

Architecture des ordinateurs

Partiel 2 – Mai 2009

Durée 1h30

Nom :

Classe :

Exercice 1. (2 points)

Codez les instructions suivantes en langage machine 68000, vous développerez les différents champs puis vous exprimerez le résultat final sous forme hexadécimale en précisant la taille des mots supplémentaires lorsque le cas se présente.

1. MOVE.L #\$FFFF, (A2)
2. MOVE.W (A4)+, 32(A3)

Exercice 2. (2 points)

Vous indiquerez après chaque instruction, le nouveau contenu des registres (sauf le PC) et/ou de la mémoire qui viennent d'être modifiés. Vous utiliserez la représentation hexadécimale.

Attention : La mémoire et les registres sont réinitialisés à chaque nouvelle instruction.

Valeurs initiales : D0 = \$00000004 A0 = \$00005000 PC = \$00006000
 D1 = \$0000FFFF A1 = \$00005008
 D2 = \$7FFFFAAA A2 = \$00005010

\$005000	55	65	E4	7F	B4	16	22	2F
\$005008	52	9F	28	A7	CF	A5	41	AA
\$005010	DC	54	0F	22	DB	BA	1D	D9

1. MOVE.W #77, -(A2)
2. MOVE.B \$5012(PC), -2(A1, D1.W)

Exercice 3. (2 Points)

Donner le résultat des additions hexadécimales suivantes, ainsi que le contenu des bits N, Z, V et C du registre d'état.

1. \$62 + \$11 opération en .B
2. \$FFFE + \$FFF1 opération en .W

Exercice 4. (9 points)

On désire réaliser un sous-programme **toupper** qui convertit les minuscules en majuscules dans une chaîne de caractères (terminée par un 0). Les autres caractères ne subiront aucune modification.

1) Réalisez dans un 1er temps le sous-programme **islower** permettant de déterminer si un caractère est une minuscule. Seule la valeur de D0 devra être modifiée en sortie du sous-programme.

Entrée : D1.B contient le caractère ASCII à tester.

Sortie : D0.L contient 1 si le caractère est une minuscule [a...z]

D0.L contient 0 si le caractère n'est pas une minuscule

2) Sachant que le code ASCII de 'A' vaut \$41 et que celui de 'a' vaut \$61, quelle opération doit-on réaliser sur un caractère pour le transformer en majuscule.

3) En vous aidant du sous-programme **islower**, réalisez **toupper**. Aucun registre ne sera modifié en sortie du sous-programme.

Entrée : A0.L pointe sur le début de la chaîne à convertir

Exemple d'appel à **toupper** :

```
lea chaine,a0
jsr toupper
illegal
```

```
chaine    dc.b "Voici uNe Chaîne, a Convertir.",0
```

Exercice 5. (2 points)

Soit un mot de 32 bits X3 X2 X1 X0 (Xn représentant un octet) contenu dans D0. Ecrivez un programme en assembleur 68000, qui a pour résultat les mots suivants :

D1 = X3 X1 X2 X0

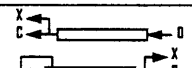
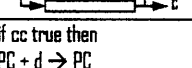
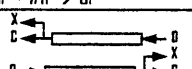
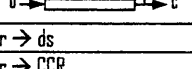
D2 = X0 X2 X1 X3

Exercice 6. Le PIC 12F508 (3 pts)

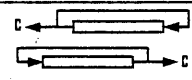
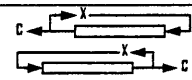
En utilisant l'extrait du DataSheet fourni en annexe, déterminer, pour le PIC12F508 :

1. La taille d'une donnée
2. Le nombre et la taille des mots programme
3. Le nombre et le(s) type(s) de mémoire(s). Préciser dans chacun des cas la particularité (volatile ou non), la taille de la mémoire et les informations stockées dans cette mémoire.

68000 Quick Reference

Opcode	Size	Operand	CCR	Effective Address												Operation	Description
	BWL	sr,ds	XNZVC	Dn	An	(An)	(An)+	-(An)	(d,An)	(d,An,Rn)	abs.W	abs.L	(d,PC)	(d,PC,Rn)	#n		
ABCD	B	Dy,Dx -(Ay),-(Ax)	*U*U*	-	-	-	-	-	-	-	-	-	-	-	-	$Dy_{10} + Dx_{10} + X \rightarrow Dx_{10}$ $-(Ay)_{10} + -(Ax)_{10} + X \rightarrow -(Ax)_{10}$	Add BCD with Extend
ADD	BWL	sr,Dn Dn,ds	*****	sr ds	sr -	sr ds	sr ds	sr ds	sr ds	sr ds	sr ds	sr ds	sr -	sr -	sr -	$sr + Dn \rightarrow Dn$ $Dn + ds \rightarrow ds$	Add binary (ADDI or ADDQ is used when source is #n)
ADDA ²	WL	sr,An	-----	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	$sr + An \rightarrow An$	Add address (.W sign-extended to .L)
ADDI ²	BWL	#n,ds	*****	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	$\#n + ds \rightarrow ds$	Add immediate
ADDQ ²	BWL	#n,ds	*****	ds	ds	ds	ds	ds	ds	ds	ds	ds	-	-	-	$\#n + ds \rightarrow ds$	Add quick immediate (#n range: 1 to 8)
ADDX	BWL	Dy,Dx -(Ay),-(Ax)	*****	-	-	-	-	-	-	-	-	-	-	-	-	$Dy + Dx + X \rightarrow Dx$ $-(Ay) + -(Ax) + X \rightarrow -(Ax)$	Add with Extend
AND	BWL	sr,Dn Dn,ds	---*00	sr ds	-	sr ds	sr ds	sr ds	sr ds	sr ds	sr ds	sr ds	sr -	sr -	sr -	$sr \& Dn \rightarrow Dn$ $Dn \& ds \rightarrow ds$	Logical AND (ANDI is used when source is #n)
ANDI ²	BWL	#n,ds	---*00	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	$\#n \& ds \rightarrow ds$	Logical AND immediate
ANDI ²	B	#n,CCR	*****	-	-	-	-	-	-	-	-	-	-	-	-	$\#n \& CCR \rightarrow CCR$	Logical AND immediate to CCR
ANDI ²	W	#n,SR	*****	-	-	-	-	-	-	-	-	-	-	-	-	$\#n \& SR \rightarrow SR$	Logical AND immediate to SR (Privileged)
ASL	BWL	Dx,Dy #n,Dy	*****	-	-	-	-	-	-	-	-	-	-	-	-		Arithmetic shift Dy Dx bits left/right
ASR	W	ds	-	-	ds	ds	ds	ds	ds	ds	ds	ds	-	-	-		Arithmetic shift ds 1 bit left/right (.W only)
Bcc	BW ³	label	-----	-	-	-	-	-	-	-	-	-	-	-	-	if cc true then $PC + d \rightarrow PC$	Branch conditionally (cc: See Table next pg) (d: 8/16-bit signed integer)
BCHG	B L	Dn,ds #n,ds	---*--	ds ds	-	ds ds	ds ds	ds ds	ds ds	ds ds	ds ds	ds ds	-	-	-	$NOT(bit\ number\ of\ ds) \rightarrow Z$ $NOT(bit\ n\ of\ ds) \rightarrow bit\ n\ of\ ds$	Set Z with state of specified bit in ds then invert the bit in ds
BCLR	B L	Dn,ds #n,ds	---*--	ds ds	-	ds ds	ds ds	ds ds	ds ds	ds ds	ds ds	ds ds	-	-	-	$NOT(bit\ number\ of\ ds) \rightarrow Z$ $0 \rightarrow bit\ number\ of\ ds$	Set Z with state of specified bit in ds then clear the bit in ds
BRA	BW ³	label	-----	-	-	-	-	-	-	-	-	-	-	-	-	$PC + d \rightarrow PC$	Branch always (d: 8/16-bit signed integer)
BSET	B L	Dn,ds #n,ds	---*--	ds ds	-	ds ds	ds ds	ds ds	ds ds	ds ds	ds ds	ds ds	-	-	-	$NOT(bit\ n\ of\ ds) \rightarrow Z$ $1 \rightarrow bit\ n\ of\ ds$	Set Z with state of specified bit in ds then set the bit in ds
BSR	BW ³	label	-----	-	-	-	-	-	-	-	-	-	-	-	-	$PC \rightarrow -(SP); PC + d \rightarrow PC$	Branch to subroutine (d: 8/16-bit sign-int)
BTST	B L	Dn,ds #n,ds	---*--	ds ds	-	ds ds	ds ds	ds ds	ds ds	ds ds	ds ds	ds ds	-	-	-	$NOT(bit\ Dn\ of\ ds) \rightarrow Z$ $NOT(bit\ \#n\ of\ ds) \rightarrow Z$	Set Z with state of specified bit in ds Leave the bit in ds unchanged
CHK	W	sr,Dn	-*UUU	sr	-	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	if $Dn < 0$ or $Dn > sr$ then TRAP	Compare Dn with 0 and upper bound [sr]
CLR	BWL	ds	-0100	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	$0 \rightarrow ds$	Clear destination to zero
CMP	BWL	sr,Dn	-----	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	set CCR with $Dn - sr$	Compare Dn to source
CMPP ²	WL	sr,An	-----	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	set CCR with $An - sr$	Compare An to source
CMPI ²	BWL	#n,ds	-----	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	set CCR with $ds - \#n$	Compare destination to #n
CMPPM ²	BWL	(Ay)+,(Ax)+	-----	-	-	-	ea	-	-	-	-	-	-	-	-	set CCR with $(Ax) - (Ay)$	Compare (Ax) to (Ay); Increment Ax & Ay
DBcc	W	Dn,label	-----	-	-	-	-	-	-	-	-	-	-	-	-	if cc false then $\{ Dn-1 \rightarrow Dn$ if $Dn < -1$ then $PC+d \rightarrow PC$ }	Test condition, decrement & branch (d: 16-bit signed integer)
DIVS	W	sr,Dn	-----	sr	-	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	$\pm 32bit\ Dn / \pm 16bit\ sr \rightarrow \pm Dn$	$Dn = [16\text{-bit remainder}, 16\text{-bit quotient}]$
DIVU	W	sr,Dn	-----	sr	-	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	$32bit\ Dn / 16bit\ sr \rightarrow Dn$	$Dn = [16\text{-bit remainder}, 16\text{-bit quotient}]$
EOR	BWL	Dn,ds	---*00	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	$Dn\ XOR\ ds \rightarrow ds$	Logical exclusive OR Dn to ds
EORI ²	BWL	#n,ds	---*00	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	$\#n\ XOR\ ds \rightarrow ds$	Logical exclusive OR #n to ds
EORI ²	B	#n,CCR	*****	-	-	-	-	-	-	-	-	-	-	-	-	$\#n\ XOR\ CCR \rightarrow CCR$	Logical exclusive OR #n to CCR
EORI ²	W	#n,SR	*****	-	-	-	-	-	-	-	-	-	-	-	-	$\#n\ XOR\ SR \rightarrow SR$	Logical exclusive OR #n to SR (Privileged)
EXG	L	Rx,Ry	-----	ea	ea	-	-	-	-	-	-	-	-	-	-	register \leftrightarrow register	Exchange registers (32-bit only)
EXT	WL	Dn	---*00	-	-	-	-	-	-	-	-	-	-	-	-	$Dn.B \rightarrow Dn.W \mid Dn.W \rightarrow Dn.L$	Sign extend (change .B to .W or .W to .L)
ILLEGAL			-----	-	-	-	-	-	-	-	-	-	-	-	-	$PC \rightarrow -(SP); SR \rightarrow -(SP)$	Generate Illegal Instruction exception
JMP		ds	-----	-	-	ds	-	-	ds	ds	ds	ds	ds	ds	ds	$ds \rightarrow PC$	Jump to address specified by ds
JSR		ds	-----	-	-	ds	-	-	ds	ds	ds	ds	ds	ds	ds	$PC \rightarrow -(SP); ds \rightarrow PC$	push PC, jump to subroutine at address ds
LEA	L	sr,An	-----	-	-	sr	-	-	sr	sr	sr	sr	sr	sr	sr	$sr \rightarrow An$	Load effective address of sr to An
LINK		An,#n	-----	-	-	-	-	-	-	-	-	-	-	-	-	$An \rightarrow -(SP); SP \rightarrow An$ $SP + \#n \rightarrow SP$	Create local workspace on stack (n must be negative to allocate!)
LSL	BWL	Dx,Dy	***0*	-	-	-	-	-	-	-	-	-	-	-	-		Logical shift Dy, Dx bits left/right
LSR	W	#n,Dy ds	-	-	ds	ds	ds	ds	ds	ds	ds	ds	-	-	-		Logical shift Dy, #n bits L/R (#n: 1 to 8) Logical shift ds 1 bit left/right (.W only)
MOVE	BWL	ea,ea	---*00	ea	sr	ea	ea	ea	ea	ea	ea	ea	sr	sr	sr	$sr \rightarrow ds$	Move data from source to destination
MOVE	W	sr,CCR	*****	sr	-	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	$sr \rightarrow CCR$	Move source to Condition Code Register
MOVE	W	sr,SR	*****	sr	-	sr	sr	sr	sr	sr	sr	sr	sr	sr	sr	$sr \rightarrow SR$	Move source to Status Register (Privileged)
MOVE	W	SR,ds	-----	ds	-	ds	ds	ds	ds	ds	ds	ds	-	-	-	$SR \rightarrow ds$	Move Status Register to destination
MOVE	L	USP,An	-----	-	-	-	-	-	-	-	-	-	-	-	-	$USP \rightarrow An$	Move User Stack Pointer to An (Privileged)
		An,USP	-----	-	-	-	-	-	-	-	-	-	-	-	-	$An \rightarrow USP$	Move An to User Stack Pointer (Privileged)

E.P.I.T.A. - Architecture des ordinateurs - Info-Spé 2008/2009

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

Condition Tests (& logical AND, + logical OR, ! logical NOT, * Unsigned)					
cc	Condition	Test	cc	Condition	Test
T	true	I	VC	overflow clear	IV
F	false	O	VS	overflow set	V
HI ^u	high	IC & IZ	PL	plus	IN
LS ^u	low or same	C + Z	MI	minus	N
CC, HS ^u	carry clear	IC	GE	greater or equal	N & V + IN & IV
CS, LO ^u	carry set	C	LT	less than	N & IV + IN & V
NE	not equal	IZ	GT	greater than	N&V& IZ + IN& IV& IZ
EQ	equal	Z	LE	less or equal	Z + N & IV + IN & V

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
PC Program Counter (24-bit)
sr Source ds Destination
#n Immediate data d Displacement
ea Effective Address (source or destination)
BCD Binary Coded Decimal

SSP Supervisor Stack Pointer (32-bit)
USP User Stack Pointer (32-bit)
SP Active Stack Pointer (same as A7)
label Destination of Branch (Assembler calculates displacement value)
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 result of operation
- not affected, O cleared, I set, U undefined

- Long only; all others are byte only
- Assembler selects appropriate opcode
- Branch sizes: B or S -128 to +127 bytes, W or L -32768 to +32767 bytes

Integer Instructions

LSL, LSRLogical Shift
(M68000 Family)**LSL, LSR****Instruction Format:**

MEMORY SHIFTS

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	0	1	dr	1	1	EFFECTIVE ADDRESS					
										MODE		REGISTER			

Instruction Fields:

dr field—Specifies the direction of the shift.

0 — Shift right

1 — Shift left

Effective Address field—Specifies the operand to be shifted. Only memory alterable addressing modes can be used as listed in the following tables:

Addressing Mode	Mode	Register
Dn	—	—
An	—	—
(An)	010	reg. number:An
(An) +	011	reg. number:An
— (An)	100	reg. number:An
(d ₁₆ ,An)	101	reg. number:An
(d ₈ ,An,Xn)	110	reg. number:An

Addressing Mode	Mode	Register
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d ₁₆ ,PC)	—	—
(d ₈ ,PC,Xn)	—	—

MC68020, MC68030, and MC68040 only

(bd,An,Xn)*	110	reg. number:An
([bd,An,Xn],od)	110	reg. number:An
([bd,An],Xn,od)	110	reg. number:An

(bd,PC,Xn)*	—	—
([bd,PC,Xn],od)	—	—
([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

Integer Instructions

MOVEMove Data from Source to Destination
(M68000 Family)**MOVE****Operation:** Source → Destination**Assembler****Syntax:** MOVE <ea> , <ea>**Attributes:** Size = (Byte, Word, Long)**Description:** Moves the data at the source to the destination location and sets the condition codes according to the data. The size of the operation may be specified as byte, word, or long. Condition Codes:

X	N	Z	V	C
—	*	*	0	0

X — Not affected.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Always cleared.

C — Always cleared.

Instruction Format:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	SIZE		DESTINATION						SOURCE					
				REGISTER		MODE				MODE		REGISTER			

Instruction Fields:

Size field—Specifies the size of the operand to be moved.

01 — Byte operation

11 — Word operation

10 — Long operation

MOVE

Move Data from Source to Destination
(M68000 Family)

MOVE

Destination Effective Address field—Specifies the destination location. Only data alterable addressing modes can be used as listed in the following tables:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	reg. number:Dn	(xxx).W	111	000
An	—	—	(xxx).L	111	001
(An)	010	reg. number:An	#<data>	—	—
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d ₁₆ ,An)	101	reg. number:An	(d ₁₆ ,PC)	—	—
(d ₈ ,An,Xn)	110	reg. number:An	(d ₈ ,PC,Xn)	—	—

MC68020, MC68030, and MC68040 only

(bd,An,Xn)*	110	reg. number:An	(bd,PC,Xn)*	—	—
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	—	—
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	—	—

*Can be used with CPU32.

MOVE

Move Data from Source to Destination
(M68000 Family)

MOVE

Source Effective Address field—Specifies the source operand. All addressing modes can be used as listed in the following tables:

Addressing Mode	Mode	Register	Addressing Mode	Mode	Register
Dn	000	reg. number:Dn	(xxx).W	111	000
An	001	reg. number:An	(xxx).L	111	001
(An)	010	reg. number:An	#<data>	111	100
(An) +	011	reg. number:An			
– (An)	100	reg. number:An			
(d ₁₆ ,An)	101	reg. number:An	(d ₁₆ ,PC)	111	010
(d ₈ ,An,Xn)	110	reg. number:An	(d ₈ ,PC,Xn)	111	011

MC68020, MC68030, and MC68040 only

(bd,An,Xn)**	110	reg. number:An	(bd,PC,Xn)**	111	011
([bd,An,Xn],od)	110	reg. number:An	([bd,PC,Xn],od)	111	011
([bd,An],Xn,od)	110	reg. number:An	([bd,PC],Xn,od)	111	011

*For byte size operation, address register direct is not allowed.

**Can be used with CPU32.

NOTE

Most assemblers use MOVEA when the destination is an address register.

MOVEQ can be used to move an immediate 8-bit value to a data register.

8/14-Pin, 8-Bit Flash Microcontrollers

Devices Included In This Data Sheet:

- PIC12F508
- PIC12F509
- PIC16F505

High-Performance RISC CPU:

- Only 33 Single-Word Instructions to Learn
- All Single-Cycle Instructions Except for Program Branches, which are Two-Cycle
- 12-Bit Wide Instructions
- 2-Level Deep Hardware Stack
- Direct, Indirect and Relative Addressing modes for Data and Instructions
- 8-Bit Wide Data Path
- 8 Special Function Hardware Registers
- Operating Speed:
 - DC – 20 MHz clock input (PIC16F505 only)
 - DC – 200 ns instruction cycle (PIC16F505 only)
 - DC – 4 MHz clock input
 - DC – 1000 ns instruction cycle

Special Microcontroller Features:

- 4 MHz Precision Internal Oscillator:
 - Factory calibrated to $\pm 1\%$
- In-Circuit Serial Programming™ (ICSP™)
- In-Circuit Debugging (ICD) Support
- Power-On Reset (POR)
- Device Reset Timer (DRT)
- Watchdog Timer (WDT) with Dedicated On-Chip RC Oscillator for Reliable Operation
- Programmable Code Protection
- Multiplexed MCLR Input Pin
- Internal Weak Pull-Ups on I/O Pins
- Power-Saving Sleep mode
- Wake-Up from Sleep on Pin Change
- Selectable Oscillator Options:
 - INTRC: 4 MHz precision Internal oscillator
 - EXTRC: External low-cost RC oscillator
 - XT: Standard crystal/resonator
 - HS: High-speed crystal/resonator (PIC16F505 only)

- LP: Power-saving, low-frequency crystal
- EC: High-speed external clock input (PIC16F505 only)

Low-Power Features/CMOS Technology:

- Operating Current:
 - < 175 μ A @ 2V, 4 MHz, typical
- Standby Current
 - 100 nA @ 2V, typical
- Low-Power, High-Speed Flash Technology:
 - 100,000 Flash endurance
 - > 40 year retention
- Fully Static Design
- Wide Operating Voltage Range: 2.0V to 5.5V
- Wide Temperature Range:
 - Industrial: -40°C to +85°C
 - Extended: -40°C to +125°C

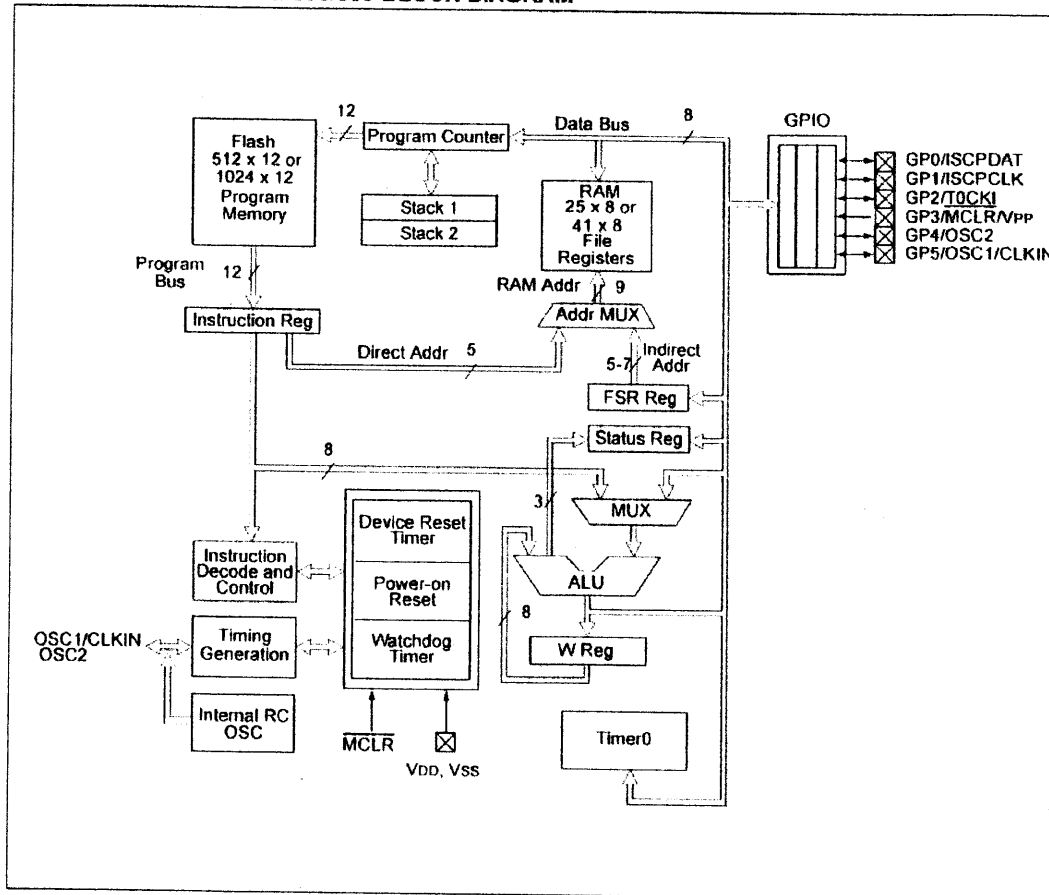
Peripheral Features (PIC12F508/509):

- 6 I/O Pins:
 - 5 I/O pins with individual direction control
 - 1 input only pin
 - High current sink/source for direct LED drive
 - Wake-on-change
 - Weak pull-ups
- 8-Bit Real-Time Clock/Counter (TMR0) with 8-Bit Programmable Prescaler

Peripheral Features (PIC16F505):

- 12 I/O Pins:
 - 11 I/O pins with individual direction control
 - 1 input only pin
 - High current sink/source for direct LED drive
 - Wake-on-change
 - Weak pull-ups
- 8-Bit Real-Time Clock/Counter (TMR0) with 8-Bit Programmable Prescaler

FIGURE 3-1: PIC12F508/509 BLOCK DIAGRAM



Device	Program Memory	Data Memory	I/O	Timers 8-bit
	Flash (words)	SRAM (bytes)		
PIC12F508	512	25	6	1
PIC12F509	1024	41	6	1
PIC16F505	1024	72	12	1