

# Contrôle S4

## Architecture des ordinateurs

Durée : 1 h 30

Répondre exclusivement sur le document réponse.

**Exercice 1 (4 points)**

Remplir le tableau présent sur le [document réponse](#). Donnez le nouveau contenu des registres (sauf le PC) et/ou de la mémoire modifiés par les instructions. **Vous utiliserez la représentation hexadécimale. La mémoire et les registres sont réinitialisés à chaque nouvelle instruction.**

Valeurs initiales :    D0 = \$1234FFF1    A0 = \$00005000    PC = \$00006000  
                           D1 = \$0000007F    A1 = \$00005008  
                           D2 = \$0000FFFD    A2 = \$00005010

                          \$005000    54 AF 18 B9 E7 21 48 C0  
                           \$005008    C9 10 11 C8 D4 36 1F 88  
                           \$005010    13 79 01 80 42 1A 2D 49

**Exercice 2 (3 points)**

Remplir le tableau présent sur le [document réponse](#). Vous devez trouver le nombre manquant (sous sa forme hexadécimale) en fonction de la taille de l'opération et de la valeur des *flags* après l'opération. **Si plusieurs solutions sont possibles, vous retiendrez uniquement la plus petite.**

**Exercice 3 (4 points)**

Soit le programme ci-dessous. Complétez le tableau présent sur le [document réponse](#).

```

Main      move.l  #$80,d7

next1     moveq.l #1,d1
          cmpi.w  #$8000,d7
          bgt   next2
          moveq.l #2,d1

next2     clr.l   d2
          move.l  #$33333333,d0
loop2     addq.l  #1,d2
          sub.b   #$10,d0
          bhi   loop2

next3     clr.l   d3
          move.b  #$0A,d0
loop3     addq.l  #1,d3
          dbra   d0,loop3      ; DBRA = DBF

next4     clr.l   d4
          move.w  #$1D,d0
loop4     addq.l  #1,d4
          dbra   d0,loop4      ; DBRA = DBF

```

### **Exercice 4 (9 points)**

Toutes les questions de cet exercice sont indépendantes. **À l'exception des registres utilisés pour renvoyer une valeur de sortie, aucun registre de donnée ou d'adresse ne devra être modifié en sortie de vos sous-programmes.** Une chaîne de caractères se termine toujours par un caractère nul (la valeur zéro). On dira qu'un caractère est blanc s'il s'agit d'un caractère *espace* ou d'un caractère *tabulation*.

1. Réalisez le sous-programme **IsBlank** qui détermine si un caractère est blanc (c'est-à-dire s'il s'agit d'un espace ou d'une tabulation).

Entrée : **D1.B** contient le code ASCII du caractère à tester.

Sortie : Si le caractère est blanc, **D0.L** renvoie 0.  
Si le caractère n'est pas blanc, **D0.L** renvoie 1.

**Indication** : La valeur numérique du code ASCII du caractère *tabulation* est 9.

2. Réalisez le sous-programme **BlankCount** qui renvoie le nombre de caractères blancs dans une chaîne de caractères. Pour savoir si un caractère est blanc, vous utiliserez le sous-programme **IsBlank**.

Entrée : **A0.L** pointe sur une chaîne de caractères.

Sortie : **D0.L** renvoie le nombre de caractères blancs de la chaîne.

**Indications :**

- Utilisez le registre **D2** comme compteur de caractères blancs (car **D0** est utilisé par **IsBlank**).
- Copier ensuite **D2** dans **D0** avant de sortir du sous-programme.

3. Réalisez le sous-programme **BlankToUnderscore** qui convertit les caractères blancs d'une chaîne de caractères en caractères *underscore*. Pour savoir si un caractère est blanc, vous utiliserez le sous-programme **IsBlank**.

Entrée : **A0.L** pointe sur une chaîne de caractères.

Sortie : Les caractères blancs de la chaîne sont remplacés par des caractères « \_ ».

**EASy68K Quick Reference v1.8**

<http://www.wowgwp.com/EASy68K.htm>

Copyright © 2004-2007 By: Chuck Kelly

| Opcode            | Size            | Operand                 | CCR     | Effective Address s=source, d=destination, e=either, i=displacement |                |      |       |       |       |          |       |       |        | Operation | Description |                |  |   |
|-------------------|-----------------|-------------------------|---------|---|----------------|------|-------|-------|-------|----------|-------|-------|--------|-----------|-------------|----------------|--|---|
|                   | BWL             | s,d                     | XNZVC   | Dn  | An             | (An) | (An)+ | -(An) | (iAn) | (iAn,Rn) | abs.W | abs.L | (i,PC) | (i,PC,Rn) | #n          |                |  |   |
| ABCD              | B               | Dy,Dx<br>-(Ay),-(Ax)    | *U*U*   | e   | -              | -    | -     | -     | -     | -        | -     | -     | -      | -         | -           | -              | $Dy_{10} + Dx_{10} + X \rightarrow Dx_{10}$<br>$-(Ay)_{10} + -(Ax)_{10} + X \rightarrow -(Ax)_{10}$                                      | Add BCD source and eXtend bit to destination, BCD result                    |
| ADD <sup>4</sup>  | BWL             | s,Dn<br>Dn,d            | *****   | e   | s              | s    | s     | s     | s     | s        | s     | s     | s      | s         | s           | s <sup>4</sup> | $s + Dn \rightarrow Dn$<br>$Dn + d \rightarrow d$  | Add binary (ADDI or ADDQ is used when source is #n. Prevent ADDQ with #n.L) |
| ADDA <sup>4</sup> | WL              | s,An                    | -----   | s   | e              | s    | s     | s     | s     | s        | s     | s     | s      | s         | s           | s              | $s + An \rightarrow An$  | Add address (.W sign-extended to .L)  |
| ADDI <sup>4</sup> | BWL             | #n,d                    | *****   | d   | -              | d    | d     | d     | d     | d        | d     | d     | -      | -         | -           | s              | $\#n + d \rightarrow d$  | Add immediate to destination  |
| ADDQ <sup>4</sup> | BWL             | #n,d                    | *****   | d   | d              | d    | d     | d     | d     | d        | d     | d     | -      | -         | -           | s              | $\#n + d \rightarrow d$  | Add quick immediate (#n range: 1 to 8)                                      |
| ADDX              | BWL             | Dy,Dx<br>-(Ay),-(Ax)    | *****   | e   | -              | -    | -     | -     | -     | -        | -     | -     | -      | -         | -           | -              | $Dy + Dx + X \rightarrow Dx$<br>$-(Ay) + -(Ax) + X \rightarrow -(Ax)$  | Add source and eXtend bit to destination                                    |
| AND <sup>4</sup>  | BWL             | s,Dn<br>Dn,d            | ---*00  | e   | -              | s    | s     | s     | s     | s        | s     | s     | s      | s         | s           | s <sup>4</sup> | $s \text{ AND } Dn \rightarrow Dn$<br>$Dn \text{ AND } d \rightarrow d$  | Logical AND source to destination (ANDI is used when source is #n)          |
| ANDI <sup>4</sup> | BWL             | #n,d                    | ---*00  | d   | -              | d    | d     | d     | d     | d        | d     | d     | -      | -         | -           | s              | $\#n \text{ AND } d \rightarrow d$   | Logical AND immediate to destination  |
| ANDI <sup>4</sup> | B               | #n,CCR                  | =====   | -   | -              | -    | -     | -     | -     | -        | -     | -     | -      | -         | -           | s              | $\#n \text{ AND } CCR \rightarrow CCR$   | Logical AND immediate to CCR  |
| ANDI <sup>4</sup> | W               | #n,SR                   | =====   | -   | -              | -    | -     | -     | -     | -        | -     | -     | -      | -         | -           | s              | $\#n \text{ AND } SR \rightarrow SR$   | Logical AND immediate to SR (Privileged)                                    |
| ASL               | BWL             | Dx,Dy                   | *****   | e   | -              | -    | -     | -     | -     | -        | -     | -     | -      | -         | -           | -              |  | Arithmetic shift Dy by Dx bits left/right                                   |
| ASR               | W               | #n,Dy                   |         | d   | -              | -    | -     | -     | -     | -        | -     | -     | -      | -         | -           | s              |  | Arithmetic shift Dy #n bits L/R (#n: 1 to 8)                                |
|                   | W               | d                       |         | -   | -              | d    | d     | d     | d     | d        | d     | d     | -      | -         | -           | -              |  | Arithmetic shift d 1 bit left/right (.W only)                               |
| Bcc               | BW <sup>4</sup> | address <sup>2</sup>    | -----   | -   | -              | -    | -     | -     | -     | -        | -     | -     | -      | -         | -           | -              | if cc true then<br>address → PC  | Branch conditionally (cc table on back) (8 or 16-bit ± offset to address)   |
| BCHG              | B L             | Dn,d<br>#n,d            | ---*--- | e   | -              | d    | d     | d     | d     | d        | d     | d     | -      | -         | -           | -              | $\text{NOT}(\text{bit number of } d) \rightarrow Z$<br>$\text{NOT}(\text{bit } n \text{ of } d) \rightarrow \text{bit } n \text{ of } d$ | Set Z with state of specified bit in d then invert the bit in d             |
| BCLR              | B L             | Dn,d<br>#n,d            | ---*--- | e   | -              | d    | d     | d     | d     | d        | d     | d     | -      | -         | -           | -              | $\text{NOT}(\text{bit number of } d) \rightarrow Z$<br>$0 \rightarrow \text{bit number of } d$   | Set Z with state of specified bit in d then clear the bit in d              |
| BRA               | BW <sup>4</sup> | address <sup>2</sup>    | -----   | -   | -              | -    | -     | -     | -     | -        | -     | -     | -      | -         | -           | -              | address → PC   | Branch always (8 or 16-bit ± offset to addr)                                |
| BSET              | B L             | Dn,d<br>#n,d            | ---*--- | e   | -              | d    | d     | d     | d     | d        | d     | d     | -      | -         | -           | -              | $\text{NOT}(\text{bit } n \text{ of } d) \rightarrow Z$<br>$1 \rightarrow \text{bit } n \text{ of } d$                                   | Set Z with state of specified bit in d then set the bit in d                |
| BSR               | BW <sup>4</sup> | address <sup>2</sup>    | -----   | -   | -              | -    | -     | -     | -     | -        | -     | -     | -      | -         | -           | -              | PC → -(SP); address → PC   | Branch to subroutine (8 or 16-bit ± offset)                                 |
| BTST              | B L             | Dn,d<br>#n,d            | ---*--- | e   | -              | d    | d     | d     | d     | d        | d     | d     | d      | d         | d           | s              | $\text{NOT}(\text{bit } Dn \text{ of } d) \rightarrow Z$<br>$\text{NOT}(\text{bit } \#n \text{ of } d) \rightarrow Z$                    | Set Z with state of specified bit in d<br>Leave the bit in d unchanged      |
| CHK               | W               | s,Dn                    | ---UUU  | e   | -              | s    | s     | s     | s     | s        | s     | s     | s      | s         | s           | s              | if $Dn < 0$ or $Dn > s$ then TRAP  | Compare Dn with 0 and upper bound [s]                                       |
| CLR               | BWL             | d                       | -0100   | d   | -              | d    | d     | d     | d     | d        | d     | d     | -      | -         | -           | -              | $0 \rightarrow d$  | Clear destination to zero   |
| CMP <sup>4</sup>  | BWL             | s,Dn                    | -----   | e   | s <sup>4</sup> | s    | s     | s     | s     | s        | s     | s     | s      | s         | s           | s <sup>4</sup> | set CCR with $Dn - s$  | Compare Dn to source  |
| CMPA <sup>4</sup> | WL              | s,An                    | -----   | s   | e              | s    | s     | s     | s     | s        | s     | s     | s      | s         | s           | s              | set CCR with $An - s$  | Compare An to source  |
| CMPI <sup>4</sup> | BWL             | #n,d                    | -----   | d   | -              | d    | d     | d     | d     | d        | d     | d     | -      | -         | -           | s              | set CCR with $d - \#n$   | Compare destination to #n   |
| CMPM <sup>4</sup> | BWL             | (Ay),-(Ax)+             | -----   | -   | -              | -    | e     | -     | -     | -        | -     | -     | -      | -         | -           | -              | set CCR with $(Ax) - (Ay)$   | Compare (Ax) to (Ay); Increment Ax and Ay                                   |
| DBcc              | W               | Dn,address <sup>2</sup> | -----   | -   | -              | -    | -     | -     | -     | -        | -     | -     | -      | -         | -           | -              | if cc false then { $Dn-1 \rightarrow Dn$<br>if $Dn < -1$ then addr → PC }  | Test condition, decrement and branch (16-bit ± offset to address)           |
| DIVS              | W               | s,Dn                    | ---*00  | e   | -              | s    | s     | s     | s     | s        | s     | s     | s      | s         | s           | s              | $\pm 32\text{bit } Dn / \pm 16\text{bit } s \rightarrow \pm Dn$  | $Dn = [16\text{-bit remainder}, 16\text{-bit quotient}]$                    |
| DIVU              | W               | s,Dn                    | ---*00  | e   | -              | s    | s     | s     | s     | s        | s     | s     | s      | s         | s           | s              | $32\text{bit } Dn / 16\text{bit } s \rightarrow Dn$  | $Dn = [16\text{-bit remainder}, 16\text{-bit quotient}]$                    |
| EOR <sup>4</sup>  | BWL             | Dn,d                    | ---*00  | e   | -              | d    | d     | d     | d     | d        | d     | d     | -      | -         | -           | s <sup>4</sup> | $Dn \text{ XOR } d \rightarrow d$  | Logical exclusive OR Dn to destination                                      |
| EORI <sup>4</sup> | BWL             | #n,d                    | ---*00  | d   | -              | d    | d     | d     | d     | d        | d     | d     | -      | -         | -           | s              | $\#n \text{ XOR } d \rightarrow d$   | Logical exclusive OR #n to destination                                      |
| EORI <sup>4</sup> | B               | #n,CCR                  | =====   | -   | -              | -    | -     | -     | -     | -        | -     | -     | -      | -         | -           | s              | $\#n \text{ XOR } CCR \rightarrow CCR$   | Logical exclusive OR #n to CCR  |
| EORI <sup>4</sup> | W               | #n,SR                   | =====   | -   | -              | -    | -     | -     | -     | -        | -     | -     | -      | -         | -           | s              | $\#n \text{ XOR } SR \rightarrow SR$   | Logical exclusive OR #n to SR (Privileged)                                  |
| EXG               | WL              | Rx,Ry                   | -----   | e   | e              | -    | -     | -     | -     | -        | -     | -     | -      | -         | -           | -              | register ↔ register  | Exchange registers (32-bit only)  |
| EXT               | WL              | Dn                      | ---*00  | d   | -              | -    | -     | -     | -     | -        | -     | -     | -      | -         | -           | -              | $Dn.B \rightarrow Dn.W \mid Dn.W \rightarrow Dn.L$   | Sign extend (change .B to .W or .W to .L)                                   |
| ILLEGAL           |                 |                         | -----   | -   | -              | -    | -     | -     | -     | -        | -     | -     | -      | -         | -           | -              | PC → -(SSP); SR → -(SSP)   | Generate Illegal Instruction exception                                      |
| JMP               |                 | d                       | -----   | -   | -              | d    | -     | -     | d     | d        | d     | d     | d      | d         | d           | -              | $\uparrow d \rightarrow PC$  | Jump to effective address of destination                                    |
| JSR               |                 | d                       | -----   | -   | -              | d    | -     | -     | d     | d        | d     | d     | d      | d         | d           | -              | PC → -(SP); $\uparrow d \rightarrow PC$  | push PC, jump to subroutine at address d                                    |
| LEA               | L               | s,An                    | -----   | -   | e              | s    | -     | -     | s     | s        | s     | s     | s      | s         | s           | -              | $\uparrow s \rightarrow An$  | Load effective address of s to An   |
| LINK              |                 | An,#n                   | -----   | -   | -              | -    | -     | -     | -     | -        | -     | -     | -      | -         | -           | -              | $An \rightarrow -(SP); SP \rightarrow An;$<br>$SP + \#n \rightarrow SP$  | Create local workspace on stack (negative n to allocate space)              |
| LSL               | BWL             | Dx,Dy                   | ***0*   | e   | -              | -    | -     | -     | -     | -        | -     | -     | -      | -         | -           | -              |  | Logical shift Dy, Dx bits left/right  |
| LSR               | W               | #n,Dy                   |         | d   | -              | -    | -     | -     | -     | -        | -     | -     | -      | -         | -           | s              |  | Logical shift Dy, #n bits L/R (#n: 1 to 8)                                  |
|                   | W               | d                       |         | -   | -              | d    | d     | d     | d     | d        | d     | d     | -      | -         | -           | -              |  | Logical shift d 1 bit left/right (.W only)                                  |
| MOVE <sup>4</sup> | BWL             | s,d                     | ---*00  | e   | s <sup>4</sup> | e    | e     | e     | e     | e        | e     | e     | s      | s         | s           | s <sup>4</sup> | $s \rightarrow d$  | Move data from source to destination  |
| MOVE              | W               | s,CCR                   | =====   | s   | -              | s    | s     | s     | s     | s        | s     | s     | s      | s         | s           | s              | $s \rightarrow CCR$  | Move source to Condition Code Register                                      |
| MOVE              | W               | s,SR                    | =====   | s   | -              | s    | s     | s     | s     | s        | s     | s     | s      | s         | s           | s              | $s \rightarrow SR$   | Move source to Status Register (Privileged)                                 |
| MOVE              | W               | SR,d                    | -----   | d   | -              | d    | d     | d     | d     | d        | d     | d     | -      | -         | -           | -              | $SR \rightarrow d$   | Move Status Register to destination   |
| MOVE              | L               | USP,An                  | -----   | -   | d              | -    | -     | -     | -     | -        | -     | -     | -      | -         | -           | -              | $USP \rightarrow An$   | Move User Stack Pointer to An (Privileged)                                  |
|                   | L               | An,USP                  | -----   | -   | s              | -    | -     | -     | -     | -        | -     | -     | -      | -         | -           | -              | $An \rightarrow USP$   | Move An to User Stack Pointer (Privileged)                                  |
|                   | BWL             | s,d                     | XNZVC   | Dn  | An             | (An) | (An)+ | -(An) | (iAn) | (iAn,Rn) | abs.W | abs.L | (i,PC) | (i,PC,Rn) | #n          |                |  |   |

| Opcode             | Size | Operand                | CCR    | Effective Address s=source, d=destination, e=either, i=displacement |    |      |       |       |        |           |       |       |        |           | Operation | Description |  |  |   |
|--------------------|------|------------------------|--------|---|----|------|-------|-------|--------|-----------|-------|-------|--------|-----------|-----------|-------------|--|--|---|
|                    |      | s,d                    | XNZVC  | Dn  | An | (An) | (An)+ | -(An) | (i,An) | (i,An,Rn) | abs.W | abs.L | (i,PC) | (i,PC,Rn) | #n        |             |  |  |   |
| MOVEA <sup>4</sup> | WL   | s,An                   | -----  | s   | e  | s    | s     | s     | s      | s         | s     | s     | s      | s         | s         | s           | s  | s → An   | Move source to An (MOVE s,An use MOVEA)   |
| MOVEM <sup>3</sup> | WL   | Rn-Rn,d<br>s,Rn-Rn     | -----  | -   | -  | d    | -     | d     | d      | d         | d     | d     | -      | -         | -         | -           | -  | Registers → d<br>s → Registers   | Move specified registers to/from memory (W source is sign-extended to L for Rn) |
| MOVEP              | WL   | Dn,(i,An)<br>(i,An),Dn | -----  | s   | -  | -    | -     | -     | d      | -         | -     | -     | -      | -         | -         | -           | -  | Dn → (i,An)...(i+2,An)...(i+4,A.<br>(i,An) → Dn...(i+2,An)...(i+4,A.   | Move Dn to/from alternate memory bytes (Access only even or odd addresses)      |
| MOVEQ <sup>4</sup> | L    | #n,Dn                  | -***00 | d   | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -         | -           | -  | #n → Dn  | Move sign extended 8-bit #n to Dn   |
| MULS               | W    | s,Dn                   | -***00 | e   | -  | s    | s     | s     | s      | s         | s     | s     | s      | s         | s         | s           | s  | ±16bit s * ±16bit Dn → ±Dn   | Multiply signed 16-bit; result: signed 32-bit                                   |
| MULU               | W    | s,Dn                   | -***00 | e   | -  | s    | s     | s     | s      | s         | s     | s     | s      | s         | s         | s           | s  | 16bit s * 16bit Dn → Dn  | Multiply unsig'd 16-bit; result: unsig'd 32-bit                                 |
| NBCD               | B    | d                      | *U*U*  | d   | -  | d    | d     | d     | d      | d         | d     | d     | -      | -         | -         | -           | -  | 0 - d <sub>10</sub> - X → d  | Negate BCD with eXtend, BCD result  |
| NEG                | BWL  | d                      | *****  | d   | -  | d    | d     | d     | d      | d         | d     | d     | -      | -         | -         | -           | -  | 0 - d → d  | Negate destination (2's complement)   |
| NEGX               | BWL  | d                      | *****  | d   | -  | d    | d     | d     | d      | d         | d     | d     | -      | -         | -         | -           | -  | 0 - d - X → d  | Negate destination with eXtend  |
| NOP                |      |                        | -----  | -   | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -         | -           | -  | None   | No operation occurs   |
| NOT                | BWL  | d                      | -***00 | d   | -  | d    | d     | d     | d      | d         | d     | d     | -      | -         | -         | -           | -  | NOT( d ) → d   | Logical NOT destination (1's complement)  |
| OR <sup>4</sup>    | BWL  | s,Dn<br>Dn,d           | -***00 | e   | -  | s    | s     | s     | s      | s         | s     | s     | s      | s         | s         | s           | s  | s OR Dn → Dn<br>Dn OR d → d  | Logical OR<br>(ORI is used when source is #n)                                   |
| ORI <sup>4</sup>   | BWL  | #n,d                   | -***00 | d   | -  | d    | d     | d     | d      | d         | d     | d     | -      | -         | -         | s           | #n OR d → d  | Logical OR #n to destination   |   |
| ORI <sup>4</sup>   | B    | #n,CCR                 | =====  | -   | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -         | s           | #n OR CCR → CCR  | Logical OR #n to CCR   |   |
| ORI <sup>4</sup>   | W    | #n,SR                  | =====  | -   | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -         | s           | #n OR SR → SR  | Logical OR #n to SR (Privileged)   |   |
| PEA                | L    | s                      | -----  | -   | -  | s    | -     | -     | s      | s         | s     | s     | s      | s         | s         | -           | -  | ↑s → -(SP)   | Push effective address of s onto stack  |
| RESET              |      |                        | -----  | -   | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -         | -           | -  | Assert RESET Line  | Issue a hardware RESET (Privileged)   |
| ROL                | BWL  | Dx,Dy                  | -***0* | e   | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -         | -           | -  | Rotate Dy, Dx bits left/right (without X)  |   |
| ROR                | W    | #n,Dy<br>d             |        | d   | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -         | -           | -  | Rotate Dy, #n bits left/right (#n: 1 to 8)<br>Rotate d 1-bit left/right (.W only)  |   |
| ROXL               | BWL  | Dx,Dy                  | ***0*  | e   | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -         | -           | -  | Rotate Dy, Dx bits L/R, X used then updated  |   |
| ROXR               | W    | #n,Dy<br>d             |        | d   | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -         | -           | -  | Rotate Dy, #n bits left/right (#n: 1 to 8)<br>Rotate destination 1-bit left/right (.W only)                                      |   |
| RTE                |      |                        | =====  | -   | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -         | -           | -  | (SP)+ → SR; (SP)+ → PC   | Return from exception (Privileged)  |
| RTR                |      |                        | =====  | -   | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -         | -           | -  | (SP)+ → CCR; (SP)+ → PC  | Return from subroutine and restore CCR  |
| RTS                |      |                        | -----  | -   | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -         | -           | -  | (SP)+ → PC   | Return from subroutine  |
| SBCD               | B    | Dy,Dx<br>-(Ay),-(Ax)   | *U*U*  | e   | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -         | -           | -  | Dx <sub>10</sub> - Dy <sub>10</sub> - X → Dx <sub>10</sub><br>-(Ax) <sub>10</sub> - (Ay) <sub>10</sub> - X → -(Ax) <sub>10</sub> | Subtract BCD source and eXtend bit from destination, BCD result                 |
| Scc                | B    | d                      | -----  | d   | -  | d    | d     | d     | d      | d         | d     | d     | -      | -         | -         | -           | -  | If cc is true then 1's → d<br>else 0's → d   | If cc true then d.B = 11111111<br>else d.B = 00000000                           |
| STOP               |      | #n                     | =====  | -   | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -         | s           | #n → SR; STOP  | Move #n to SR, stop processor (Privileged)   |   |
| SUB <sup>4</sup>   | BWL  | s,Dn<br>Dn,d           | *****  | e   | s  | s    | s     | s     | s      | s         | s     | s     | s      | s         | s         | s           | s  | Dn - s → Dn<br>d - Dn → d  | Subtract binary (SUBI or SUBQ used when source is #n. Prevent SUBQ with #n.L)   |
| SUBA <sup>4</sup>  | WL   | s,An                   | -----  | s   | e  | s    | s     | s     | s      | s         | s     | s     | s      | s         | s         | s           | s  | An - s → An  | Subtract address (.W sign-extended to L)  |
| SUBI <sup>4</sup>  | BWL  | #n,d                   | *****  | d   | -  | d    | d     | d     | d      | d         | d     | d     | -      | -         | -         | s           | d - #n → d   | Subtract immediate from destination  |   |
| SUBQ <sup>4</sup>  | BWL  | #n,d                   | *****  | d   | d  | d    | d     | d     | d      | d         | d     | d     | -      | -         | -         | s           | d - #n → d   | Subtract quick immediate (#n range: 1 to 8)  |   |
| SUBX               | BWL  | Dy,Dx<br>-(Ay),-(Ax)   | *****  | e   | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -         | -           | -  | Dx - Dy - X → Dx<br>-(Ax) - (Ay) - X → -(Ax)   | Subtract source and eXtend bit from destination                                 |
| SWAP               | W    | Dn                     | -***00 | d   | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -         | -           | -  | bits[31:16] ↔ bits[15:0]   | Exchange the 16-bit halves of Dn  |
| TAS                | B    | d                      | -***00 | d   | -  | d    | d     | d     | d      | d         | d     | d     | -      | -         | -         | -           | -  | test d → CCR; 1 → bit7 of d  | N and Z set to reflect d, bit7 of d set to 1                                    |
| TRAP               |      | #n                     | -----  | -   | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -         | s           | PC → -(SSP); SR → -(SSP);<br>(vector table entry) → PC | Push PC and SR, PC set by vector table #n (#n range: 0 to 15)  |   |
| TRAPV              |      |                        | -----  | -   | -  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -         | -           | -  | If V then TRAP #7  | If overflow, execute an Overflow TRAP   |
| TST                | BWL  | d                      | -***00 | d   | -  | d    | d     | d     | d      | d         | d     | d     | -      | -         | -         | -           | -  | test d → CCR   | N and Z set to reflect destination  |
| UNLK               |      | An                     | -----  | -   | d  | -    | -     | -     | -      | -         | -     | -     | -      | -         | -         | -           | -  | An → SP; (SP)+ → An  | Remove local workspace from stack   |
|                    | BWL  | s,d                    | XNZVC  | Dn  | An | (An) | (An)+ | -(An) | (i,An) | (i,An,Rn) | abs.W | abs.L | (i,PC) | (i,PC,Rn) | #n        |             |  |  |   |

| Condition Tests (+ OR, ! NOT, ⊕ XOR; ° Unsigned, ° Alternate cc) |                |          |    |                  |                |
|--|----------------|----------|----|------------------|----------------|
| cc   | Condition      | Test     | cc | Condition        | Test           |
| T  | true           | I        | VC | overflow clear   | IV             |
| F  | false          | O        | VS | overflow set     | V              |
| HI <sup>o</sup>  | higher than    | I(C + Z) | PL | plus             | IN             |
| LS <sup>o</sup>  | lower or same  | C + Z    | MI | minus            | N              |
| HS <sup>o</sup> , CC <sup>o</sup>                                | higher or same | IC       | GE | greater or equal | !(N ⊕ V)       |
| LO <sup>o</sup> , CS <sup>o</sup>                                | lower than     | C        | LT | less than        | (N ⊕ V)        |
| NE   | not equal      | IZ       | GT | greater than     | !((N ⊕ V) + Z) |
| EQ   | equal          | Z        | LE | less or equal    | (N ⊕ V) + Z    |

**An** Address register (16/32-bit, n=0-7)  
**Dn** Data register (8/16/32-bit, n=0-7)  
**Rn** any data or address register  
**s** Source, **d** Destination  
**e** Either source or destination  
**#n** Immediate data, **i** Displacement  
**BCD** Binary Coded Decimal  
**↑** Effective address  
**1** Long only; all others are byte only  
**2** Assembler calculates offset  
**3** Branch sizes: **.B** or **.S** -128 to +127 bytes, **.W** or **.L** -32768 to +32767 bytes  
**4** Assembler automatically uses A, I, Q or M form if possible. Use #n.L to prevent Quick optimization

**SSP** Supervisor Stack Pointer (32-bit)  
**USP** User Stack Pointer (32-bit)  
**SP** Active Stack Pointer (same as A7)  
**PC** Program Counter (24-bit)  
**SR** Status Register (16-bit)  
**CCR** Condition Code Register (lower 8-bits of SR)  
**N** negative, **Z** zero, **V** overflow, **C** carry, **X** extend  
 \* set according to operation's result, ⊕ set directly  
 - not affected, 0 cleared, 1 set, U undefined

Revised by Peter Csaszar, Lawrence Tech University – 2004-2006

Distributed under the GNU general public use license.

Nom : ..... Prénom : ..... Classe : .....

**DOCUMENT RÉPONSE À RENDRE**

**Exercice 1**

| Instruction                | Mémoire                                 | Registre                           |
|----------------------------|---|------------------------------------|
| Exemple                    | \$005000 54 AF <b>00 40</b> E7 21 48 C0 | A0 = \$00005004<br>A1 = \$0000500C |
| Exemple                    | \$005008 C9 10 11 C8 D4 36 <b>FF</b> 88 | Aucun changement                   |
| MOVE.L #1234,2(A2)         |   |                                    |
| MOVE.W \$5012,7(A1,D0.W)   |   |                                    |
| MOVE.B -(A1),10(A0)        |   |                                    |
| MOVE.B 1(A1),-128(A2,D1.L) |   |                                    |

**Exercice 2**

| Opération  | Taille (bits) | Nombre manquant (hexadécimal) | N | Z | V | C |
|------------|---------------|-------------------------------|---|---|---|---|
| \$7F + \$? | 8             |                               | 1 | 0 | 0 | 0 |
| \$7F + \$? | 16            |                               | 1 | 0 | 0 | 0 |
| \$7F + \$? | 32            |                               | 1 | 0 | 0 | 0 |

**Exercice 3**

|   |         |
|---|---------|
| Valeurs des registres après exécution du programme.<br>Utilisez la représentation hexadécimale sur 32 bits. |         |
| D1 = \$   | D3 = \$ |
| D2 = \$   | D4 = \$ |

**Exercice 4**

IsBlank

BlankCount

BlankToUnderscore