

Partiel 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 = \$FFFFFFE2 A0 = \$00005000 PC = \$00006000
 D1 = \$00000070 A1 = \$00005008
 D2 = \$FFFF006A 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  #$48f5,d7
next1     moveq.l #1,d1
          cmpi.b  #1,d7
          blt   next2
          moveq.l #2,d1
next2     clr.l   d2
          move.l #$44444444,d0
loop2    addq.l  #1,d2
          sub.w  #2,d0
          bne   loop2
next3     clr.l   d3
          move.b #$54,d0
loop3    addq.l  #1,d3
          dbra  d0,loop3      ; DBRA = DBF
next4     move.l  #$1234,d4
          rol.w  #4,d4
          ror.l  #8,d4
          rol.b  #4,d4

```

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 0).

Attention ! Tous les sous-programmes sont limités à 10 lignes d'instructions au maximum.

1. Réalisez le sous-programme **next_42** qui renvoie l'adresse où se trouve la prochaine occurrence « 42 » dans une chaîne de caractères.

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

Sortie : **A0.L** pointe sur la prochaine occurrence « 42 » dans la chaîne de caractères

(il pointe sur le caractère « 4 »). Si aucune occurrence n'est trouvée, il contient la valeur 0.

2. À l'aide du sous-programme **next_42**, réalisez le sous-programme **replace_42_by_char** qui remplace toutes les occurrences « 42 » d'une chaîne de caractères par un nouveau nombre à deux chiffres. Le nouveau nombre est passé en paramètre sous la forme de codes ASCII. La chaîne est modifiée directement en mémoire.

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

D1.B contient le code ASCII du chiffre des unités du nouveau nombre.

D2.B contient le code ASCII du chiffre des dizaines du nouveau nombre.

3. À l'aide du sous-programme **replace_42_by_char**, réalisez le sous-programme **replace_42_by_int** qui remplace toutes les occurrences « 42 » d'une chaîne de caractères par un nouveau nombre à deux chiffres. Le nouveau nombre est passé en paramètre sous la forme d'un entier. La chaîne est modifiée directement en mémoire. Pour rappel, le code ASCII du caractère « 0 » est égal à \$30.

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

D0.L contient le nouveau nombre (nombre entier compris entre 0 et 99).

Par exemple :

```

Main          lea.l   String1,a0
              move.b  #'7',d2
              move.b  #'5',d1
              jsr     replace_42_by_char

              lea.l   String2,a0
              move.l  #75,d0
              jsr     replace_42_by_int

              illegal

String1       dc.b    "Two occurrences: 42 and 42",0
String2       dc.b    "Two occurrences: 42 and 42",0

```

Après l'exécution de ce programme, les deux chaînes (*String1* et *String2*) contiendront :
"Two occurrences: 75 and 75"

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 -(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	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	$s + An \rightarrow An$	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: l to B)
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	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: l to B)
Bcc	BW ²	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	-	-	-	s	$\text{NOT}(\text{bit number of } d) \rightarrow Z$ $\text{NOT}(\text{bit } n \text{ of } d) \rightarrow \text{bit } n \text{ of } d$	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	-	-	-	s	$\text{NOT}(\text{bit number of } d) \rightarrow Z$ $0 \rightarrow \text{bit number of } d$	Set Z with state of specified bit in d then clear the bit in d
BRA	BW ²	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	-	-	-	s	$\text{NOT}(\text{bit } n \text{ of } d) \rightarrow Z$ $1 \rightarrow \text{bit } n \text{ of } d$	Set Z with state of specified bit in d then set the bit in d
BSR	BW ²	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	-	-	-	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	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	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	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 \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	s	$\pm 32\text{bit } Dn / \pm 16\text{bit } s \rightarrow \pm Dn$	Dn = [16-bit remainder, 16-bit quotient]
DIVU	W	s,Dn	---*0	e	-	s	s	s	s	s	s	s	s	s	s	s	$32\text{bit } Dn / 16\text{bit } s \rightarrow Dn$	Dn = [16-bit remainder, 16-bit quotient]
EOR ⁴	BWL	Dn,d	---*00	e	-	d	d	d	d	d	d	d	-	-	-	s ⁴	$Dn \text{ XOR } d \rightarrow d$	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	d	$\uparrow d \rightarrow \text{PC}$	Jump to effective address of destination
JSR		d	-----	-	-	d	-	-	-	d	d	d	d	d	d	d	PC \rightarrow -(SP); $\uparrow d \rightarrow \text{PC}$	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; SP + #n \rightarrow SP	Create local workspace on stack (negative n to allocate space)
LSL	BWL	Dx,Dy #n,Dy	***0*	e	-	-	-	-	-	-	-	-	-	-	-	-		Logical shift Dy, Dx bits left/right
LSR	W	d	-----	d	-	d	d	d	d	d	d	d	-	-	-	s		Logical shift Dy, #n bits L/R (#n: l to B)
MOVE ⁴	BWL	s,d	---*00	e	s ⁴	e	e	e	e	e	e	e	s	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	$s \rightarrow \text{CCR}$	Move source to Condition Code Register
MOVE	W	s,SR	=====	s	-	s	s	s	s	s	s	s	s	s	s	s	$s \rightarrow \text{SR}$	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 An,USP	-----	-	d	-	-	-	-	-	-	-	-	-	-	-	USP \rightarrow An An \rightarrow USP	Move User Stack Pointer to An (Privileged) 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)	(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 ⁴	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,(i,An) (i,An),Dn	-----	s	-	-	-	-	d	-	-	-	-	-	-	-	Dn → (i,An)...(i+2,An)...(i+4,An) (i,An) → Dn...(i+2,An)...(i+4,An)	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 unsig'd 16-bit; result: unsig'd 32-bit
NBCD	B	d	*U*U*	d	-	d	d	d	d	d	d	d	-	-	-	-	0 - d ₀ - 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*	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	=====	-	-	-	-	-	-	-	-	-	-	-	-	-	#n OR CCR → CCR	Logical OR #n to CCR
ORI ⁴	W	#n,SR	=====	-	-	-	-	-	-	-	-	-	-	-	-	-	#n OR SR → SR	Logical OR #n to SR (Privileged)
PEA	L	s	-----	-	-	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	-	-	-	-	-	-	-	-	-	-	-	-		
ROR	W	#n,Dy		d	-	-	-	-	-	-	-	-	-	-	-	-		
ROXL	BWL	Dx,Dy	***0*	e	-	-	-	-	-	-	-	-	-	-	-	-		
ROXR	W	#n,Dy		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	-	-	-	-	d - #n → d	Subtract immediate from destination
SUBQ ⁴	BWL	#n,d	*****	d	d	d	d	d	d	d	d	d	-	-	-	-	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	-----	-	-	-	-	-	-	-	-	-	-	-	-	-	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	1	VC	overflow clear	IV
F	false	0	VS	overflow set	V
HP [°]	higher than	!(C + Z)	PL	plus	IN
LS [°]	lower or same	C + Z	MI	minus	N
HS [°] , CC [°]	higher or same	!C	GE	greater or equal	!(N ⊕ V)
LD [°] , CS [°]	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** -28 to +27 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 00 40 E7 21 48 C0	A0 = \$00005004 A1 = \$0000500C
Exemple	\$005008 C9 10 11 C8 D4 36 FF 88	Aucun changement
MOVE.W -(A2), -(A2)		
MOVE.L #510, 40(A0, D0.L)		
MOVE.W 4(A1), (A1)		
MOVE.B 7(A2), -\$6F(A2, D2.W)		

Exercice 2

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

Exercice 3

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

Exercice 4

next_42

replace_42_by_char

replace_42_by_int