# Graphes
# QCM 5
## 24 novembre 2025

1. **Dans un graphe non orienté, s'il existe une chaîne $x \rightsquigarrow y$ pour toute paire de sommets disjoints $\{x, y\}$, le graphe est**
   - (a) complet
   - (b) transitif
   - (c) connexe
   - (d) fortement connexe
   - (e) parfait

2. **Si lors du parcours largeur d'un graphe non orienté à partir d'un sommet quelconque, tous les sommets sont marqués sans avoir eu besoin de relancer le parcours dans la fonction d'appel, alors le graphe est**
   - (a) connexe
   - (b) complet
   - (c) fortement connexe
   - (d) parfait

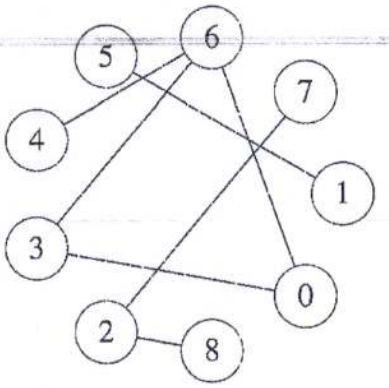
3. **Considérons les forêts couvrantes des parcours largeur et profondeur d'un même graphe non orienté (en choisissant les sommets en ordre croissant).**
   - (a) Il y a toujours le même nombre d'arbres dans les deux forêts couvrantes.
   - (b) La forêt couvrante du parcours profondeur peut contenir plus d'arbres.
   - (c) La forêt couvrante du parcours largeur peut contenir plus d'arbres.

4. **Un graphe non orienté d'ordre $n$ peut être connexe à partir de**
   - (a) $n-1$ arêtes
   - (b) $n$ arêtes
   - (c) $n+1$ arêtes
   - (d) $2n$ arêtes

5. **Soit $G$ un graphe non orienté connexe, sa fermeture transitive est**
   - (a) un sous-graphe de $G$
   - (b) un graphe partiel de $G$
   - (c) un graphe connexe
   - (d) un graphe complet
   - (e) un graphe parfait

6. **Soit $G =< S, A >$ un graphe non orienté d'ordre $N$ avec $M$=Card(A), la complexité de l'algorithme de Warshall sur $G$ est en**
   - (a) $O(max(N, M))$
   - (b) $O(N^2)$
   - (c) $O(N.M)$
   - (d) $O(N^3)$
   - (e) $O(M^2)$

Soit le graphe non orienté $G$ représenté par la figure suivante.




7. **Combien de composantes connexes a le graphe $G$ ?**
   (a) 1
   (b) 2
   (c) 3
   (d) 4

8. **Le sous-graphe de $G$ issu de l'ensemble de sommets $\{2, 7, 8\}$ est une composante connexe de $G$.**
   (a) Vrai
   (b) Faux
   (c) Ça dépend

9. **Le sous-graphe de $G$ issu de l'ensemble de sommets $\{0, 3, 6\}$ est une composante connexe de $G$.**
   (a) Vrai
   (b) Faux
   (c) Ça dépend

10. **Lors de la construction de la fermeture transitive de $G$, combien d'arêtes vont être ajoutées à celles déjà existantes dans $G$ ?**
    (a) 1
    (b) 2
    (c) 3
    (d) 4

# QCM N°6

Lundi 24 novembre 2025

## Question 11

Dans $E = \mathbb{R}^3$, on considère la famille $\mathcal{F} = \big(u_1{=}(1,0,0), u_2{=}(1,1,0), u_3{=}(4,1,0)\big)$. Alors :

a. $\text{Vect}\,\mathcal{F} = \{0_E\}$

b. $\text{Vect}\,\mathcal{F}$ est une droite

X X c. $\text{Vect}\,\mathcal{F}$ est un plan

d. $\text{Vect}\,\mathcal{F} = E$

e. Aucun des autres choix

## Question 12

Dans $E = \mathbb{R}_2[X]$, on considère la famille $\mathcal{F} = \big(X-1, X^2-X, X^2\big)$. Alors la dimension de $\text{Vect}\,\mathcal{F}$ vaut :

a. $\dim\big(\text{Vect}\,\mathcal{F}\big) = 0$

b. $\dim\big(\text{Vect}\,\mathcal{F}\big) = 1$

? c. $\dim\big(\text{Vect}\,\mathcal{F}\big) = 2$

X d. $\dim\big(\text{Vect}\,\mathcal{F}\big) = 3$

e. Aucun des autres choix

## Question 13

Dans $E = \mathbb{R}_2[X]$, on considère une famille de la forme $\mathcal{F} = (P_1, P_2, P_3, P_4)$. Alors on sait que :

a. La famille $\mathcal{F}$ est libre

X X b. La famille $\mathcal{F}$ est liée

c. $\mathcal{F}$ est une famille génératrice de $E$

d. $\mathcal{F}$ n'est pas une famille génératrice de $E$

e. Aucun des autres choix

## Question 14

Dans $E = \mathbb{R}^2$, on considère la base $\mathcal{B} = (\varepsilon_1{=}(1,-1), \varepsilon_2 = (2,1))$.

Soit $u \in E$ dont les coordonnées dans $\mathcal{B}$ sont $X = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$. Alors ce vecteur $u$ est égal à :

X a. $u = (5,1)$

b. $u = (1,2)$

c. $u = (-1,1)$

d. $u = (1,0)$

e. Aucun des autres choix

## Question 15

Soit la matrice $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$. Alors le déterminant de $A$ vaut :

a. $\det(A) = 2$

X b. $\det(A) = -2$

c. $\det(A) = 10$

d. $\det(A) = -5$

e. Aucun des autres choix

## Question 16

Soit la matrice $A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 2 \\ 3 & 0 & 4 \end{pmatrix}$. Alors le déterminant de $A$ vaut :

X a. $\det(A) = 2$

b. $\det(A) = -2$

c. $\det(A) = 10$

d. $\det(A) = -5$

e. Aucun des autres choix

## Question 17

Soit $(A, B) \in \left(\mathcal{M}_3(\mathbb{R})\right)^2$. Alors :

a. $\det(A+B) = \det(A) + \det(B)$

b. $\det(2A) = 2\det(A)$

X X c. $\det(AB) = \det(A) \times \det(B)$

d. Aucun des autres choix

## Question 18

Soient $A \in \mathcal{M}_3(\mathbb{R})$ et $\lambda \in \mathbb{R}$. Alors $\lambda$ est valeur propre de $A$ si et seulement si :

a. $\exists u \in \mathbb{R}^3,\ u \neq 0_{\mathbb{R}^3}$ et $\lambda A u = 0_{\mathbb{R}^3}$

b. $Au = \lambda u$

c. $\exists u \in \mathbb{R}^3,\ Au = \lambda u$

X X d. $\exists u \in \mathbb{R}^3,\ u \neq 0_{\mathbb{R}^3}$ et $Au = \lambda u$

e. Aucun des autres choix

## Question 19

Soient $A \in \mathcal{M}_3(\mathbb{R})$ et $I_3$ la matrice identité de $\mathcal{M}_3(\mathbb{R})$. Alors le polynôme caractéristique de $A$ est :

X X a. $P_A(X) = \det(A - XI_3)$

b. $P_A(X) = \det(XA - I_3)$

c. $P_A(X) = \det(A + XI_3)$

d. $P_A(X) = \det(XA + I_3)$

e. Aucun des autres choix

## Question 20

Soit $A = \begin{pmatrix} 1 & 2 & -1 \\ 0 & -2 & 1 \\ 0 & 0 & -1 \end{pmatrix}$. Alors :

a. Le polynôme caractéristique de $A$ est $P_A(X) = (1-X)^2(-2-X)$

X X b. Le polynôme caractéristique de $A$ est $P_A(X) = (1-X)(-2-X)(-1-X)$

X X c. Une valeur propre de $A$ est $\lambda = -2$

d. Une valeur propre de $A$ est $\lambda = 2$

e. Aucun des autres choix

# QCM 4
# Architecture des ordinateurs

**Lundi 24 novembre 2025**

**Pour toutes les questions, une ou plusieurs réponses sont possibles.**

21. À quoi sert le symbole '#' ?
    - A. Il indique qu'un opérande est sous forme décimale.
    - B. Il indique qu'un opérande est une adresse.
    - X C. Il indique qu'un opérande est une donnée immédiate.
    - D. Il indique qu'un opérande est sous forme hexadécimale.

22. Quelle instruction sert essentiellement à appeler un sous-programme ?
    - X A. JSR
    - B. JMP
    - C. Aucune de ces réponses.
    - D. RTS

23. Les étapes pour dépiler une donnée sont :
    - A. Décrémenter A7 puis lire la mémoire pointée par A7.
    - X B. Lire la mémoire pointée par A7 puis incrémenter A7.
    - C. Décrémenter A7 puis écrire dans la mémoire pointée par A7.
    - D. Écrire dans la mémoire pointée par A7 puis décrémenter A7.

24. Pour empiler une donnée :
    - A. On ne change pas A7.
    - B. Aucune de ces réponses.
    - X C. On incrémente A7 d'abord.
    - X D. On décrémente A7 d'abord.

25. Soient les deux instructions suivantes :

```
CMP.L D1,D2
BLO   NEXT
```

Branchement à `NEXT` si :

- A. D1 = $FF0000FF et D2 = $FF0000FF
- X B. D1 = $00FFFF00 et D2 = $FF0000FF
- C. D1 = $00FFFF00 et D2 = $00FFFF00
- X D. D1 = $FF0000FF et D2 = $00FFFF00

26. Soient les deux instructions suivantes :

```
CMP.L D1,D2
BLT   NEXT
```

Branchement à NEXT si :

- A. D1 = $00FFFF00 et D2 = $FF0000FF
- B. D1 = $FF0000FF et D2 = $FF0000FF
- C. D1 = $FF0000FF et D2 = $00FFFF00
- D. D1 = $00FFFF00 et D2 = $00FFFF00

27. Soient les deux instructions suivantes :

```
CMP.W D1,D2
BLE   NEXT
```

Branchement à NEXT si :

- A. D1 = $00FFFF00 et D2 = $FF0000FF
- B. D1 = $FF0000FF et D2 = $FF0000FF
- C. D1 = $FF0000FF et D2 = $00FFFF00
- D. D1 = $00FFFF00 et D2 = $00FFFF00

28. Soient les deux instructions suivantes :

```
CMP.B D1,D2
BLE   NEXT
```

Branchement à NEXT si :

- A. D1 = $FF0000FF et D2 = $FF0000FF
- B. D1 = $00FFFF00 et D2 = $FF0000FF
- C. D1 = $00FFFF00 et D2 = $00FFFF00
- D. D1 = $FF0000FF et D2 = $00FFFF00

29. Soient les deux instructions suivantes :

```
CMP.B D1,D2
BNE   NEXT
```

Branchement à NEXT si :

- A. D1 = $FF0000FF et D2 = $FF0000FF
- B. D1 = $00FFFF00 et D2 = $FF0000FF
- C. D1 = $00FFFF00 et D2 = $00FFFF00
- D. D1 = $FF0000FF et D2 = $00FFFF00

30. Soient les deux instructions suivantes :

```
TST.W D0
BMI   NEXT
```

L'instruction `BMI` effectue le branchement si :

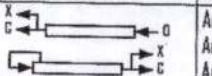
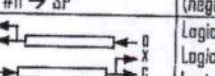
A. D0 = $00000080

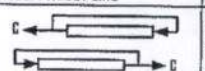
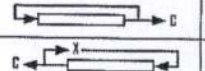
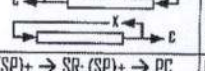
✗ B. D0 = $00008000

C. D0 = $80000000

D. D0 = $00FF00FF

**EASy68K Quick Reference v1.8** http://www.wowgwep.com/EASy68K.htm 

| 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) | (i,An) | (i,An,Rn) | abs.W | abs.L | (i,PC) | (i,PC,Rn) | #n | | |
| ABCD | B | Dy,Dx | *U*U* | e | - | - | - | - | - | - | - | - | - | - | - | $Dy_{10} + Dx_{10} + X \rightarrow Dx_{10}$ | Add BCD source and eXtend bit to |
| | | -(Ay),-(Ax) | | - | - | - | - | e | - | - | - | - | - | - | - | $-(Ay)_{10} + -(Ax)_{10} + X \rightarrow -(Ax)_{10}$ | destination, BCD result |
| ADD [4] | BWL | s,Dn | ***** | e | s | s | s | s | s | s | s | s | s | s | s[4] | s + Dn → Dn | Add binary (ADDI or ADDQ is used when |
| | | Dn,d | | e | d[4] | d | d | d | d | d | d | d | - | - | - | Dn + d → d | 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 → An | Add address (.W sign-extended to .L) |
| ADDI [4] | BWL | #n,d | ***** | d | - | d | d | d | d | d | d | d | - | - | s | #n + d → d | Add immediate to destination |
| ADDQ [4] | BWL | #n,d | ***** | d | d | d | d | d | d | d | d | d | - | - | s | #n + d → d | Add quick immediate (#n range: 1 to 8) |
| ADDX | BWL | Dy,Dx | ***** | e | - | - | - | - | - | - | - | - | - | - | - | Dy + Dx + X → Dx | Add source and eXtend bit to destination |
| | | -(Ay),-(Ax) | | - | - | - | - | e | - | - | - | - | - | - | - | -(Ay) + -(Ax) + X → -(Ax) | |
| AND [4] | BWL | s,Dn | -**00 | e | - | s | s | s | s | s | s | s | s | s | s[4] | s AND Dn → Dn | Logical AND source to destination |
| | | Dn,d | | e | - | d | d | d | d | d | d | d | - | - | - | Dn AND d → d | (ANDI is used when source is #n) |
| ANDI [4] | BWL | #n,d | -**00 | d | - | d | d | d | d | d | d | d | - | - | s | #n AND d → d | Logical AND immediate to destination |
| ANDI [4] | B | #n,CCR | ===== | - | - | - | - | - | - | - | - | - | - | - | s | #n AND CCR → CCR | Logical AND immediate to CCR |
| ANDI [4] | W | #n,SR | ===== | - | - | - | - | - | - | - | - | - | - | - | s | #n AND SR → SR | Logical AND immediate to SR (Privileged) |
| ASL | BWL | Dx,Dy | ***** | e | - | - | - | - | - | - | - | - | - | - | - |  | Arithmetic shift Dy by Dx bits left/right |
| ASR | | #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 ds 1 bit left/right (.W only) |
| Bcc | BW[3] | address[2] | ------ | - | - | - | - | - | - | - | - | - | - | - | - | if cc true then address → PC | Branch conditionally (cc table on back) (8 or 16-bit ± offset to address) |
| BCHG | B L | Dn,d | --*-- | e[1] | - | d | d | d | d | d | d | d | - | - | - | NOT(bit number of d) → Z | Set Z with state of specified bit in d then |
| | | #n,d | | d[1] | - | d | d | d | d | d | d | d | - | - | s | NOT(bit n of d) → bit n of d | invert the bit in d |
| BCLR | B L | Dn,d | --*-- | e[1] | - | d | d | d | d | d | d | d | - | - | - | NOT(bit number of d) → Z | Set Z with state of specified bit in d then |
| | | #n,d | | d[1] | - | d | d | d | d | d | d | d | - | - | s | 0 → bit number of d | clear the bit in d |
| BRA | BW[3] | address[2] | ------ | - | - | - | - | - | - | - | - | - | - | - | - | address → PC | Branch always (8 or 16-bit ± offset to addr) |
| BSET | B L | Dn,d | --*-- | e[1] | - | d | d | d | d | d | d | d | - | - | - | NOT( bit n of d ) → Z | Set Z with state of specified bit in d then |
| | | #n,d | | d[1] | - | d | d | d | d | d | d | d | - | - | s | 1 → bit n of d | set the bit in d |
| BSR | BW[3] | address[2] | ------ | - | - | - | - | - | - | - | - | - | - | - | - | PC → -(SP); address → PC | Branch to subroutine (8 or 16-bit ± offset) |
| BTST | B L | Dn,d | --*-- | e[1] | - | d | d | d | d | d | d | d | d | d | - | NOT( bit Dn of d ) → Z | Set Z with state of specified bit in d |
| | | #n,d | | d[1] | - | d | d | d | d | d | d | d | d | d | s | NOT(bit #n of d ) → Z | 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 → 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 |
| CMPM [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 → Dn if Dn <> -1 then addr →PC } | Test condition, decrement and branch (16-bit ± offset to address) |
| DIVS | W | s,Dn | -***0 | e | - | s | s | s | s | s | s | s | s | s | s | ±32bit Dn / ±16bit s → ±Dn | Dn= [ 16-bit remainder, 16-bit quotient ] |
| DIVU | W | s,Dn | -***0 | e | - | s | s | s | s | s | s | s | s | s | s | 32bit Dn / 16bit s → Dn | Dn= [ 16-bit remainder, 16-bit quotient ] |
| EOR [4] | BWL | Dn,d | -**00 | e | - | d | d | d | d | d | d | d | - | - | s[4] | Dn XOR d → d | Logical exclusive OR Dn to destination |
| EORI [4] | BWL | #n,d | -**00 | d | - | d | d | d | d | d | d | d | - | - | s | #n XOR d → d | Logical exclusive OR #n to destination |
| EORI [4] | B | #n,CCR | ===== | - | - | - | - | - | - | - | - | - | - | - | s | #n XOR CCR → CCR | Logical exclusive OR #n to CCR |
| EORI [4] | W | #n,SR | ===== | - | - | - | - | - | - | - | - | - | - | - | s | #n XOR SR → SR | Logical exclusive OR #n to SR (Privileged) |
| EXG | L | Rx,Ry | ------ | e | e | - | - | - | - | - | - | - | - | - | - | register ←→ register | Exchange registers (32-bit only) |
| EXT | WL | Dn | -**00 | d | - | - | - | - | - | - | - | - | - | - | - | Dn.B → Dn.W \| Dn.W → 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 → PC | Jump to effective address of destination |
| JSR | | d | ------ | - | - | d | - | - | d | d | d | d | d | d | - | PC → -(SP); ↑d → PC | push PC, jump to subroutine at address d |
| LEA | L | s,An | ------ | - | e | s | - | - | s | s | s | s | s | s | - | ↑s → An | Load effective address of s to An |
| LINK | | An,#n | ------ | - | - | - | - | - | - | - | - | - | - | - | - | An → -(SP); SP → An; SP + #n → 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 | | #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 [4] | BWL | s,d | -**00 | e | s[4] | e | e | e | e | e | e | e | s | s | s[4] | s → d | Move data from source to destination |
| MOVE | W | s,CCR | ===== | s | - | s | s | s | s | s | s | s | s | s | s | s → CCR | Move source to Condition Code Register |
| MOVE | W | s,SR | ===== | s | - | s | s | s | s | s | s | s | s | s | s | s → SR | Move source to Status Register (Privileged) |
| MOVE | W | SR,d | ------ | d | - | d | d | d | d | d | d | d | - | - | - | SR → d | Move Status Register to destination |
| MOVE | L | USP,An | ------ | - | d | - | - | - | - | - | - | - | - | - | - | USP → An | Move User Stack Pointer to An (Privileged) |
| | | An,USP | | - | s | - | - | - | - | - | - | - | - | - | - | An → USP | Move An to User Stack Pointer (Privileged) |
| | BWL | s,d | XNZVC | Dn | An | (An) | (An)+ | -(An) | (i,An) | (i,An,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) | (i,An) | (i,An,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 → An | Move source to An (MOVE s,An use MOVEA) |
| MOVEM[4] | WL | Rn-Rn,d | ------ | - | - | d | - | d | d | d | d | d | - | - | - | Registers → d | Move specified registers to/from memory |
| | | s,Rn-Rn | | - | - | s | s | - | s | s | s | s | s | s | - | s → Registers | (.W source is sign-extended to .L for Rn) |
| MOVEP | WL | Dn,(i,An) | ------ | s | - | - | - | - | d | - | - | - | - | - | - | Dn → (i,An)...(i+2,An)...(i+4,A. | Move Dn to/from alternate memory bytes |
| | | (i,An),Dn | | d | - | - | - | - | s | - | - | - | - | - | - | (i,An) → Dn...(i+2,An)...(i+4,A. | (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 | ±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 | 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 - $d_{10}$ - 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 | -**00 | e | - | s | s | s | s | s | s | s | s | s | s[4] | s OR Dn → Dn | Logical OR |
| | | Dn,d | | e | - | d | d | d | d | d | d | d | - | - | - | Dn OR d → d | (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 → -(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 | | #n,Dy | | d | - | - | - | - | - | - | - | - | - | - | s | | Rotate Dy, #n bits left/right (#n: 1 to 8) |
| | W | d | | - | - | d | d | d | d | d | d | d | - | - | - |  | 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 | | #n,Dy | | d | - | - | - | - | - | - | - | - | - | - | s | | Rotate Dy, #n bits left/right (#n: 1 to 8) |
| | W | d | | - | - | d | d | d | d | d | d | d | - | - | - |  | 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 | *U*U* | e | - | - | - | - | - | - | - | - | - | - | - | $Dx_{10}$ - $Dy_{10}$ - X → $Dx_{10}$ | Subtract BCD source and eXtend bit from |
| | | -(Ay),-(Ax) | | - | - | - | - | e | - | - | - | - | - | - | - | $-(Ax)_{10}$ - $-(Ay)_{10}$ - X → $-(Ax)_{10}$ | destination, BCD result |
| Scc | B | d | ------ | d | - | d | d | d | d | d | d | d | - | - | - | If cc is true then 1's → d | If cc true then d.B = 11111111 |
| | | | | | | | | | | | | | | | | else 0's → d | else d.B = 00000000 |
| STOP | | #n | ===== | - | - | - | - | - | - | - | - | - | - | - | s | #n → SR; STOP | Move #n to SR, stop processor (Privileged) |
| SUB[4] | BWL | s,Dn | ***** | e | s | s | s | s | s | s | s | s | s | s | s[4] | Dn - s → Dn | Subtract binary (SUBI or SUBQ used when |
| | | Dn,d | | e | d[4] | d | d | d | d | d | d | d | - | - | - | d - Dn → d | 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 | 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 | ***** | e | - | - | - | - | - | - | - | - | - | - | - | Dx - Dy - X → Dx | Subtract source and eXtend bit from |
| | | -(Ay),-(Ax) | | - | - | - | - | e | - | - | - | - | - | - | - | -(Ax) - -(Ay) - X → -(Ax) | 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); | Push PC and SR, PC set by vector table #n |
| | | | | | | | | | | | | | | | | (vector table entry) → PC | (#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; ᵘ Unsigned, ᵃ Alternate cc) | | | | | |
|---|---|---|---|---|---|
| cc | Condition | Test | cc | Condition | Test |
| T | true | 1 | VC | overflow clear | !V |
| F | false | 0 | VS | overflow set | V |
| HIᵘ | higher than | !(C + Z) | PL | plus | !N |
| LSᵘ | lower or same | C + Z | MI | minus | N |
| HSᵘ, CCᵃ | higher or same | !C | GE | greater or equal | !(N ⊕ V) |
| LOᵘ, CSᵃ | lower than | C | LT | less than | (N ⊕ V) |
| NE | not equal | !Z | 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

QCM 2 Azar Chap20 condits3 ex 16 (fall23late bus no.6) Nov 25

Choose the one correct answer for each sentence. One answer only unless otherwise indicated.

31. The bus you take to school was late yesterday. You say:

a. If the bus had arrive on time, I would not have been late for class.

b. If the bus had arrived on time, I would not have been late for class.

c. If the bus arrived on time, I would not have been late for class.

d. If the bus arrived on time, I would not be late for class.

32. The sentence, "If Ann had practiced more, she would have given a better presentation," refers to:

a. the future.

b. the past.

c. the present and the future.

d. the present.

33. You say: "If Rashid had written the paper by himself, there would have been many mistakes." In truth, this means:

a. There were many mistakes.

b. Rashid wrote the paper by himself.

c. Rashid did not write the paper by himself.

d. Rashid is angry about the paper.

34. If you say: "If the president had any self-respect, she would resign," this refers to:

a. The future.

b. The past.

c. The present.

35. Teacher: If Lola ___ to class late yesterday, I ____ let her in.

a. came / would not have

b. had come / will not

c. had come / would not have

d. didn't come / would not

36. Zakir tried to send the boss a warning by email last night, but he didn't have enough time. In other words:

a. If he had had enough time, he would have sent her a warning.

b. If he had enough time, he would have sent her a warning.

b. If he hadn't enough time, he would have sent her a warning.

d. If he had had enough time, he would send her a warning.

37. Choose the **one** correct sentence.

a. If I had been born rich, I will not study.

b. If I had been born rich, I was happy.

c. If I had been born rich, I will be happy.

d. If I had been born rich, I would not have gone to school.

38. ____ all the spectators arrived on time, the match took place as planned.

a. When

b. Because

c. Even though

d. Whether

39. Coco wants to change heaters because the one she has now is old. Which sentence matches?

a. If her heater were newer, she would not think about changing it.

b. If her heater had been newer, she would not think about changing it.

c. If her heater would be newer, she would keep it.

d. If her heater were newer, she keeps it.

40. Which TWO sentences are perfect?

a. Jo would have bought the stock only if interest rates had gone down.

b. Jo will buy the stock only if interest rates goes down.

c. Jo will buy the stock only if interest rates go down.

d. Jo will have bought the stock only if interest rates go down.