

Examen (Session 1) – Corrigé

Architecture des ordinateurs

Durée : 1 h 30

Inscrivez vos réponses **exclusivement** sur le document réponse.
Ne pas détailler les calculs sauf si cela est explicitement demandé.
Ne pas écrire, ni à l'encre rouge ni au crayon à papier.

Exercice 1 (3 points)

1. Simplifiez au maximum les expressions présentes sur le [document réponse](#) (pas de détail). Le résultat ne devra pas contenir de parenthèses.
2. Remplissez le diagramme de Karnaugh présent sur le [document réponse](#) et donnez son expression logique associée.

Remarques :

- Aucun point ne sera attribué à une expression si son tableau est faux.
- La simplification à l'aide de OU EXCLUSIF n'est pas demandée.

Exercice 2 (5 points)

Un système à microprocesseur comporte une mémoire morte (ROM), une mémoire vive (RAM) et deux périphériques (**P1** et **P2**). Leurs capacités (en octets) sont respectivement 64 Kio, 8 Kio, 2 Kio et 512 octets. Le microprocesseur possède un bus d'adresse de 20 bits (les bits d'adresse sont numérotés de A0 à A19 et A0 est le bit poids faible). Tous les composants ont un bus de donnée de 8 bits. La ROM sera située dans les adresses les plus faibles, viendront ensuite la RAM, **P1** et **P2**.

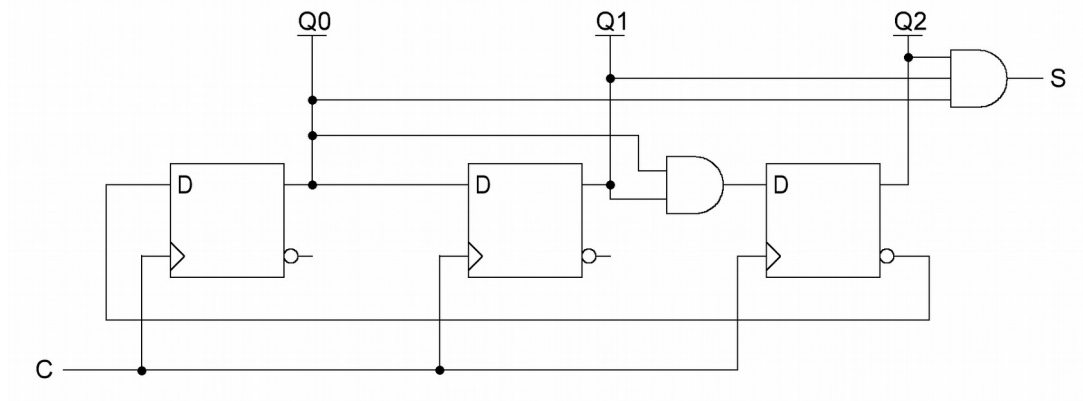
1. Donnez la taille du bus d'adresse de chaque mémoire et de chaque périphérique.
2. Est-il possible de réaliser un décodage de type linéaire ?

Pour tout le reste de l'exercice, c'est le mode zone qui sera utilisé avec le moins de zones possible.

3. Donnez les bits de sélection qui serviront au décodage.
4. En tenant compte du signal AS (*Address Strobe*) que fournit le microprocesseur et qui indique si la valeur sur son bus d'adresse est valide, donnez les expressions des signaux CS pour chaque composant relié au microprocesseur.
5. Donnez les adresses hautes et basses de chaque composant (vous utiliserez la représentation hexadécimale à 5 chiffres).
6. Quel est le nombre d'images pour chaque composant ?

Exercice 3 (2 points)

Complétez les chronogrammes sur le [document réponse](#) (jusqu'à la dernière ligne verticale pointillée) pour le montage ci-dessous.

**Exercice 4 (3 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 = \$FFFF0005 A0 = \$00005000 PC = \$00006000
 D1 = \$10000002 A1 = \$00005008
 D2 = \$FFFFFFFF 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 5 (3 points)

Remplissez le tableau présent sur le [document réponse](#). Donnez le résultat des additions ainsi que le contenu des bits N, Z, V et C du registre d'état.

Exercice 6 (4 points)

Soit le programme ci-dessous :

```

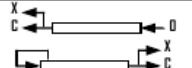
Main      move.l    #$12345678,d7
next1     moveq.l   #1,d1
          cmpi.b    #$80,d7
          blt       next2
          moveq.l   #2,d1
next2     move.l    d7,d2
          swap      d2
          ror.b     #4,d2
          rol.l     #8,d2
          rol.w     #4,d2
next3     clr.l     d3
          move.l    d7,d0
loop3     addq.l    #1,d3
          subq.w    #1,d0
          bne       loop3
next4     clr.l     d4
          move.l    d7,d0
loop4     addq.l    #1,d4
          dbra      d0,loop4      ; DBRA = DBF
quit      illegal

```

Complétez le tableau présent sur le [document réponse](#).

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 -(Ay),-(Ax) | *U*U* | e | - | - | - | - | - | - | - | - | - | - | - | $Dy_{10} + Dx_{10} + X \rightarrow Dx_{10}$ $-(Ay)_{10} + -(Ax)_{10} + X \rightarrow -(Ax)_{10}$ | Add BCD source and eXtend bit to destination, BCD result |
| ADD ⁴ | BWL | s,Dn Dn,d | ***** | e | s | s | s | s | s | s | s | s | s | s | s ⁴ | $s + Dn \rightarrow Dn$ $Dn + d \rightarrow d$ | Add binary (ADDI or ADDQ is used when source is #n. Prevent ADDQ with #n.L) |
| ADDA ⁴ | WL | s,An | ----- | s | e | s | s | s | s | s | s | s | s | s | s | $s + An \rightarrow An$ | Add address (.W sign-extended to .L) |
| ADDI ⁴ | BWL | #n,d | ***** | d | - | d | d | d | d | d | d | d | - | - | s | $\#n + d \rightarrow d$ | Add immediate to destination |
| ADDQ ⁴ | 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 -(Ay),-(Ax) | ***** | e | - | - | - | - | - | - | - | - | - | - | - | $Dy + Dx + X \rightarrow Dx$ $-(Ay) + -(Ax) + X \rightarrow -(Ax)$ | Add source and eXtend bit to destination |
| AND ⁴ | BWL | s,Dn Dn,d | -**00 | e | - | s | s | s | s | s | s | s | s | s | s ⁴ | $s \text{ AND } Dn \rightarrow Dn$ $Dn \text{ AND } d \rightarrow d$ | Logical AND source to destination (ANDI is used when source is #n) |
| ANDI ⁴ | 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 ⁴ | B | #n,CCR | ===== | - | - | - | - | - | - | - | - | - | - | - | s | $\#n \text{ AND } CCR \rightarrow CCR$ | Logical AND immediate to CCR |
| ANDI ⁴ | 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) |
| Bcc | BW ⁴ | address ² | ----- | - | - | - | - | - | - | - | - | - | - | - | - | if cc true then address \rightarrow PC | Branch conditionally (cc table on back) (8 or 16-bit \pm offset to address) |
| BCHG | B L | Dn,d #n,d | ---*-- | e ¹ | - | d | d | d | d | d | d | d | - | - | - | $\text{NOT}(\text{bit number of } d) \rightarrow Z$ $\text{NOT}(\text{bit } n \text{ of } d) \rightarrow \text{bit } n \text{ of } d$ | Set Z with state of specified bit in d then invert the bit in d |
| BCLR | B L | Dn,d #n,d | ---*-- | e ¹ | - | d | d | d | d | d | d | d | - | - | - | $\text{NOT}(\text{bit number of } d) \rightarrow Z$ $0 \rightarrow \text{bit number of } d$ | Set Z with state of specified bit in d then clear the bit in d |
| BRA | BW ⁴ | address ² | ----- | - | - | - | - | - | - | - | - | - | - | - | - | address \rightarrow PC | Branch always (8 or 16-bit \pm offset to addr) |
| BSET | B L | Dn,d #n,d | ---*-- | e ¹ | - | d | d | d | d | d | d | d | - | - | - | $\text{NOT}(\text{bit } n \text{ of } d) \rightarrow Z$ $1 \rightarrow \text{bit } n \text{ of } d$ | Set Z with state of specified bit in d then set the bit in d |
| BSR | BW ⁴ | address ² | ----- | - | - | - | - | - | - | - | - | - | - | - | - | PC \rightarrow -(SP); address \rightarrow PC | Branch to subroutine (8 or 16-bit \pm offset) |
| BTST | B L | Dn,d #n,d | ---*-- | e ¹ | - | d | d | d | d | d | d | d | d | d | s | $\text{NOT}(\text{bit } Dn \text{ of } d) \rightarrow Z$ $\text{NOT}(\text{bit } \#n \text{ of } d) \rightarrow Z$ | Set Z with state of specified bit in d Leave the bit in d unchanged |
| CHK | W | s,Dn | -*UUU | e | - | 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 ⁴ | BWL | s,Dn | ***** | e | s ⁴ | s | s | s | s | s | s | s | s | s | s ⁴ | set CCR with $Dn - s$ | Compare Dn to source |
| CMPA ⁴ | WL | s,An | ***** | s | e | s | s | s | s | s | s | s | s | s | s | set CCR with $An - s$ | Compare An to source |
| CMPI ⁴ | BWL | #n,d | ***** | d | - | d | d | d | d | d | d | d | - | - | s | set CCR with $d - \#n$ | Compare destination to #n |
| CMPP ⁴ | BWL | (Ay),-(Ax)+ | ***** | - | - | - | e | - | - | - | - | - | - | - | - | set CCR with $(Ax) - (Ay)$ | Compare (Ax) to (Ay); Increment Ax and Ay |
| DBcc | W | Dn,address ² | ----- | - | - | - | - | - | - | - | - | - | - | - | - | if cc false then { $Dn-1 \rightarrow Dn$ if $Dn < -1$ then addr \rightarrow PC } | Test condition, decrement and branch (16-bit \pm offset to address) |
| DIVS | W | s,Dn | *****0 | e | - | 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 | *****0 | e | - | 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 ⁴ | BWL | Dn,d | -**00 | e | - | d | d | d | d | d | d | d | - | - | s ⁴ | $Dn \text{ XOR } d \rightarrow d$ | Logical exclusive OR Dn to destination |
| EORI ⁴ | 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 ⁴ | B | #n,CCR | ===== | - | - | - | - | - | - | - | - | - | - | - | s | $\#n \text{ XOR } CCR \rightarrow CCR$ | Logical exclusive OR #n to CCR |
| EORI ⁴ | W | #n,SR | ===== | - | - | - | - | - | - | - | - | - | - | - | s | $\#n \text{ XOR } SR \rightarrow SR$ | Logical exclusive OR #n to SR (Privileged) |
| EXG | L | Rx,Ry | ----- | e | e | - | - | - | - | - | - | - | - | - | - | register \leftrightarrow 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 \rightarrow -(SSP); SR \rightarrow -(SSP) | Generate Illegal Instruction exception |
| JMP | | 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 | - | PC \rightarrow -(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 | - | $\uparrow s \rightarrow$ An | Load effective address of s to An |
| LINK | | An,#n | ----- | - | - | - | - | - | - | - | - | - | - | - | - | $An \rightarrow$ -(SP); $SP \rightarrow$ An; $SP + \#n \rightarrow SP$ | 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) |
| MOVE ⁴ | BWL | s,d | -**00 | e | s ⁴ | e | e | e | e | e | e | e | s | s | s ⁴ | $s \rightarrow d$ | Move data from source to destination |
| MOVE | W | s,CCR | ===== | s | - | s | s | s | s | s | s | s | s | s | s | $s \rightarrow$ CCR | Move source to Condition Code Register |
| MOVE | W | s,SR | ===== | 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) |
| | | 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 |
|--------------------|------|----------------------|--------|---|----|------|-------|-------|-------|----------|-------|-------|--------|-----------|----|----------------|---|--|
| | BWL | s,d | XNZVC | Dn | An | (An) | (An)+ | -(An) | (iAn) | (iAn,Rn) | abs.W | abs.L | (i,PC) | (i,PC,Rn) | #n | | | |
| MOVEA ⁴ | WL | s,An | ----- | s | e | s | s | s | s | s | s | s | s | s | s | s | s → An | Move source to An (MOVE s,An use MOVEA) |
| MOVEM ³ | WL | Rn-Rn,d s,Rn-Rn | ----- | - | - | d | - | d | d | d | d | d | - | - | - | - | Registers → d s → Registers | Move specified registers to/from memory (W source is sign-extended to .L for Rn) |
| MOVEP | WL | Dn,(iAn) (iAn),Dn | ----- | s | - | - | - | - | d | - | - | - | - | - | - | - | Dn → (iAn)...(i+2,An)...(i+4,A. (iAn) → Dn...(i+2,An)...(i+4,A. | Move Dn to/from alternate memory bytes (Access only even or odd addresses) |
| MOVEQ ⁴ | L | #n,Dn | -***00 | d | - | - | - | - | - | - | - | - | - | - | - | s | #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 | ±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 | 16bit s * 16bit Dn → Dn | Multiply unsigned 16-bit; result: unsigned 32-bit |
| NBCD | B | d | *U*U* | d | - | d | d | d | d | d | d | d | - | - | - | - | 0 - d ₁₀ - 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 ⁴ | BWL | s,Dn Dn,d | -***00 | e | - | s | s | s | s | s | s | s | s | s | s | s ⁴ | s OR Dn → Dn Dn OR d → d | Logical OR (ORI is used when source is #n) |
| ORI ⁴ | BWL | #n,d | -***00 | d | - | d | d | d | d | d | d | d | - | - | - | s | #n OR d → d | Logical OR #n to destination |
| ORI ⁴ | B | #n,CCR | ===== | - | - | - | - | - | - | - | - | - | - | - | - | s | #n OR CCR → CCR | Logical OR #n to CCR |
| ORI ⁴ | 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 #n,Dy | -***0* | e | - | - | - | - | - | - | - | - | - | - | - | - |  | Rotate Dy, Dx bits left/right (without X) Rotate Dy, #n bits left/right (#n: 1 to 8) Rotate d 1-bit left/right (.W only) |
| ROXL | BWL | Dx,Dy #n,Dy | ***0* | e | - | - | - | - | - | - | - | - | - | - | - | - |  | Rotate Dy, Dx bits L/R, X used then updated Rotate Dy, #n bits left/right (#n: 1 to 8) Rotate destination 1-bit left/right (.W only) |
| ROXR | W | d | | - | - | d | d | d | d | d | d | d | - | - | - | - | | |
| 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 -(Ay),-(Ax) | *U*U* | e | - | - | - | - | - | - | - | - | - | - | - | - | Dx ₁₀ - Dy ₁₀ - X → Dx ₁₀ -(Ax) ₁₀ - -(Ay) ₁₀ - X → -(Ax) ₁₀ | 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 else 0's → d | If cc true then d.B = 11111111 else d.B = 00000000 |
| STOP | | #n | ===== | - | - | - | - | - | - | - | - | - | - | - | - | s | #n → SR; STOP | Move #n to SR, stop processor (Privileged) |
| SUB ⁴ | BWL | s,Dn Dn,d | ***** | e | s | s | s | s | s | s | s | s | s | s | s | s ⁴ | Dn - s → Dn d - Dn → d | Subtract binary (SUBI or SUBQ used when source is #n. Prevent SUBQ with #n.L) |
| SUBA ⁴ | WL | s,An | ----- | s | e | s | s | s | s | s | s | s | s | s | s | s | An - s → An | Subtract address (W sign-extended to .L) |
| SUBI ⁴ | BWL | #n,d | ***** | d | - | d | d | d | d | d | d | d | - | - | - | s | d - #n → d | Subtract immediate from destination |
| SUBQ ⁴ | 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 -(Ay),-(Ax) | ***** | e | - | - | - | - | - | - | - | - | - | - | - | - | Dx - Dy - X → Dx -(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); (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) | (iAn) | (iAn,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 ^a | higher than | I(C + Z) | PL | plus | IN |
| LS ^a | lower or same | C + Z | MI | minus | N |
| HS ^a , CC ^a | higher or same | IC | GE | greater or equal | !(N ⊕ V) |
| LO ^a , CS ^a | 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 |

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

Distributed under the GNU general public use license.

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, **O** cleared, **I** set, **U** undefined

Nom : Prénom : No étudiant :

DOCUMENT RÉPONSE

Cadre réservé au correcteur.

Exercice 1

| Expression non simplifiée | Expression simplifiée (sans parenthèses) |
|--|--|
| $A + B.D + \overline{A}.(\overline{B} + \overline{D}).(A.C + D)$ | $A + D$ |
| $\overline{\overline{A}.B + A.C + \overline{A}.B.C}$ | $A.B + A.C$ |

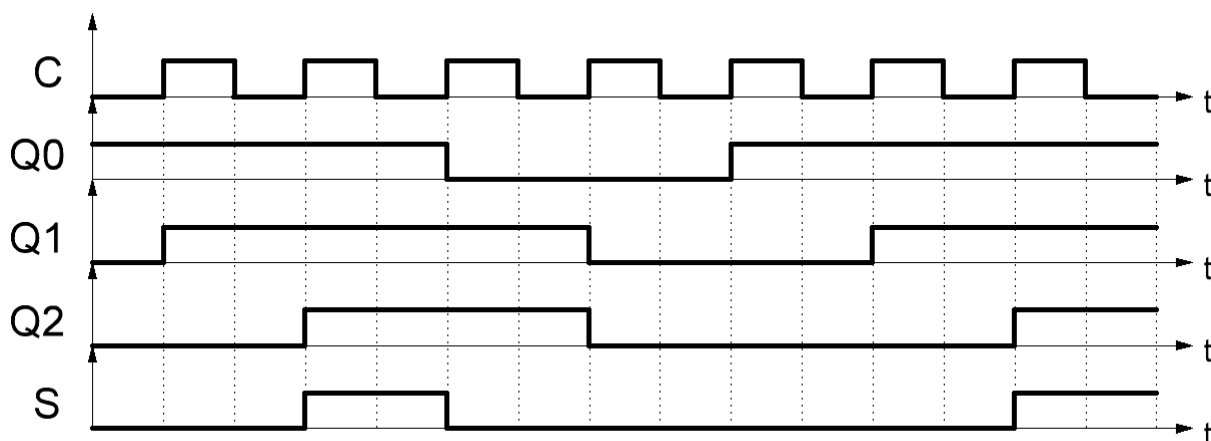
| | | | | | |
|----|----|----|----|----|----|
| | | CD | | | |
| | X | 00 | 01 | 11 | 10 |
| AB | 00 | 1 | 1 | 1 | 1 |
| | 01 | 0 | 1 | 1 | 0 |
| | 11 | 1 | 1 | 1 | 1 |
| | 10 | 1 | 1 | 1 | 1 |

$$X = A + \overline{B} + D$$

Exercice 2

| | | | |
|-----------------------------------|-----------------------------|--|------------------------------------|
| 1. ROM : 16 bits RAM : 13 bits | P1 : 11 bits P2 : 9 bits | 2. Décodage linéaire possible ? Oui | 3. Bits de sélection : A19, A18 |
|-----------------------------------|-----------------------------|--|------------------------------------|

| Composant | 4. | 5. | | 6. |
|-----------|---|---------------|---------------|-----------------|
| | CS | Adresse basse | Adresse haute | Nombre d'images |
| ROM | $CS_{ROM} = AS.\overline{A19}.\overline{A18}$ | 00000 | 0FFFF | 4 |
| RAM | $CS_{RAM} = AS.\overline{A19}.A18$ | 40000 | 41FFF | 32 |
| P1 | $CS_{P1} = AS.A19.\overline{A18}$ | 80000 | 807FF | 128 |
| P2 | $CS_{P2} = AS.A19.A18$ | C0000 | C01FF | 512 |

Exercice 3**Exercice 4**

| Instruction | Mémoire | Registre |
|-----------------------|---|------------------------------------|
| Exemple | \$005000 54 AF 00 40 E7 21 48 C0 | A0 = \$00005004 A1 = \$0000500C |
| Exemple | \$005008 C9 10 11 C8 D4 36 FF 88 | Aucun changement |
| MOVE.W \$5002, -4(A1) | \$005000 54 AF 18 B9 18 B9 48 C0 | Aucun changement |
| MOVE.L #\$5002, (A0)+ | \$005000 00 00 50 02 E7 21 48 C0 | A0 = \$00005004 |
| MOVE.B -1(A2), -(A1) | \$005000 54 AF 18 B9 E7 21 48 88 | A1 = \$00005007 |

Exercice 5

| Opération | Taille (bits) | Résultat (hexadécimal) | N | Z | V | C |
|-------------------------|---------------|------------------------|---|---|---|---|
| \$52 + \$3A | 8 | \$8C | 1 | 0 | 1 | 0 |
| \$52 + \$3A | 16 | \$008C | 0 | 0 | 0 | 0 |
| \$12345678 + \$FFFFFFFF | 32 | \$12345677 | 0 | 0 | 0 | 1 |

Exercice 6

| Valeurs des registres après exécution du programme. Utilisez la représentation hexadécimale sur 32 bits. | |
|---|-----------------|
| D1 = \$00000002 | D3 = \$00005678 |
| D2 = \$78123564 | D4 = \$00005679 |