68000 Addressing Modes

→ Addressing modes are concerned with the way data is accessed
→ Addressing can be by actual address or based on a offset from a known position.
→ Theoretically, only absolute addressing is required; however, other addressing modes are introduced in order to improve efficiency.

1 Absolute Addressing:
   - Absolute Addressing uses the actual address of an operand;
     either a memory location (e.g., CLR.B $1234) or,
   - If a register is involved, this type is also called data register direct, e.g., MOVE.B D2,$2000

2 Immediate Addressing:
   - With Immediate Addressing, the actual operand is part of the instruction; e.g., MOVE.B #25,D2
68000 Addressing Modes

3 Address Register Indirect Addressing:
   - This addressing mode uses the 8 address registers.
   - These registers are assumed to contain the address of the data rather than the data itself. e.g. CLR.B (A0)

4 Address Register Indirect with Post-incrementing:
   - A variation of address register indirect in which the operand address is incremented after the operation is performed. The syntax is (Ai)+

5 Address Register Indirect with Pre-decrementing:
   - A variation of address register indirect in which the operand is decremented before the operation is performed. The syntax is -(Ai)
Address Register Indirect with Post-incrementing / Pre-decrementing

**Examples**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOVE.B (A0)+,D3</td>
<td>Byte data addressed by A0 are copied to D3. Then the contents of A0 are incremented by 1.</td>
</tr>
<tr>
<td>MOVE.W (A0)+,D3</td>
<td>The word data addressed by A0 are copied to D3. Then the contents of A0 are incremented by 2.</td>
</tr>
<tr>
<td>MOVE.L (A0)+,D3</td>
<td>The long word data addressed by A0 are copied to D3. Then the contents of A0 are incremented by 4.</td>
</tr>
<tr>
<td>MOVE.B -(A0),D3</td>
<td>The address stored in A0 is first decremented by 1. Then the data addressed are copied to D3.</td>
</tr>
<tr>
<td>MOVE.W -(A0),D3</td>
<td>The address stored in A0 is first decremented by 2. Then the data addressed are copied to D3.</td>
</tr>
<tr>
<td>MOVE.L -(A0),D3</td>
<td>The address stored in A0 is first decremented by 4. Then the data addressed are copied to D3.</td>
</tr>
</tbody>
</table>
# 68000 Instructions Summary

<table>
<thead>
<tr>
<th>Instr</th>
<th>Description</th>
<th>Instr</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABCD</td>
<td>Add decimal with extend</td>
<td>MOVE</td>
<td>Move source to destination</td>
</tr>
<tr>
<td>ADD</td>
<td>Add</td>
<td>MULS</td>
<td>Signed multiply</td>
</tr>
<tr>
<td>AND</td>
<td>Logical AND</td>
<td>MULU</td>
<td>Unsigned multiply</td>
</tr>
<tr>
<td>ASL</td>
<td>Arithmetic shift left</td>
<td>NBCD</td>
<td>Negate Decimal with Extend</td>
</tr>
<tr>
<td>ASR</td>
<td>Arithmetic shift right</td>
<td>NEG</td>
<td>Negate</td>
</tr>
<tr>
<td>Bcc</td>
<td>Branch conditionally</td>
<td>NOP</td>
<td>No operation</td>
</tr>
<tr>
<td>BCHG</td>
<td>Bit test and change</td>
<td>NOT</td>
<td>Ones complement</td>
</tr>
<tr>
<td>BCLR</td>
<td>Bit test and clear</td>
<td>OR</td>
<td>Logical OR</td>
</tr>
<tr>
<td>BRA</td>
<td>Branch always</td>
<td>PEA</td>
<td>Push effective address on stack</td>
</tr>
<tr>
<td>BSET</td>
<td>Bit test and set</td>
<td>RESET</td>
<td>Reset External devices</td>
</tr>
<tr>
<td>BSR</td>
<td>Branch to subroutine</td>
<td>ROL</td>
<td>Rotate left without extend</td>
</tr>
<tr>
<td>BTST</td>
<td>Bit test</td>
<td>ROR</td>
<td>Rotate right without extend</td>
</tr>
<tr>
<td>CHK</td>
<td>Check register against bounds</td>
<td>ROXL</td>
<td>Rotate left with extend</td>
</tr>
<tr>
<td>CLR</td>
<td>Clear operand</td>
<td>ROXR</td>
<td>Rotate right with extend</td>
</tr>
<tr>
<td>CMP</td>
<td>Compare</td>
<td>RTD</td>
<td>Return and deallocate</td>
</tr>
<tr>
<td>DBcc</td>
<td>Decrement and branch conditionally</td>
<td>RTE</td>
<td>Return from exception</td>
</tr>
<tr>
<td>DIVS</td>
<td>Signed divide</td>
<td>RTR</td>
<td>Return and restore</td>
</tr>
<tr>
<td>DIVU</td>
<td>Unsigned divide</td>
<td>RTS</td>
<td>Return from subroutine</td>
</tr>
<tr>
<td>EOR</td>
<td>Exclusive OR</td>
<td>SBCD</td>
<td>Subtract decimal with extend</td>
</tr>
<tr>
<td>EXG</td>
<td>Exchange registers</td>
<td>SCC</td>
<td>Set conditional</td>
</tr>
<tr>
<td>EXT</td>
<td>Sign extend</td>
<td>STOP</td>
<td>Stop</td>
</tr>
<tr>
<td>JMP</td>
<td>Jump</td>
<td>SUB</td>
<td>Subtract</td>
</tr>
<tr>
<td>JSR</td>
<td>Jump to subroutine</td>
<td>SWAP</td>
<td>Swap data register halves</td>
</tr>
<tr>
<td>LEA</td>
<td>Load Effective Address</td>
<td>TAS</td>
<td>Test and set operand</td>
</tr>
<tr>
<td>LINK</td>
<td>Link stack</td>
<td>TRAP</td>
<td>Trap</td>
</tr>
<tr>
<td>LSL</td>
<td>Logical shift left</td>
<td>TRAPV</td>
<td>Trap on overflow</td>
</tr>
<tr>
<td>LSR</td>
<td>Logical shift right</td>
<td>TST</td>
<td>Test</td>
</tr>
</tbody>
</table>
Classification of 68000 Instructions

- Data Movement Instructions
- Compare Instructions
- Branch Instructions:
  - Conditional
  - Unconditional
- Special Instructions for Address Registers
- Arithmetic Instructions
- Logic Instructions
- Bit Manipulation Instructions
- Stack and Subroutine Related Instructions
Data Movement Instructions

- A total of 13 instructions in all:
  
  MOVE, MOVEA, MOVE to CCR, MOVE to SR, MOVE from SR, MOVE USP, MOVEM, MOVEQ, MOVEP, LEA, PEA, EXG, SWAP

- MOVE copies data from one location to another and may be qualified by ".B" to move 8 bits; ".W" to move 16 bits; and ".L" to move 32 bits.

- MOVE does not change the source location only the destination location.

- MOVE updates the CCR as follows:
  
  - N  Set (=1) if the result (destination) is negative, cleared (=0) otherwise.
  - Z  Set if the result is zero, cleared otherwise.
  - V  Always cleared.
  - C  Always cleared.
  - X  Not affected.

- Examples:
  
  MOVE.B D1,D2  Register to register
  MOVE.B D1,1234 Register to memory
  MOVE.B 1234,D1 Memory to register
  MOVE.B 1234,2000 Memory to memory
  MOVE.B #4,D0 Literal to register
  MOVE.B #4,1234 Literal to memory
MOVE  Move 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:

<table>
<thead>
<tr>
<th>X</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>*</td>
<td>*</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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.
Data Movement Instructions

MOVEA
Copies a source operand to an address register. May use only ".W" and ".L". In the case of ".W", the source operand is sign extended. No effect on CCR
- Source Operand: All addressing modes.
- Destination Operand: Address register direct.

Move to CCR
Copies the lower order byte of the operand to the CCR register.
- Source Operand: All except address register direct.
- Destination Operand: CCR register.

EXG
Exchanges the entire 32-bit contents of two registers. Frequently used to copy an address register to a data register for processing. No effect on CCR.
- Source Operand: Address or data register.
- Destination Operand: Address or data register.
MOVEA

Move Address
(M68000 Family)

Operation: Source → Destination

Assembler Syntax: MOVEA < ea > ,An

Attributes: Size = (Word, Long)

Description: Moves the contents of the source to the destination address register. The size of the operation is specified as word or long. Word-size source operands are sign-extended to 32-bit quantities.

Condition Codes:
Not affected.
Data Movement Instructions

SWAP

Exchanges the upper and lower order words of a data register.

- Source Operand: Data register
- Destination Operand: N/A
- CCR set according to the resulting register value.

LEA

Copies an effective address into an address register.

- Source Operand:
  All except data register, address register direct, address register indirect with pre-decrement or post-increment or immediate.
- Destination Operand: Address register
- No effect on CCR.
LEA
Load Effective Address
(M68000 Family)

Operation: \(< ea > \rightarrow An\)

Assembler Syntax: \(\text{LEA} < ea > , An\)

Attributes: \(\text{Size} = (\text{Long})\)

Description: Loads the effective address into the specified address register. All 32 bits of the address register are affected by this instruction.

Condition Codes:
Not affected.
# Load Effective Address, LEA Examples

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEA $0010FFFFF, A5</td>
<td>Loads the absolute address</td>
</tr>
<tr>
<td>LEA (A0), A5</td>
<td>Loads the contents of another address register</td>
</tr>
<tr>
<td>LEA (12, A0), A5</td>
<td>Loads the contents of an address register plus a displacement.</td>
</tr>
<tr>
<td>LEA (12, A0, D4.L), A5</td>
<td>Loads the contents of an address register plus a data register plus a displacement (used for index addressing).</td>
</tr>
</tbody>
</table>
Compare Instructions

- All compare instructions subtract the source operand, usually the contents of one register (or memory location) from the contents of the destination operand, usually another register (or memory location) in order to set the CCR (except the X-bit). The results of the subtraction are discarded.

- Compare instructions include the following:

  CMP   Source operand: Any of the addressing modes
         Destination: Must be a data register.

  CMPA  Source operand: Any of the addressing modes
         Destination: Must be an address register.

  CMPI  Source operand: An immediate value
         Destination: Any of the addressing modes except address register direct or immediate.

  CMPM  Compares one memory location with another
         Only addressing mode permitted is address register indirect with auto- incrementing.
Compare Instructions

**CMP <source>,<destination>**

- The compare instruction, CMP <source>,<destination>, subtracts the source operand from the destination operand and updates the bits of the condition code register (CCR), according to the result. The result of the subtraction is discarded.

- CMP or another compare instruction is usually followed immediately by a conditional branch (e.g., BEQ branch on zero, BNE branch on zero, BGT branch if greater than, BLT branch if less than, etc). Consider the high-level language construct:

```
MOVE.B X,D0
CMP.B Y,D0   Evaluate X - Y
BGE X_Bigger If X is greater or equal to Y branch
MOVE.B Q,P   IF X < Y THEN P = Q
BRA Exit      IF X >= Y THEN P = R
X_Bigger    MOVE.B R,P
Exit          STOP #$2700 Exit point for code-fragment
```
**CMP**

**Compare**

(M68000 Family)

**Operation:** Destination – Source → cc

**Assembler Syntax:** CMP < ea > , Dn

**Attributes:** Size = (Byte, Word, Long)

**Description:** Subtracts the source operand from the destination data register and sets the condition codes according to the result; the data register is not changed. The size of the operation can be byte, word, or long.

**Condition Codes:**

<table>
<thead>
<tr>
<th>X</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>—</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

- **X** — Not affected.
- **N** — Set if the result is negative; cleared otherwise.
- **Z** — Set if the result is zero; cleared otherwise.
- **V** — Set if an overflow occurs; cleared otherwise.
- **C** — Set if a borrow occurs; cleared otherwise.
CMPA

Compare Address
(M68000 Family)

Operation:  Destination – Source → cc

Assembler
Syntax:    CMPA < ea > , An

Attributes:  Size = (Word, Long)

Description: Subtracts the source operand from the destination address register and sets the condition codes according to the result; the address register is not changed. The size of the operation can be specified as word or long. Word length source operands are sign- extended to 32 bits for comparison.

Condition Codes:

<table>
<thead>
<tr>
<th>X</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

X — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if an overflow is generated; cleared otherwise.
C — Set if a borrow is generated; cleared otherwise.
CMPM

Compare Memory
(M68000 Family)

Operation: Destination – Source → cc

Assembler Syntax: CMPM (Ay) + , (Ax) +

Attributes: Size = (Byte, Word, Long)

Description: Subtracts the source operand from the destination operand and sets the condition codes according to the results; the destination location is not changed. The operands are always addressed with the postincrement addressing mode, using the address registers specified in the instruction. The size of the operation may be specified as byte, word, or long.

Condition Codes:

<table>
<thead>
<tr>
<th>X</th>
<th>N</th>
<th>Z</th>
<th>V</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

X — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.
V — Set if an overflow is generated; cleared otherwise.
C — Set if a borrow is generated; cleared otherwise.
Conditional Branch Instructions

- Identified by the mnemonic Bcc where "cc" represents the condition to be checked.
- General form: Bcc Address_Label
- If the condition is true, then control will branch to "Address_Label".
- No effect on condition codes.
- These instructions can be grouped according the type of condition being checked:
  - Instructions that depend on a single CCR flag:
    BNE  BEQ  BPL  BMI  BCC  BCS  BVC  BVS
  - Instructions for signed comparison:
    BGE  BGT  BLE  BLT
  - Instructions for unsigned comparison:
    (BHS or BCC)  BHI  BLS  (BLO or BCS)
# Conditional Branch Instructions Depending on A Single CCR Flag

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Instruction</th>
<th>Branch Taken If</th>
</tr>
</thead>
<tbody>
<tr>
<td>BNE</td>
<td>Branch on not equal</td>
<td>Z=0</td>
</tr>
<tr>
<td>BEQ</td>
<td>Branch on equal</td>
<td>Z=1</td>
</tr>
<tr>
<td>BPL</td>
<td>Branch on not negative</td>
<td>N=0</td>
</tr>
<tr>
<td>BMI</td>
<td>Branch on negative</td>
<td>N=1</td>
</tr>
<tr>
<td>BCC</td>
<td>Branch on carry clear</td>
<td>C=0</td>
</tr>
<tr>
<td>BCS</td>
<td>Branch on carry set</td>
<td>C=1</td>
</tr>
<tr>
<td>BVC</td>
<td>Branch on overflow clear</td>
<td>V=0</td>
</tr>
<tr>
<td>BVS</td>
<td>Branch on overflow set</td>
<td>V=1</td>
</tr>
</tbody>
</table>
## Conditional Branch Instructions

For Signed Comparison

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Instruction</th>
<th>Branch Taken If</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGE</td>
<td>Branch on greater than or equal</td>
<td>(N=1 AND V=1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OR (N = 0 AND V=0)</td>
</tr>
<tr>
<td>BGT</td>
<td>Branch on greater than</td>
<td>(N=1 AND V=1 AND Z=0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OR (N=0 AND V=0 AND Z=0)</td>
</tr>
<tr>
<td>BLE</td>
<td>Branch on less than or equal</td>
<td>Z=1 OR (N=1 AND V=0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OR (N=0 AND V =1)</td>
</tr>
<tr>
<td>BLT</td>
<td>Branch on less than</td>
<td>(N=1 AND V=0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OR (N=0 AND V=1)</td>
</tr>
</tbody>
</table>
# Conditional Branch Instructions

## For Unsigned Comparison

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Instruction</th>
<th>Branch Taken If</th>
</tr>
</thead>
<tbody>
<tr>
<td>BHS, BCC</td>
<td>Branch on higher than or equal</td>
<td>C=0</td>
</tr>
<tr>
<td>BHI</td>
<td>Branch on higher than</td>
<td>C=0 AND Z=0</td>
</tr>
<tr>
<td>BLS</td>
<td>Branch on less than or equal</td>
<td>C=1 AND Z=1</td>
</tr>
<tr>
<td>BLO, BCS</td>
<td>Branch on less than</td>
<td>C=1</td>
</tr>
</tbody>
</table>
Unconditional Branch Instructions

Two types of unconditional branch instructions:

BRA Address_Label
Branches to a statically determined address indicated by Address_Label

Examples: BRA START
BRA EXIT

JMP
Jump to an address that can be changed during execution

Examples: JMP (A0)
JMP D0
**BRA**

**Branch Always**  
*(M68000 Family)*

**Operation:**  
$PC + d_n \rightarrow PC$

**Assembler Syntax:**  
BRA < label >

**Attributes:**  
Size = (Byte, Word, Long*)  
*(MC68020, MC68030, MC68040 only)*

**Description:** Program execution continues at location (PC) + displacement. The program counter contains the address of the instruction word of the BRA instruction plus two. The displacement is a two's complement integer that represents the relative distance in bytes from the current program counter to the destination program counter. If the 8-bit displacement field in the instruction word is zero, a 16-bit displacement (the word immediately following the instruction) is used. If the 8-bit displacement field in the instruction word is all ones ($\text{FF}$), the 32-bit displacement (long word immediately following the instruction) is used.

**Condition Codes:**  
Not affected.
Special Instructions for Address Registers

• If an address register is specified as the destination operand, then the following address register instructions:

\[
\text{MOVEA, ADDA, SUBA, CMPA}
\]

must be used instead of MOVE, ADD, SUB and CMP, respectively.

• Address instructions only apply to words and long words.

• In the case of a word operation, the source operand is sign extended to a long word,
  - e.g., $0004 becomes $00000004 and $FFF4 becomes $FFFFFFF4.

• Address instructions do not change any of condition codes (bits of the CCR).
Example: Min(X,Y) Using Comparison

* This program demonstrates how to find the smaller of
* two numbers X and Y using the comparison operator.
* if (X <= Y) then
*   D0 := X
* else
*   D0 := Y
* X and Y are stored in memory and the result of the comparison is stored in
* register D0

ORG $400  Program origin
MOVE.B X,D0  Store X in D0
CMP.B Y,D0  Compare Y and D0
BLE  Exit  Branch if X <= Y
MOVE.B Y,D0  Otherwise, Y is smaller
STOP #$2700  Halt processor at end of program

ORG $1000
X DC.B 4
Y DC.B 5
END $400
Example: Comparing Two Memory Blocks

- This program compares two blocks of memory. If the memory is equal, then FF is stored in address register D0, otherwise, 00 is stored.

```
ORG    $400          Program origin
LEA    Block1,A0    Point to the beginning of memory block 1
LEA    Block2,A1    Point to the beginning of memory block 2
MOVE.W #Size,D0    Store the long word count in size

LOOP    CMPM.L   (A0)+,(A1)+  Compare the long words
         BNE    NotEq    Branch if not equal
         SUBQ.W #1,D0   Otherwise, decrement the count
         BNE             LOOP    Go back for another comparison
         CLR.L         D0     Two strings are equal so set
         MOVE.B #$FF,D0  D0 to FF
         BRA             Exit

NotEq    CLR.L    D0     Otherwise, set D0 to 00
Exit     STOP    #$2700

Size     EQU     2       Compare 2 words
ORG      $600

Block1   DC.L 'Bloc','1234'  Block 1
ORG      $700

Block2   DC.L 'Bloc','1234'  Block 2
END      $400
```
Example: Reversing a String

This program reverses the contents of a string.

– Address register A0 points to the beginning of the string
– Address register A1 points to the end of the string.
– These pointers move towards each other and swap bytes until the A0 pointer meets or passes A1.
– Register D0 is used for temporary storage.

```
ORG $400  Program origin
* This section moves the A1 pointer to the end (0 character) of the string
LEA String,A1  Point to the beginning of the string
Loop1 TST.B (A1)+  Move to the end of the string
    BNE Loop1  Until the EOS is encountered
    SUBA #1,A1  Back up to the EOS
```
Example: Reversing a String (Continued)

* This section swaps the bytes at the opposite ends and moves the
* pointers towards the middle of the string until they meet.

LEA String,A0 Make A0 point to the beginning

Loop2 MOVE.B -(A1),D0 Save the bottom byte
CMPA.L A1,A0 If A0 has reached or passed A1
BHS Exit Then the string is reversed
MOVE.B (A0),(A1) Move the top to the bottom byte
MOVE.B D0,(A0)+ Move the previously saved bottom byte to the top byte

BRA Loop2 Loop back for another byte

STOP #$2700

ORG $1000

String DS.B 128 Reserve up to 128 bytes for the string

END $400