

Summary: Addressing Modes

1. Data or address register D_n / A_n
 - The respective data or address register is accessed directly
 - no memory access takes place
 - no effective address
2. Absolute address $address$
 - The address is part of the instruction
 - the effective address is the address in the instruction
 - this is a ‘memory operand’
 - the value of the addressed memory cell (B=byte, W=word, L=two words) is fetched or set
3. Immediate value $\#value / \#address$
 - The data is part of the instruction
 - this is not a ‘memory operand’
 - no effective address
 - cannot be a destination
4. Indirect on address register (A_n)
 - The effective address is given as the value of an address register
 - the memory cell indicated by the effective address is fetched or set
 - this is a ‘memory operand’
 - indirect addressing can be performed through registers only, not through memory cells
5. Autoincrement/-decrement $(A_n)+$
 $-(A_n)$
 - Identical to indirect mode except:
 - in postincrement mode, the value of A_n is incremented *after* the value has been fetched or stored
 - in predecrement mode, the value of A_n is decremented *before* the value is fetched or stored
 - value added to or subtracted from A_n equals number of bytes transferred
6. Indexed basic $d(A_n)$
 - Identical to indirect mode except:
 - effective address is calculated from the *base address* in a given address register A_n and a word-length displacement d
 - the effective address is $A_n + d$
 - no autoincrement/-decrement on A_n
 - d cannot be an absolute address

7. Relative basic

$d(PC)$

- Identical to indexed basic except:
- the base address is given by the program counter after instruction word
- the effective address is $PC + d$
- if d is a label indicating an absolute address, it is replaced by the difference of the address of this label and the current instruction address
- cannot be a destination

8. Indexed/Relative full

$d(An, Dk)$

$d(PC, Dk)$

$d(An, Ak)$

$d(PC, Ak)$

- These modes are identical to their respective indexed and relative basic modes, except:
- in addition to the base address given in an address register or the program counter, a second index register is specified
- the effective address is $\{An, PC\} + \{Dk, Ak\} + d$
- the displacement d is restricted to byte-length values
- the relative full mode cannot be specified as a destination

The abbreviation E in the instruction descriptions indicates that usually all addressing modes can be used for this operand

Summary: Instruction Set

The major instructions of the 68000:

1. Data handling:

MOVE $.[BWL] E_1, E_2$

basic instruction to load and store data between registers and memory, two registers, or two memory locations

MOVEA $.[WL] E_1, An$

to be used if the destination is an address register

MOVEQ $.[L] \#v, Dn$

to load a byte value into a data register for initialization

MOVEM $.[WL] <reglist>, E$

MOVEM $.[WL] E, <reglist>$

to save and restore a set of register values in memory or on a stack

2. Arithmetic instructions:

ADD $.[BWL] E, Dn / Dn, E$

SUB $.[BWL] E, Dn / Dn, E$

addition and subtraction, where one operand must be a data register

CMP *.[BWL] E, Dn*
 performs a subtraction but sets the conditions codes only without storing the results (compare)

ADDA *.[WL] E, An*

SUBA *.[WL] E, An*

CMPA *.[WL] E, An*
 add to, subtract from, compare with address register

ADDI *.[BWL] #v, E*

SUBI *.[BWL] #v, E*

CMPI *.[BWL] #v, E*
 add, subtract, compare immediate

ADDQ *.[BWL] #v, E*

SUBQ *.[BWL] #v, E*
 add or subtract quick with $v \leq 8$

MULS *E, Dn*

MULU *E, Dn*
 signed and unsigned multiplication (sources have word length, but result is long word!)

DIVS *E, Dn*

DIVU *E, Dn*
 signed and unsigned division (note special representation of the result)

EXT *.[WL] Dn*
 arithmetic extend from byte→word or word→long word

3. Bit and word operations:

AND *.[BWL] E, Dn / Dn, E*

OR *.[BWL] E, Dn / Dn, E*

EOR *.[BWL] Dn, E*
 bit-wise logical \cdot , $+$, \oplus operations

ANDI *.[BWL] #v, E*

ORI *.[BWL] #v, E*

EORI *.[BWL] #v, E*
 bit-wise logical \cdot , $+$, \oplus immediate

ROL, ROR *.[BWL] #v, Dn / E*
 rotate $v \leq 8$ bits of Dn or one bit of E to the left/right without using X

ROXL, ROXR *.[BWL] #v, Dn / E*
 rotate $v \leq 8$ bits of Dn or one bit of E including X as additional bit

LSL, LSR *.[BWL] #v, Dn / E*
 logical shift (includes sign-bit), store lost bit in X and C

ASL, ASR *.[BWL] #v, Dn / E*
 arithmetic shift (retain sign-bit), store lost bit in X and C

SWAP *.[W] Dn*
 swap word halves of Dn

4. Condition testing:

TST *.[BWL] E*
set N and Z according to value *E*

BTST *.[BL] Dn, E*

BTST *.[BL] #v, E*
set Z from bit number *Dn/#v* in *E*

5. Program flow control:

JMP *E*
unconditional branch, with absolute, indirect, indexed, relative addressing allowed

BRA *d(PC)*

BRA.S *d(PC)*
unconditional branch, with relative addressing only, and short form for branches within $|d| < 128$ range

Bcc *d(PC)*

Bcc.S *d(PC)*
conditional branches, see table C.6 for cc

DBRA *Dn, d(PC)*
decrement *Dn* (word) by one and branch until $Dn = -1$

DBcc *Dn, d(PC)*
decrement *Dn* (word) by one and branch until $Dn = -1$ or the abort condition cc is true

6. Subroutine control:

JSR *E*
subroutine call, with absolute, indirect, indexed, and relative addressing allowed (return address pushed on stack)

BSR *d(PC)*

BSR.S *d(PC)*
subroutine call, with relative addressing only, and short form for $|d| < 128$ (return address pushed on stack)

RTS
return from subroutine
(address popped from stack)

LINK *An, #-d*
use *An* as frame pointer and
reserve *d* bytes on stack

UNLK *An*
release frame pointer *An*
and restore stack

Summary: Memory Usage

How many bytes for an 68000 instruction?

- The 68000 has 16 bit word size and 24 bit address space, thus one word fills two bytes, and an address specification needs two words
- Depending on the instruction, an *instruction word* is followed by optional *extension words* for each operand:
 1. Data or address register *none*
 2. Absolute address *2 ext. words*
 3. Immediate value *1 or 2 ext. words*
 4. Indirect on address register *none*
 5. Autoincrement/-decrement *none*
 6. Indexed/Relative basic *1 ext. word*
 7. Indexed/Relative full *1 ext. word*
- Absolute and immediate modes require 1 extension word for byte and word sizes, and 2 extension words for long sizes.

Extension Word Specialities

- Some instructions have an immediate operand included into the instruction word (thus no extension word for this value):
quick MOVEQ, ADDQ, SUBQ,
immediate ROL, ROR, ROXL, ROXR,
immediate ASL, ASR, LSL, LSR
- Short branches ± 127 bytes have the displacement included into the instruction word as well (no extension word)
- There is no short version for the DBRA and DBcc instructions (always require an extension word for the displacement)
- Some instructions need an additional extension word (e.g., MOVEM to specify the set of registers to be moved).

Memory Access Count

How many memory accesses for an instruction? Summing up:

- The fetching of the instruction word from the main memory is one access
- Each extension word requires one further access (i.e., for 24/32 bit operands \rightarrow two accesses!)

- Loading or storing of data is one access for byte and word values, but two accesses (higher and lower halves) for long values.
- If a memory operand is both source and destination, count *both* the accesses for reading the operand and writing the result.
- Direct access (either load or store) on the value of a register *never* involves main memory
- Indirect or indexed addressing however does; they can be performed on (address) registers only

Examples

Instruction	Bytes	Accesses
MOVE.L D0, D1	2	1
ADDQ.L #4, A1	2	1
ADDI.W #876, D5	4	2
MOVE.L D1, DATA	6	$3_{\text{INSTR}} + 2_{\text{DATA}}$
MOVE.L D1, (A1)	2	$1_{\text{INSTR}} + 2_{\text{DATA}}$
ADD.W 23(A1), D1	4	$2_{\text{INSTR}} + 1_{\text{DATA}}$
ADD.W D1, 23(A1)	4	$2_{\text{INSTR}} + 2_{\text{DATA}}$ [!]
SUB.L -7(A1, D3), D0	4	$2_{\text{INSTR}} + 2_{\text{DATA}}$
MOVE.W 23(PC), -2(A0, A3)	6	$3_{\text{INSTR}} + 2_{\text{DATA}}$
MOVE.L ABSA, ABSB	10	$5_{\text{INSTR}} + 4_{\text{DATA}}$
JMP LABEL	6	3
BRA -14(PC)	2	1
DBRA D7, -14(PC)	4	2
BRA 814(PC)	4	2
JSR SUB1	6	$3_{\text{INSTR}} + 2_{\text{STACK}}$
BSR 23(PC)	2	$1_{\text{INSTR}} + 2_{\text{STACK}}$
RTS	2	$1_{\text{INSTR}} + 2_{\text{STACK}}$