Arbres et Recherche II
QCM 4
6 octobre 2025

1. **Il y a collision primaire entre x et y lorsque (avec h fonction de hachage)**
   - (a) $x = y$ et $h(x) = h(y)$
   - (b) $x \neq y$ et $h(x) = h(y)$
   - (c) $x = y$ et $h(x) \neq h(y)$
   - (d) $x \neq y$ et $h(x) \neq h(y)$

2. **La gestion des collisions primaires peut se gérer par**
   - (a) une fonction de hachage
   - (b) calcul
   - (c) chaînage
   - (d) aléatoirement
   - (e) universellement

3. **Avec les méthodes de résolution des collisions directes :**
   - (a) il est possible d'utiliser une fonction de hachage universelle
   - (b) les éléments en collisions sont ignorés
   - (c) les éléments en collisions sont chaînés entre eux
   - (d) les collisions sont gérées par une fonction d'essais successifs
   - (e) il n'est pas nécessaire d'avoir un tableau de hachage

4. **Quelles méthodes de gestion des collision opèrent par calcul ?**
   - (a) Le hachage à double chaînage
   - (b) Le hachage avec chaînage séparé
   - (c) Le hachage linéaire
   - (d) Le hachage coalescent
   - (e) Le double hachage

5. **Une fonction d'essais successifs est utilisée dans**
   - (a) les méthodes directes
   - (b) le hachage linéaire
   - (c) le hachage avec chaînage séparé
   - (d) le hachage coalescent

6. **Le hachage linéaire**
   - (a) génère moins de collisions que les méthodes indirectes
   - (b) est extrêmement simple à mettre en œuvre
   - (c) est plus performant que le hachage avec chaînage séparé
   - (d) est moins rapide que d'autres méthodes
   - (e) est bien adapté aux ensembles dynamiques

Pour les questions suivantes, considérons le tableau des valeurs de hachage suivant :

éléments	x_1	x_2	x_3	x_4	x_5	x_6
valeurs de hachage	3	1	3	1	6	3

Les éléments x_i sont insérés, dans l'ordre du tableau, à une table de hachage de taille $m = 7$ (indexée de 1 à 7).

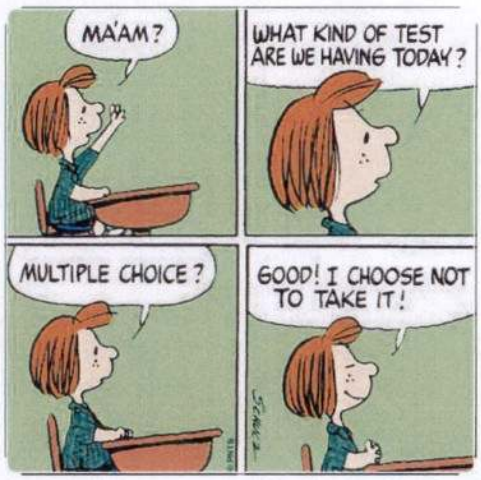
7. **Dans le cas du hachage linéaire, quelle sera la place de l'élément x_4 dans la table de hachage ?**
   (a) 1
   (b) 2
   (c) 3
   (d) 4
   (e) 5

8. **Dans le cas du hachage linéaire, quelle sera la place de l'élément x_6 dans la table de hachage ?**
   (a) 2
   (b) 3
   (c) 4
   (d) 5
   (e) 6

La fonction d'essais du hachage linéaire classique est remplacée par $essai_i(x) = h(x) \oplus k \times (i-1)$.
k est appelé *coefficient de décalage*.

9. **Dans le cas du hachage linéaire avec *coefficient de décalage* $k = 3$, quelle sera la place de l'élément x_4 dans la table de hachage ?**
   (a) 1
   (b) 2
   (c) 3
   (d) 4
   (e) 5

10. **Dans le cas du hachage linéaire avec *coefficient de décalage* $k = 3$, quelle sera la place de l'élément x_6 dans la table de hachage ?**
    (a) 2
    (b) 3
    (c) 4
    (d) 5
    (e) 6



NE PAS COCHER LES CASES 11 à 20 !

QCM 2
Architecture des ordinateurs

Lundi 6 octobre 2025

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

21. Le *flag* V est positionné à 0 quand :
    - A. Un résultat est négatif.
    - B. Aucune de ces réponses.
    - C. Aucun dépassement signé n'apparaît.
    - D. Un dépassement non signé apparaît.

22. Le registre PC
    - A. Aucune de ces réponses.
    - B. Contient l'adresse de la prochaine instruction à exécuter.
    - C. Contient l'adresse de la pile.
    - D. Contient l'état du microprocesseur.

23. Le 68000 possède :
    - A. Deux pointeurs de pile : USP et SSP
    - B. Aucune de ces réponses.
    - C. Deux pointeurs de pile : CCR et SR
    - D. Un pointeur de pile : PC

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

25. Quelles sont les deux instructions de branchements inconditionels ?
    - A. BRA et BEQ
    - B. BRA et JMP
    - C. BEQ et BNE
    - D. BSR et BNE

26. L'instruction BNE effectue un branchement si :
    - A. N = 1
    - B. Z = 1
    - C. V = 1
    - D. Aucune de ces réponses.

27. Le mode d'adressage direct :
    A. Spécifie un emplacement mémoire.
    B. Ne spécifie pas d'emplacement mémoire.
    C. Est uniquement sur 8 bits.
    D. Aucune de ces réponses.

28. Soit l'instruction suivante : `MOVE.L -(A0),D0`
    A. A0 est décrémenté de 1.
    B. A0 est décrémenté de 2.
    C. A0 est décrémenté de 4.
    D. A0 ne change pas.

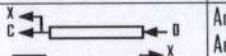
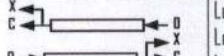
29. Soit l'instruction suivante : `MOVE.L -4(A0),D0`
    A. A0 est décrémenté de 1.
    B. A0 est décrémenté de 2.
    C. A0 est décrémenté de 4.
    D. A0 ne change pas.

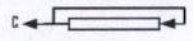
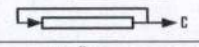
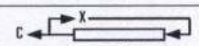
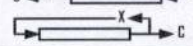
30. Soient les deux instructions suivantes :

```
TST.B D0
BMI   NEXT
```

    L'instruction `BMI` effectue le branchement si :
    A. D0 = $00000000
    B. D0 = $00000050
    C. D0 = $0000007F
    D. D0 = $000000FF

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,addres[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 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	*****	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); (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; 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

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

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

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

QCM 2 Azar new Chap13 (Adjec ob-verb ob-prep pp. 276–278ex8-14) B5 Sept 25

In 31 - 34, the two sentences have been combined for you, with the second sentence as an adjective clause. Choose the correct logical combination(s). More than one answer is possible for 31-34.

31. The book got excellent reviews. I wrote it last summer.

a. The book I wrote last summer got excellent reviews.

b. The book what I wrote last summer got excellent reviews.

c. The book which I wrote last summer got excellent reviews.

d. The book who I wrote last summer got excellent reviews.

32. Sue cannot visualize the man. She met him at a conference just last week.

a. Sue cannot visualize the man which she met at the conference just last week.

b. Sue cannot visualize the man she met at the conference just last week.

c. Sue cannot visualize the man that she met at the conference just last week.

d. Sue cannot visualize the man meet at the conference just last week.

33. I developed the game. My ex-teacher plays it all the time.

a. I developed the game whom my ex-teacher plays all the time.

b. I developed the game that my ex-teacher plays all the time.

c. I developed the game my ex-teacher plays all the time.

d. I developed the game that my ex-teacher plays it all the time.

34. The dogs were very obedient. We trained them yesterday.

a. The dogs whom we trained yesterday were very obedient.

b. We trained the dogs whom were very obedient yesterday.

c. The dogs which we trained yesterday were very obedient.

d. The dogs we trained yesterday were very obedient.

Choose the one adjective clause that is **NOT** correct for sentences 35 and 36.

35. The pizza ___ arrived two hours late.

a. that I was waiting for

b. I was waiting for

c. which I was waiting for

d. who I was waiting for

36. The manager ___ at the store was able to reimburse me.

a. to who I spoke to

b. who I spoke to

c. to whom I spoke

d. I spoke to

Identify the **one** adjective clause in these sentences.

37. John found my passport, which I had lost in the garden.

a. John found my passport

b. in the garden

c. which I had lost

d. in the garden

38. Last night at the football game, I ran into a woman my brother had shared a room with at college.

a. at the football game

b. I ran into a woman

c. my brother had shared a room with

d. I ran into a woman my brother had shared a room

39. Ralph explained in detail a play that he was writing.

a. in detail

b. Ralph explained in detail a play

c. a play

d. that he was writing

40. My dad knows a tax auditor who is accused of tax evasion.

a. a tax auditor who is accused

b. My dad knows

c. who is accused of tax evasion

d. My dad knows a tax auditor

ETHIQUE

Sélectionner la bonne réponse.

41) On appelle « contenu normatif » :
- a) un ensemble de pratiques induites dans un contexte ou un outil technique donné
- b) les lois qui encadrent la démocratisation d'un outil numérique
- c) la bonne conduite à adopter dans l'utilisation d'un outil technique
- d) le résultat d'un algorithme d'apprentissage

42) Quelle est l'origine commune de la morale et de l'éthique ?
- a) les mœurs
- b) l'éducation
- c) les rituels
- d) les sacrifices

43) Quand on parle d'éthique, à quel type de situations s'intéresse-t-on ?
- a) Des situations strictement juridiques
- b) Des situations où un choix est à faire
- c) Des situations où une seule réponse est possible
- d) Des situations purement techniques

44) Qu'est-ce qui caractérise une situation à composante morale ?
- a) Elle oppose toujours la loi et l'illégalité
- b) Elle ne mobilise aucun principe de valeur
- c) Elle interroge le bien-fondé de notre comportement vis-à-vis d'autrui
- d) Elle est obligatoirement religieuse

45) Quelle est la différence d'origine entre "morale" et "éthique" ?
- a) La morale vient du grec, l'éthique du latin
- b) La morale est rationnelle, l'éthique est religieuse
- c) La morale vient du latin, l'éthique du grec
- d) Il n'y a aucune différence d'origine

46) Pourquoi évite-t-on souvent la phase de questionnement dans les dilemmes moraux ?
- a) Parce qu'elle est inutile
- b) Parce qu'elle ne concerne que soi et non autrui
- c) Parce qu'elle demande des efforts et peut aller à l'encontre de ce qui "se fait"
- d) Parce qu'elle est interdite par la loi

47) Quel exemple montre que la technique n'est pas neutre dans le domaine du numérique ?
- a) L'usage de la monnaie dans les transports publics
- b) Le téléchargement illégal, facile mais considéré comme du vol
- c) Le prêt de livres à la bibliothèque

d) La diffusion de journaux papier

48) Parmi ces propositions, laquelle est la meilleure définition de la morale ?

a) La distinction entre ce qui est légal et illégal

b) Le caractère de ce qui est moral, lié au bien et au mal

c) L'étude rationnelle des comportements sociaux

d) Une règle politique imposée par l'État

49) Comment se définit la morale selon ses grandes caractéristiques ?

a) Une conception éternelle du bien et du mal

b) Une réflexion toujours adaptée aux réalités concrètes et changeantes

c) Un ensemble de pratiques sociales sans portée universelle

d) Une méthode scientifique pour analyser le droit

50) Quelle limite majeure de la morale rend son application difficile dans des situations concrètes comme le numérique ?

a) Elle est trop subjective et changeante selon chaque individu

b) Elle repose sur des principes déconnectés de la réalité concrète des hommes

c) Elle interdit toute réflexion personnelle

d) Elle se limite uniquement aux situations illégales