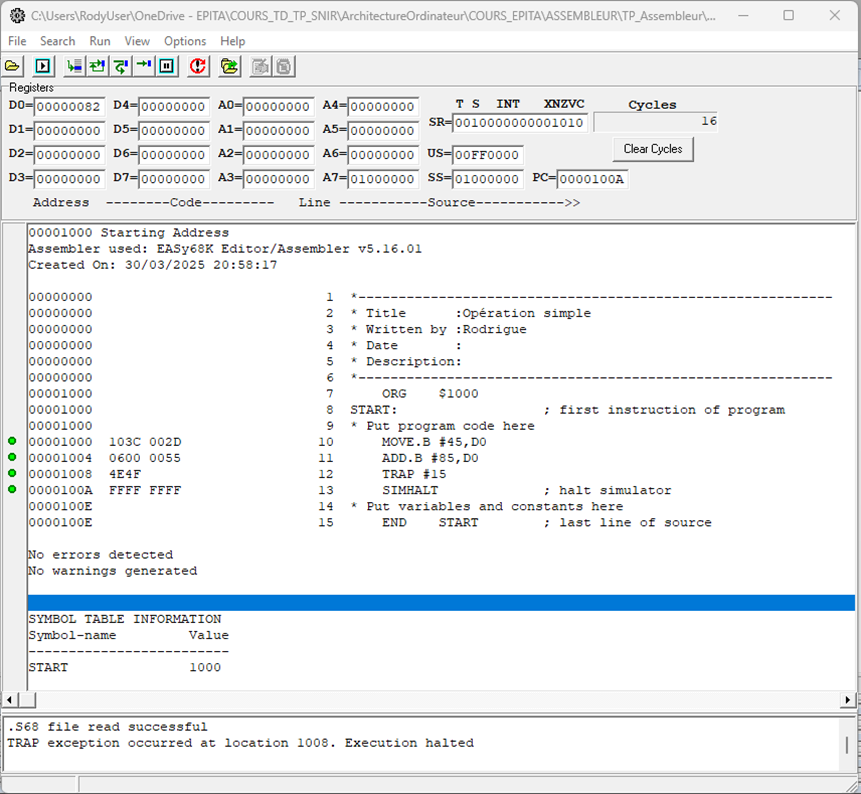
1. Après compilation, exécute :

Une image contenant texte, logiciel, ordinateur, Icône d’ordinateur

Le contenu généré par l’IA peut être incorrect.

# Simulation du programme

La figure suivante affiche la fenêtre principale du simulateur permettant de suivre l’évolution de l’exécution du programme.



Cette fenêtre permet de suivre l’évolution des registres **D0…D7, A0…A7, SS, US, SR et PC** au cours de la simulation d’un programme.

Elle offre aussi divers outils pour la manipulation et le suivi du contenu de la mémoire, de la pile, ainsi qu’une maquette de simulation ‘hardware’.

Cette fenêtre affiche aussi le fichier ‘.L68’ au cours de la simulation. Ce fichier comporte plusieurs colonnes qui simplifient la compréhension et la suivie de l’exécution du programme :

* **Address : l’adresse à partir de laquelle l’instruction commence ;**
* **Code : le code de l’opération en hexadécimal ;**
* **Line : Le numéro de la ligne correspondante ;**
* **Source : le contenu du fichier source du programme ;**

La barre d’outils offre aussi plusieurs boutons permettant d’interagir avec le simulateur, en sélectionnant le mode de simulation ainsi que l’opération à réaliser.

# Travail demandé

Exécuter le programme en mode ligne par ligne et suivez l’évolution des registres pour répondre aux questions suivantes :

1. Ecrire un code pour afficher “ **Hello votre\_prénom !** “ dans le terminal Sim68K
2. Expliquez pourquoi la commande suivante ne peut pas fonctionner :

**ADDI.B #12345, D5**

**Comme on cherche en forme de BYTE soit une valeur de 255 maximum, donc l’instruction ne fonctionne pas avec le ADDI.B**

1. Faites une exécution en mode ligne par ligne et dresser un tableau contenant les registres qui changent à chaque étape.

Par exemple : **MOVE.B #45,D0**

**ADD.B #85,D0**

**TRAP #15**

|  |  |
| --- | --- |
| **Ligne de commande** | **Registre D0** |
| MOVE.B #45,D0 | 2D |
| ADD.B #85,D0 | 82 |
| TRAP #15 | 82 |

Afficher le contenu de la mémoire ?

Une image contenant texte, Police, capture d’écran, conception

Le contenu généré par l’IA peut être incorrect.

Donner les adresses de début et de fin de la partie code et de la partie données de ce programme ?

Addresse de début partie code: 1000, adresse de fin partie code : 1008, adresse de début de la partie données : 100A

Comment peut-ton définir la tâche que doit effectuer l’instruction **TRAP #15** ? Et comment elle récupère les paramètres nécessaires ?

L'instruction TRAP #15 appelle une routine système du simulateur défini par un code et utilise les registres (comme A1 ou D0) pour transmettre ou recevoir les paramètres nécessaires.

1. On souhaite calculer la somme de 345 et 1885. Utiliser le simulateur 68000 pour calculer la somme. Notez le code qui fonctionne !



Capture des registres et expliquer

Une image contenant texte, capture d’écran, Police, ligne

Le contenu généré par l’IA peut être incorrect.

La valeur 8B6, en D0 en base 16 (hexadécimal) vaut 2230 en base 10 (décimal), ce qui est le résultat exact de la somme entre 345 et 1885.

1. Vérifier le contenu de registre D0 pour le calcul utilisant la commande SUB, calculer 45-12.



Capture des registres et expliquer

**Une image contenant texte, capture d’écran, Police, ligne

Le contenu généré par l’IA peut être incorrect.**

La valeur 21, en D0 en base 16 (hexadécimal) vaut 33 en base 10 (décimal), ce qui est le résultat exact de la différence entre 45 et 12.

1. En utilisant la commande MULS, calculer 45\*12.



Capture des registres et expliquer

**Une image contenant texte, capture d’écran, Police, ligne

Le contenu généré par l’IA peut être incorrect.**

La valeur 21C, en D0 en base 16 (hexadécimal) vaut 540 en base 10 (décimal), ce qui est le résultat exact de la multiplication entre 45 et 12.

1. En utilisant la commande MULU, calculer 45\*12.



Capture des registres et expliquer

**Une image contenant texte, capture d’écran, Police, ligne

Le contenu généré par l’IA peut être incorrect.**

La valeur 21C, en D0 en base 16 (hexadécimal) vaut 540 en base 10 (décimal), ce qui est le résultat exact de la multiplication entre 45 et 12.

1. Exercice : Utilisant la commande MULS, calculer -45\*12.

…………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………….………………………………..…………………………………………………………………………………………………………………………………………………….…………………………………………………………………………………………………………………………………………………………………….…

Capture des registres et expliquer

Une image contenant texte, Police, ligne, nombre

Le contenu généré par l’IA peut être incorrect.

La valeur FFFFFDE4, en D0 en base 16 (hexadécimal) vaut (-540) en base 10 signé (décimal), ce qui est le résultat exact de la multiplication entre (-45) et 12.

On peut aussi noter que le CRR pour les nombres signés est à 1, signifiant que le nombre est négatif.

1. Calculer la somme de 45+12-48+18. Notez le code qui fonctionne !

Une image contenant texte, Police, conception

Le contenu généré par l’IA peut être incorrect.

Capture des registres et expliquer

Une image contenant texte, capture d’écran, Police, ligne

Le contenu généré par l’IA peut être incorrect.

La valeur 1B, en D0 en base 16 (hexadécimal) vaut 27 en base 10 (décimal), ce qui est le résultat exact de la somme entre 45, 12, (-48), et 18.

1. La même question que précédent mais il y a plus de données : 45 + 12 - 48 + 18 - 5 + 81 + 12 + 35 - 18 + 89. Notez la commande de qui fonctionne !

Une image contenant texte, Police, capture d’écran

Le contenu généré par l’IA peut être incorrect.

Ici capture d’écran

Une image contenant texte, Police, ligne, nombre

Le contenu généré par l’IA peut être incorrect.

1. Calculer la somme 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 + 11 + 12 + 13. Notez le code qui fonctionne !

Une image contenant texte, Police, capture d’écran

Le contenu généré par l’IA peut être incorrect.

Capture des registres et expliquer

Une image contenant texte, Police, ligne, Parallèle

Le contenu généré par l’IA peut être incorrect.

En effet, la somme de 3 à 13 donne 88 en base 10 (décimal) et donc 58 en base 16 (hexadécimal). Ce que nous retrouvons dans le registre D0.

1. Ecrire un sous-programme qui reçoit deux valeurs signées (mots) dans D1 et D2 et qui retourne le max des deux dans D3. Tester le programme pour différentes valeurs de D1 et D2.

Une image contenant texte, Police, capture d’écran

Le contenu généré par l’IA peut être incorrect.

1. Ecrire un sous-programme qui reçoit une valeur dans D0 et qui retourne dans D1, 1 si D0 est impaire et 0 si D0 est paire.



Capture des registres et expliquer

**Une image contenant texte, capture d’écran, Police, conception

Le contenu généré par l’IA peut être incorrect.**

**Une image contenant texte, capture d’écran, Police, conception

Le contenu généré par l’IA peut être incorrect.**

**Dans les cas de 15 (ici ‘F’) et 16 (ici ‘10’), on voit bien que 15 est impaire, D1 = 1. Et on voit que 16 est pair, D1 = 0.**

1. On souhaite calculer 5\*10 sans utilisation de la commande MUL. C’est-à-dire que l’on calcule 10+10+10+10+10. Ecrire le programme en utilisant l’instruction DBRA. Notez le code qui fonctionne !

Capture des registres et expliquer

D0 = résultat

D1 = opérande 1

D2 = opérande 2

D3 = Nombre d’iter

Une image contenant texte, Police, capture d’écran, conception

Le contenu généré par l’IA peut être incorrect. Une image contenant texte, capture d’écran, Police, conception

Le contenu généré par l’IA peut être incorrect.

Ça fait une boucle pour faire une suite d’addition.

1. Ecrire un sous-programme qui reçoit deux valeurs signées (mots) dans D0 et D1 et qui retourne le max des deux dans D2. Notez le code qui fonctionne !

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

1. Ecrire un sous-programme ABS qui reçoit une valeur signée dans D0 et qui retourne sa valeur absolue dans D1. Testez votre programme pour des valeurs positive et négative de D0.

………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………….

1. Ecrire un sous-programme MIN qui reçoit trois valeurs signées (mots) dans D0, D1 et D2 et qui retourne la main des trois dans D3.

…………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………………

1. Ecrire un sous-programme FONC qui reçoit une variable x dans D0.W et qui retourne dans D1.L la valeur : x² + x + 5

Une image contenant texte, Police, capture d’écran, conception

Le contenu généré par l’IA peut être incorrect.

Soit le code suivant :

**MOVE.B #5,D0**

**BCL:**

**DBF D0,BCL**

Décrire le programme et applique le fonctionnement en mode pas à pas.

Il fait d’abords, la nombre au carré, puis ajoute le nombre D0, puis ajoute 5.